SH24-5001-4
File No. S370/4300-50

Data Language/I Disk Operating System/ Virtual Storage (DL/I DOS/VS) Guide for New Users

Program Product

Program Number 5746-XX1

IBM

Fifth Edition (December 1983)

This edition, SH24-5001-4, is a major revision of SH24-5001-3. It applies to Version 1, Release 7 (Version 1.7) of Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS), Program Number 5746-XX1 and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information contained herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Summary of Changes

For a list of changes, see page iii.

Changes or additions are indicated by a vertical line to the left of the change.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to:

IBM Corporation
Dept 812BP
1133 Westchester Avenue
White Plains, NY 10604, USA

or

IBM Deutschland GmbH
Dept 3282
Schoenaicher Strasse 220
D-7030 Boeblingen, Federal Republic of Germany

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.

Summary of Changes for DL/I DOS/VS Guide for New Users

Summary of Changes for SH24-5001-4 Version 1.7

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

Interactive Utility Generation
This provides an interactive facility to assist with the generation of utility job streams.

IMF Enhancements
The Interactive Macro Facility (IMF) has been enhanced in several areas, including the support of the Documentation Aid Facility.

Documentation Aid
This provides an ease-of-use facility to document DL/I definitions that can be accessed directly by ISQL.

ISQL Extract Defines Utility
This utility uses the SQL/DS tables created by the DL/I documentation aid to create an ISQL routine containing EXTRACT DEFINE commands.

IMF Adaptation to ISPF
The Interactive Macro Facility (IMF) now runs on the Interactive System Productivity Facility (ISPF) program product, 5668-960.

Variable Length Index Source Segment
This allows an Index Source Segment of a DL/I Secondary Index to be variable in length.

Utilities Operational Improvements
Various modifications have been made to the DL/I utilities to permit tape rewind options, omission of partition dump, suppression of informational messages to SYSLOG, selectable creation of Secondary Indexes, automatic open of WORKFIL, and change accumulation data base specification.

HLPI Support of Boolean Operations
The Boolean AND and OR operators can now be used with the WHERE clause.

MPS Restart
MPS batch environment jobs can be restarted in the event of failure. This supports the use of VSE Checkpoint/Restart in conjunction with the DL/I Checkpoint.

HLPI Support of Key Feedback
KEYFEEDBACK and FEEDBACKLEN can be specified with GET commands to retrieve the key feedback area from the PCB.

Online Initialization Messages
Messages have been added to provide status information during online initialization. These messages include information concerning the status of DL/I logging, DL/I version currently being run, and the status of program isolation.

Miscellaneous
This manual also includes some miscellaneous corrections and updates to existing information.

Summary of Changes for SH24-5001-3 Version 1.6

This edition has been revised to reflect the features for the new users of DL/I DOS/VS, including the High Level Programming Interface (HLPI) and the ACCESS statement, a new way of defining data bases.

High Level Programming Interface
The HLPI is an easy-to-use method of coding the functions performed by a DL/I data base application program in a batch, multiple partition support (MPS) batch, or CICS/VS online environment. The HLPI is available only for application programs written in COBOL and PL/I programming languages. Use of the DL/I HLPI requires the CICS/VS EXEC translator for translation of the commands.

ACCESS Statement
The ACCESS statement is an ease-of-use tool whose purpose is to help the data base administrator more easily define data bases. It is an alternative to the index definition function previously supplied by the LCHILD and XDFIELD statements and the /SX and /CK name types of FIELD statements.

Miscellaneous
Other changes to this manual include the addition of a special chapter containing information for RPG II, deletion of Assembler language information, a brief description of the partial data base reorganization function, and miscellaneous corrections and updates to existing information.

Note: The HLPI, ACCESS statement, and application programs written in RPG are not supported by IMS/VS.

Summary of Changes for SH24-5001-2 as updated by SN24-5659 High Level Programming Interface ICR

This Technical Newsletter supports the DL/I High Level Programming Interface ICR (Independent Component Release) of DL/I DOS/VS Version 1.5. It contains information relating to the support of the DL/I High Level Programming Interface (HLPI).

Summary of Changes
for SH24-5001-2
as updated by SN24-5657
Interactive Macro Facility ICR

This Technical Newsletter supports the Version 1, Release 5 (Version 1.5) ICR (Independent Component Release) of DL/I DOS/VS. It contains information relating to the support of IMF (Interactive Macro Facility).

Miscellaneous corrections and updates also appear throughout this manual.

Summary of Changes
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.

Preface

DL/I DOS/VS (Data Language/I Disk Operating System/Virtual Storage) is a data base management control system. DL/I DOS/VS permits the writing of data independent applications and provides data base integrity. The DL/I DOS/VS system supports application programs written in COBOL, PL/I, RPG II, Assembler, and programs, which use the high level programming interface written in PL/I or COBOL applications. DL/I DOS/VS executes as an application program under DOS/VSE in the batch and MPS batch environments.

CICS/VS (Customer Information Control System/Virtual Storage) application programs may also use DL/I DOS/VS to access DL/I DOS/VS data bases in an online environment. In the CICS/VS online or DL/I MPS batch environments, DL/I DOS/VS permits concurrent scheduling of programs, which use DL/I DOS/VS, thereby allowing concurrent access by more than one user to the same or different data bases.

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

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 guide to designing simple data base structures, and to accessing 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 most often used 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 functions available in DL/I.
- 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.

This publication repeats certain information that is also presented in other DL/I DOS/VS publications to minimize references to those other manuals.

The new user is expected to be familiar with DL/I DOS/VS *General Information,* GH20-1246, before using this manual. The following IBM publications provide additional details about DL/I DOS/VS:

Related Publications

DL/I DOS/VS Application Programming: High Level Programming Interface, SH24-5009

DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces, SH12-5411

DL/I DOS/VS Data Base Administration, SH24-5011

DL/I DOS/VS Resource Definition and Utilities, SH24-5021

DL/I DOS/VS Application and Data Base Design, SH24-5022

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

DL/I DOS/VS Interactive Resource Definition and Utilities, SH24-5029

DL/I DOS/VS Recovery/Restart Guide, SH24-5030

Note: For information on the DL/I DOS/VS library, see the *DL/I DOS/VS Library Guide and Master Index,* GH24-5008.

The following VSE publications are referenced in this publication:

VSE/Advanced Functions System Control Statements, SC33-6095

VSE/Advanced Functions Application Programming: User's Guide, SC24-5210

VSE/Advanced Functions Application Programming: Reference, SC24-5211

For users of RPG II:

DOS/VS RPG II Language, SC33-6031

If you are writing application programs using DL/I DOS/VS in an online environment you should be familiar with the content of, and have access to, the following CICS/VS publications:

CICS/DOS/VS Installation and Operations Guide, SC33-0070

CICS/VS Application Programmer's Reference Manual (Command Level), SC33-0077

CICS/VS System/Application Design Guide, SC33-0068

CICS/VS Intercommunication Facilities Guide, SC33-0133

CICS/VS Recovery and Restart Guide, SC33-0135

References in this manual to DL/I mean DL/I DOS/VS unless otherwise noted.

Contents

Chapter 1: General Information

Introduction

Data Language/I DOS/VS (DL/I) is a data base management system that improves the DOS/VSE 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/VSE 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 makes 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 or in a CICS/VS partition under DOS/VSE. DL/I allows transaction processing programs accessing data bases to run in the teleprocessing environment provided by CICS/VS.
- It provides a subset of the batch data management facilities offered by IMS/VS.
- 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 COBOL, PL/I, Assembler, or RPG II. COBOL and PL/I programmers may use the DL/I high level programming interface commands to access the data base.

- 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.
- 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/VSE 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 or HLPI commands.
- 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 DBD (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 programs.

  These phases are stored in the core image library. For online execution, the CICS/VS system control functions load the phases 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 to 3375 or FBA (or any combination of these) can be made with no application program recompile.

Programming Languages

DL/I supports four programming languages: COBOL, PL/I, RPG II, and Assembler. DL/I provides a high level programming interface (HLPI) and a CALL interface. HLPI may be used only with COBOL and PL/I. Programs written in RPG II use the RQDLI (request DL/I) command. Programs written in COBOL, PL/I, and Assembler may use the CALL interface. These commands or calls retrieve, delete, add, and change 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 to the application program after every request. This information indicates successful or unsuccessful completion, and identifies the data base segment retrieved or processed.

This manual discusses primarily the COBOL and PL/I HLPI commands. (See *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces* for detailed information on the CALL interface.)

DL/I Execution Environments

DL/I executes in three environments: online, batch, and multiple partition support (MPS) batch. The online and MPS batch environments use the CICS/VS-DL/I DOS/VS interface.

Online Applications

The relationship between DL/I and an online application program are illustrated in Figure 1-1.

1. The CICS/VS Program Control Program passes control to the application program, which runs in the CICS/VS partition. The application program issues an HLPI command and control is passed to CICS/VS.
2. The CICS/VS EXEC Interface program passes the command to the DL/I EXEC Interface (HLPI) program unless the CICS/VS Execution Diagnostic Facility (EDF) is active. EDF is an interactive debug tool that allows you to see, in EXEC command format, what your program is doing at execution time. (See the *CICS/VS Application Programmer's Reference Manual (Command Level)* for further information on EDF.)
3. The DL/I EXEC (HLPI) Interface program builds DL/I calls from HLPI commands.
4. The DL/I analyzer decodes the call parameters into specific data base actions.
5. The DL/I action modules translate the data base requests into I/O requests to the appropriate data base.
6. DL/I logs any changes to the data base. Use of the CICS/VS journal is recommended although the DL/I log may be used.
7. The access method 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.

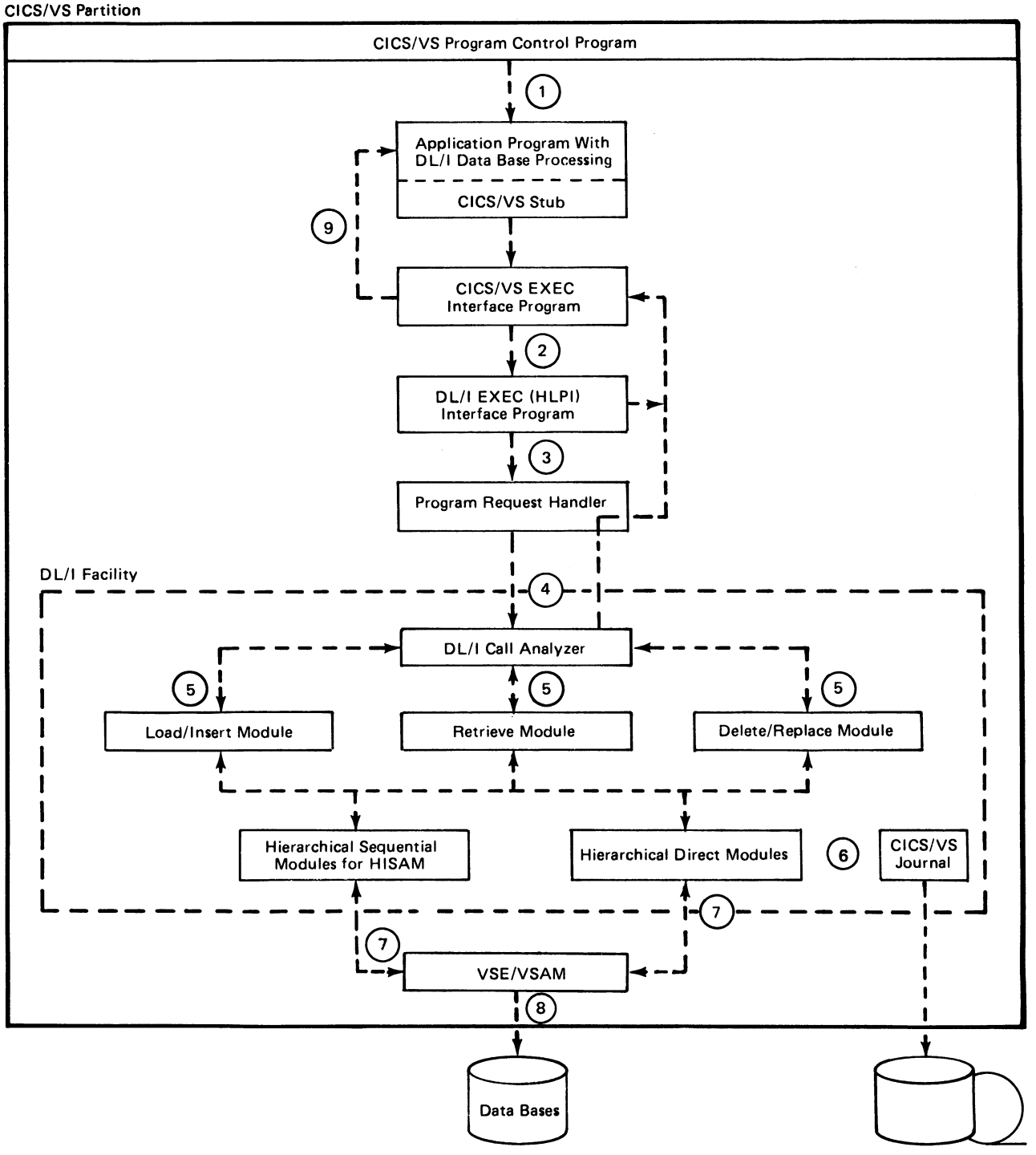


Figure 1-1. DL/I online system

Batch Applications

The relationship between DL/I and a batch application program is illustrated in Figure 1-2. The batch environment contains most of the functional parts listed for the online system except for the following:

1. VSE loads DL/I and gives control to DL/I. DL/I loads the PSB to determine the data base requirements of the application program. DL/I then loads and initializes the Data Management Blocks (DMBs). (Step 1)
2. DL/I loads and gives control to the application program. (Step 2)

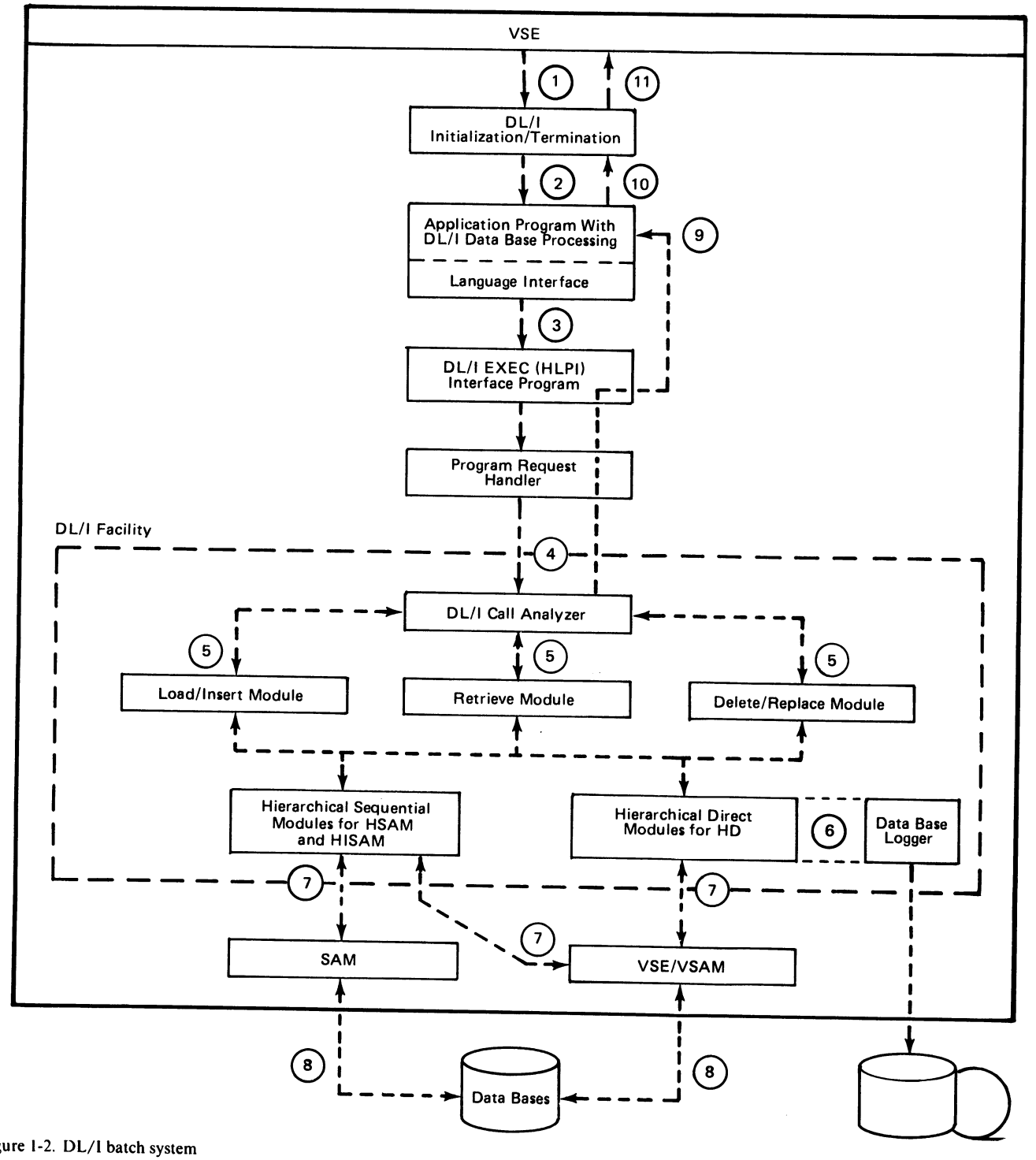


Figure 1-2. DL/I batch system

3-9. Steps 3 - 9 are the same for batch and online programs as described earlier except Step 6 where only DL/I disk or tape logging is permitted.

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 VSE.

Multiple Partition Support (MPS)

MPS uses the CICS/VS-DL/I DOS/VS interface. MPS batch requests are passed to the CICS/VS partition via the cross-partition event control facilities of VSE. These requests are executed by a mirror program, which issues the appropriate DL/I calls and passes the results back to the batch partition. All data base I/O is performed by the DL/I facility within the CICS/VS partition. MPS is transparent to the application program and, therefore, existing batch application programs (except for job control statements) need not be changed when used in this environment.

Utility Programs

DL/I supplies a number of utility programs that provide for the reorganization and recovery of a data base file. The use of the reorganization and recovery utilities is discussed in Chapters 7 and 8 of this manual.

File Integrity and Recovery

All modifications to any data base used in the DL/I online environment should be recorded on the CICS/VS system journal. Data base logging provides the DL/I system with a record of all modifications to all data bases used during program execution.

The CICS/VS system journal or the DL/I log may be used; it is recommended that the CICS/VS system journal be used in the online environment so that CICS/VS dynamic transaction backout can also be used. With the CICS/VS journal, the journal file contains DL/I log records and any other system-provided or user-provided CICS/VS journal records. DL/I log records may, in either case, be used by the DL/I data base recovery utilities to rebuild a data base. *DL/I DOS/VS Resource Definition and Utilities* provides additional detail on the use of data base log information for recovery.

Data Base Facility

Data Base Concepts

A data base is 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 primary unit of information in a data base processed by DL/I is called a segment. There may be 255 different segment types in a data base. For application programs to distinguish between them, each segment type must be named. There may be any number of occurrences of a particular segment type stored in the data base.

Most segments are composed of one or more data fields that are related and are normally processed together. A field can contain up to 256 bytes of data. The maximum size of a segment can vary from 4068 bytes to 32766 bytes, depending on the DL/I access method used.

There are several functions, described later in this book, that DL/I can perform with information stored in data bases. These functions include:

- *loading* (initially) segments into a data base
- *retrieving* segments from a data base
- *inserting* segments into an existing data base
- *replacing* segments in a data base
- *deleting* segments from a data base
- *checkpointing* to make sure changes to the data bases have been physically made
- *scheduling* the use of a data base in an online environment
- *terminating* the use of a data base in an online environment.

Segment Sensitivity

For data base security, each application program using DL/I can be allowed to access its own subset of the data base segments. Access to these segments, called segment sensitivity, is defined in the PSB that the application program uses during execution.

Field Level Sensitivity

In addition to segment sensitivity, DL/I provides field-level sensitivity. With field-level sensitivity, the user can define a new view of the segment for use by a particular application program. Field level sensitivity is also defined in the PSB.

Hierarchical Data Base Structure

Data base *records* are presented to the application programmer in a hierarchical segmented structure as illustrated in Figure 1-3. Each record in the data base (except for HSAM and SHSAM) must contain a *key* identifying that record. Data base records can be variable in length and contents, as required, and normally contain all the data logically related to a particular key. 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.

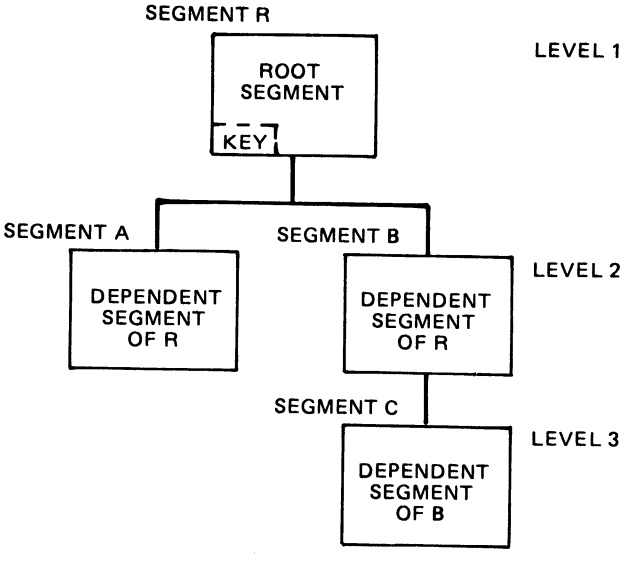


Figure 1-3. Hierarchical Data Structure

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. As shown in Figure 1-3, the existence of level 3 segments such as segment type C depend on the existence of the level 2 segment type B and cannot 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 the *parent* of segment C. Each segment type is fixed or variable length, contains logically associated fields, and has a one- to eight-character name.

File Record Layout

A customer file record layout might appear as shown in Figure 1-4.

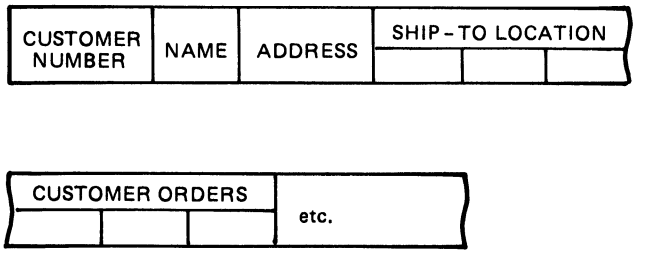


Figure 1-4. Customer File Record Layout

Figures 1-5 and 1-6 illustrate the format and contents of a customer data base record using a hierarchical data structure. Multiple occurrences of a segment type is illustrated by the presence of two ORDER ITEM segments for the September (770920) CUSTOMER ORDER segment (Figure 1-6). At times, a segment type may not have any occurrences. The hierarchical sequence of segments is top-to-bottom, left-to-right, and front-to-back. Thus, the sequential retrieval for the data base structure shown in Figure 1-5 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.

Note: The above numbers relate to the numbers along the right side of Figure 1-6.

Rules for Data Base Structures

The rules concerning data base structures are:

- Any number of data bases may be defined.
- A data base may consist of any number of data base records.
- A data base record may consist of 1 to 255 segment types (in Figure 1-5 there are 5 segment types). The segment type, CUSTOMER, is the root segment.

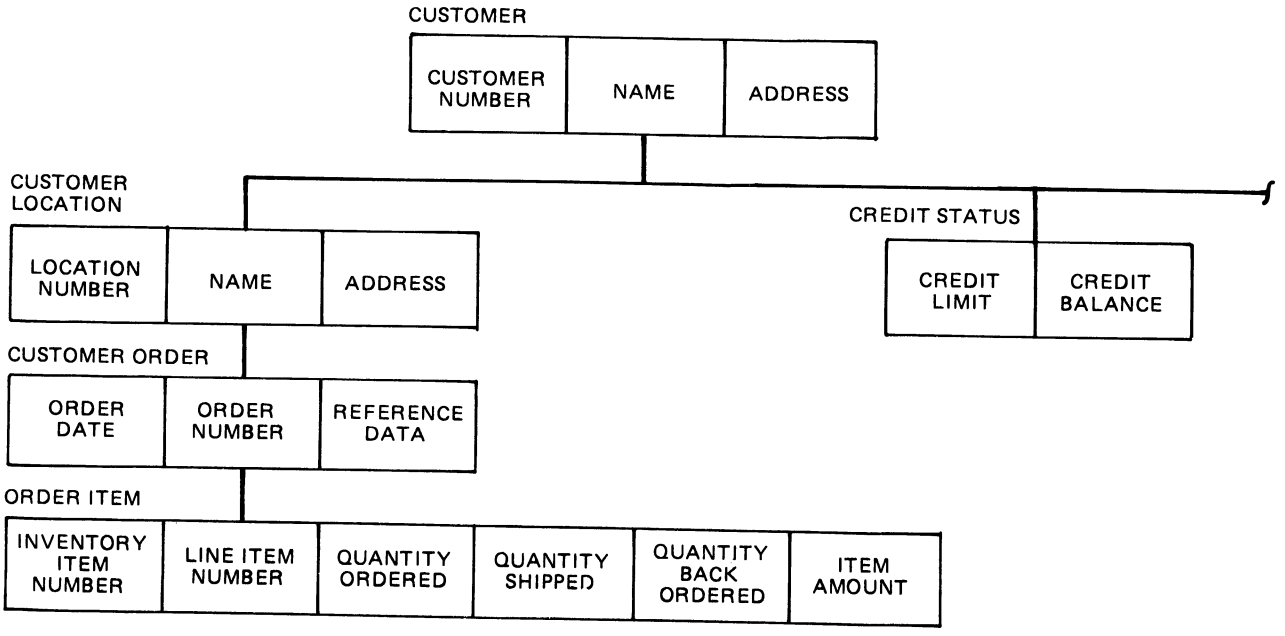


Figure 1-5. Customer Data Base

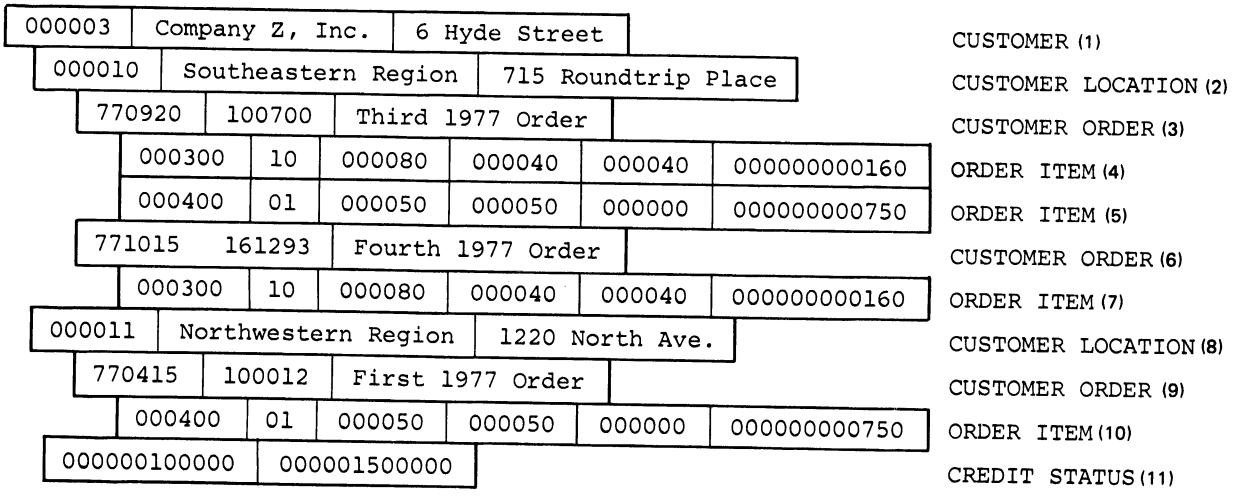


Figure 1-6. Customer Data Base -- Sample Record (Hierarchical Sequence)

- Within a data base record, each segment type may have any number of occurrences (including zero), 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 1-5 has 4 segment levels.)
- A dependent segment can exist only if its parent exists.
- Each segment type has a one- to eight-byte alphameric name.
- Each segment type can be either fixed, or for direct 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. This key field is used 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 have a sequence field. If no sequence field is defined, segment sequence is controlled by rules specified when describing the data base.

Adding New Segment Types

To modify the structure of a data base, you write a new DBD (data base description), and replace 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. This 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 DL/I statement, referred to as a high level programming interface (HLPI) command, from the COBOL or PL/I language program. For RPG II, your application program issues the RQDLI (request DL/I) command to access the data base. Details regarding the coding of HLPI commands 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 HLPI commands 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 IN PARENT)
- Replace the data in the existing segment (REPLACE)
- Delete an existing segment (DELETE)
- Insert a new segment (INSERT)
- Load segments (initially) into a data base (LOAD)
- Write a checkpoint record to the DL/I log (CHECKPOINT).

An HLPI command 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 commands
- Comparisons between fields within the segments in the specified path, and values supplied with the command.

The HLPI commands are independent of the data base access method.

Physical Data Base Structure

Five 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 five access methods are:

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

SHSAM and HSAM use the DOS/VSE 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/VSE 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 (HD)

HD consists of one entry sequenced file (VSAM ESDS) and, if primary access is by index, one key sequenced file (VSAM KSDS).

Basic Segment Types in a Hierarchical Data Structure

Figure 1-7 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 of that particular data base record. The key field for this root segment is the customer number.
- *Dependent Segment:* The dependent segment relies on higher level segments 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 requires 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 (except for sequential access) 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.*

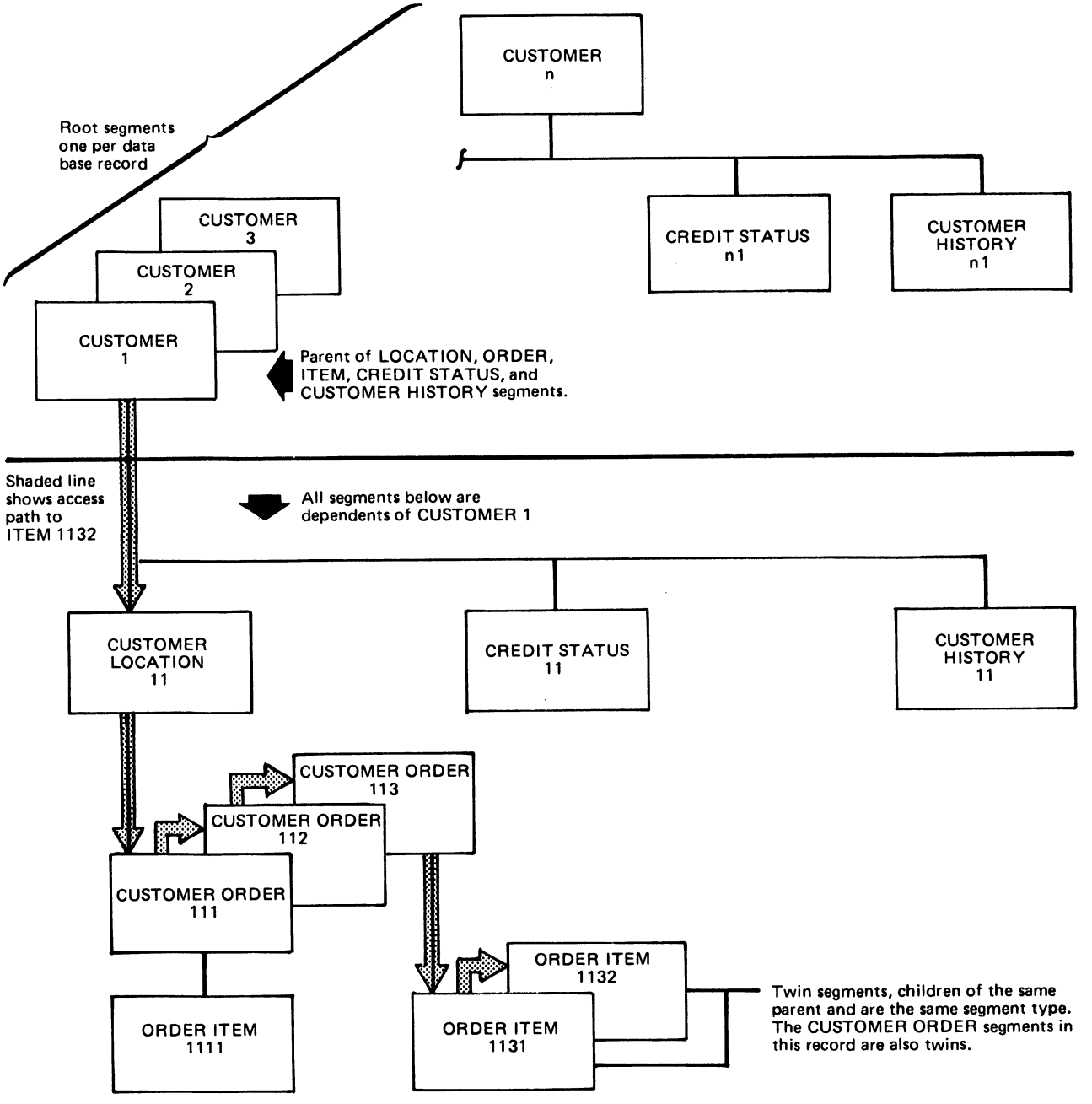


Figure 1-7. Segment Types and Their Relationships in a Hierarchical Data Structure

Figure 1-7 shows as a shaded line an example of a DL/I access path to the ORDER ITEM 1132 segment. It must always start with the root segment. 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 paths to the segments. The segments can belong to the same or different data bases, and form a *logical data base.* This logical data base presents a new hierarchical structure to the application program.

You build a logical relation by creating a dependent segment as a *logical child* that points to a second parent, the *logical parent.*

In Figure 1-8, 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 1-9 can be made available for application program processing. The ORDER ITEM/INVENTORY ITEM segment of the customer data base in Figure 1-9A, is a *concatenated segment.* It consists of the logical child segment plus the logical parent segment. The ORDER ITEM/CUSTOMER ORDER segment of the inventory data base in Figure 1-9B 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 1-8, 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 1-8, 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 deal with it separately in this manual. To show the use of logical relationships, we use phase 2 of the sample application as described in Chapter 2.

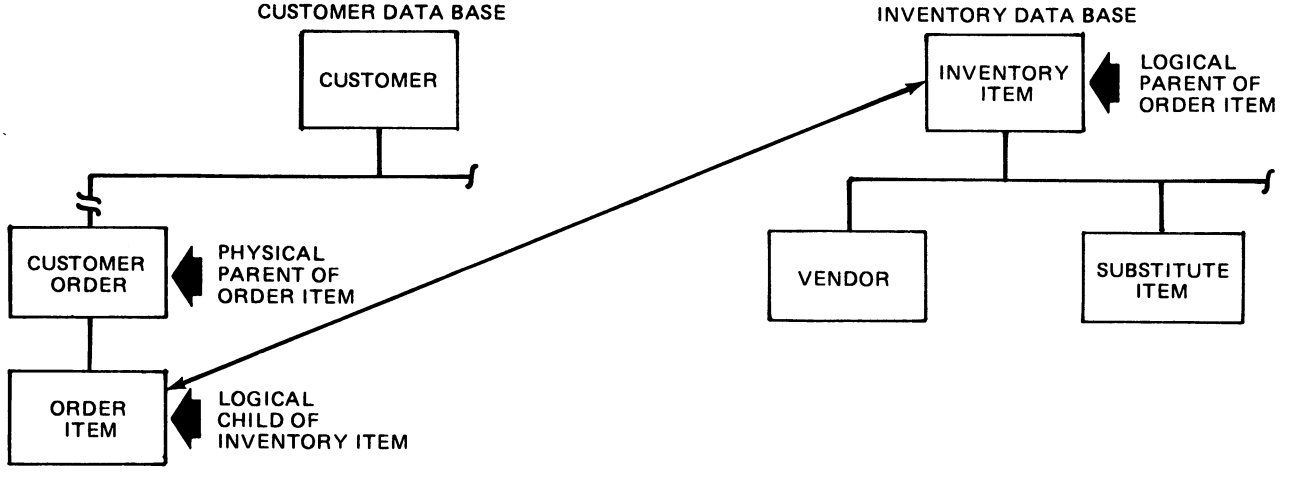


Figure 1-8. Two Logically Related Data Bases, CUSTOMER and INVENTORY

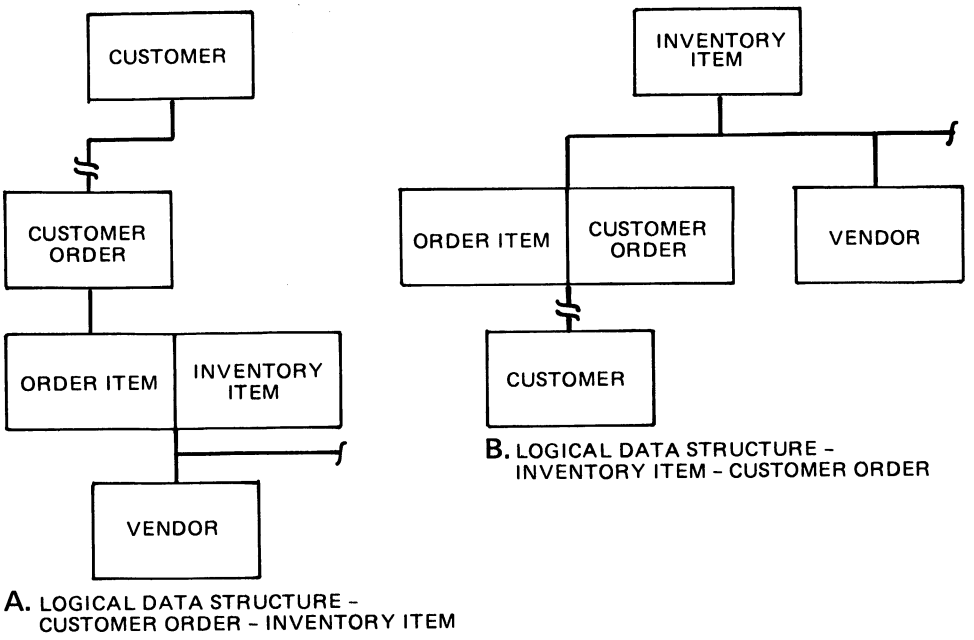


Figure 1-9. The Logical Data Bases After Relating CUSTOMER and INVENTORY Data Bases

Secondary Indexing

DL/I provides additional access flexibility with *secondary index data sets.* 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 1-10 can be retrieved based on the order number in the CUSTOMER ORDER segment, if an index were defined for that field. Once defined, DL/I automatically maintains 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 1-10:

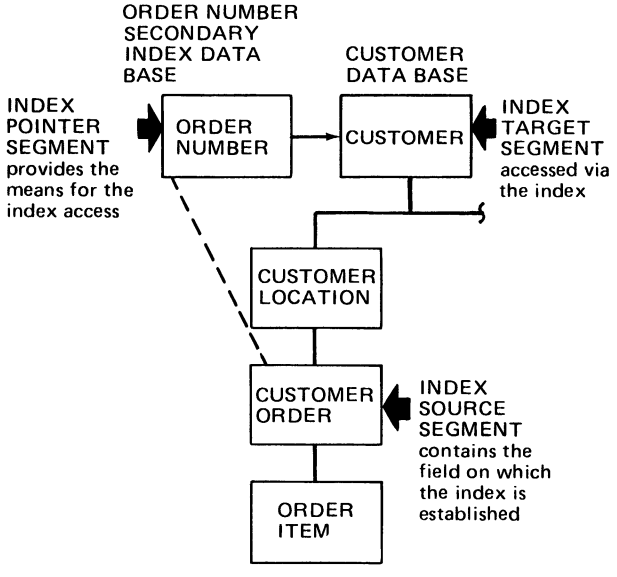


Figure 1-10. A Data Base and its Secondary Index

- 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 1-10.

In our examples, we 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 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) to provide 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 program. You provide the statements to define the hierarchical data structure and physical organization of the data base. These statements are assembled to produce a DBD.

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 program to the physical organization of the data used by VSE. The data structure can be remapped 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 two 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 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 data structure for each application program. The PSB defines which segments of the data base a specific program requires 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 online 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).

Using the Interactive Facilities

DL/I provides two easy-to-use interactive facilities which allow you to generate control blocks or job streams for each of the DL/I utilities at a 3270-type terminal. The Interactive Macro Facility (IMF) provides procedures to generate jobs that let you create, modify, and delete DL/I control blocks. The Interactive Utility Generation (IUG) facility permits you to generate job streams for the DL/I utilities.

Both procedures are implemented through the Interactive System Productivity Facility (ISPF) program product, 5668-960. This program product controls a series of menu-driven panels and prompts that are used to carry on a dialogue between you and the system. This dialogue takes place via display devices through the following forms:

Menu Panels
: give you a list of DL/I activities that you can do interactively. Each activity has an associated number and description. You select the activity you want by keying in the associated activity number.

Data Entry Panels
: prompt you to enter specific data for the activity you selected.

Help Panels
: contain helpful information associated with each menu or data entry panel. You may choose to view help panels at any time during a data entry session.

All the DL/I options that are available through the conventional method of coding DL/I macros to do DBD, PSB, and online nucleus generations are included in IMF. Similarly, IUG provides for interactive entry of all parameters required for a particular utility job stream. Both IMF and IUG, however, reduce the need for a detailed knowledge of either the macros or utilities.

For both IMF and IUG, data entry panels are displayed to get the required information. Parameters that are not necessary for a specific generation are either not shown or it is indicated that they are to be used only for certain conditions.

The information you provide is saved in files after data for each IMF macro or IUG panel is complete. If there is an abend during your data entry, only information for the most recent entry panel is lost and must be reentered.

If you enter information having incorrect syntax, such as a value outside the acceptable range, a descriptive error message appears on the data entry panel so that you can enter the correct data.

Each entry item on a data entry panel is identified by a name and brief description. Underscores or arrows indicate where you enter your reply. Your reply replaces the underscores. The underscores also indicate the maximum length for each data item. When available, IMF and IUG provide DL/I default values. In addition, after IMF or IUG is used to define a macro or generate a utility job stream, many other defaults are provided based on values that were entered in the previous activity. You can use or change all defaults.

Interactive Macro Facility (IMF)

IMF has three major functions: DBDGEN, PSBGEN, and ACTGEN.

DBDGEN
: gathers information you provide to create or update a DBD (data base description). Describing a data base is usually the first activity a new IMF user will do.

  The conventional method of coding DL/I macros to generate a DBD is described in Chapter 3 of this publication.

PSBGEN
: gathers information you provide to create or update a PSB (program specification block).

  The conventional method of coding DL/I macros to generate a PSB is also described in Chapter 3 of this publication.

ACTGEN
: gathers information you provide to create or update a DL/I online nucleus.

  The conventional method of coding DL/I macros to generate an online nucleus is described in Chapter 5 of this publication.

After you provide the information needed to complete any one of the above functions, IMF allows you to review or modify it before you request generation of DL/I control blocks.

IMF also allows you to migrate DBD and PSB tables created by the ELIAS-I program product to IMF format.

Interactive Utility Generation Facility (IUG)

IUG provides the following functions:

REORGANIZATION
: reorganizes data bases with or without using logical relationship utilities. You can also use this function to reorganize index data bases and for partial data base reorganization.

LOAD
: initially loads data bases with logical relationships and secondary indexes using logical relationship utilities. You can also use this function for simple initial loads of data bases where there are no logical relationship utilities.

RECOVERY
: initiates data base recovery job streams for Change Accumulation, Image Copy, Recovery, Backout, and Log Print.

PROBLEM DETERMINATION
: generates either the Log Print or Trace Print utility job stream.

EXTRACT DEFINES
generates a job stream for the Extract Defines utility.

RESUME INTERRUPTED JOB
resumes a job stream generation which was previously interrupted by a system failure or user intervention; processing will continue at the panel following the last one you successfully completed.

DATA BASE LABEL INFORMATION
permits you to add, modify, delete or review any data base label information entered through IUG.

DELETE OR DISPLAY DATA
permits you to delete tables containing DL/I utility job stream information which you no longer need or display data contained in IUG tables.

The functions of each of these utilities is further described in Chapters 7 and 8 of this publication. Detailed information on the conventional method of coding jobs for these utilities is contained in the *DL/I DOS/VS Resource Definition and Utilities* manual; interactive generation of jobs for these utilities is described in *DL/I DOS/VS Interactive Resource Definition and Utilities.*

User Responsibilities

System Installation

DL/I leaves you with 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 a data processing environment:
   - Data Bases
   - Batch Processing Programs
   - CICS/VS as the teleprocessing support for transaction processing programs

Data Base Administration

The centralization of and access control to 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 requires 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 an 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 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 a 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:

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

Other on-going activities include:

- Administration
- Documentation
- Technical support.

Figure 1-11 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.

Feasibility Study: 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 standards and guidelines is

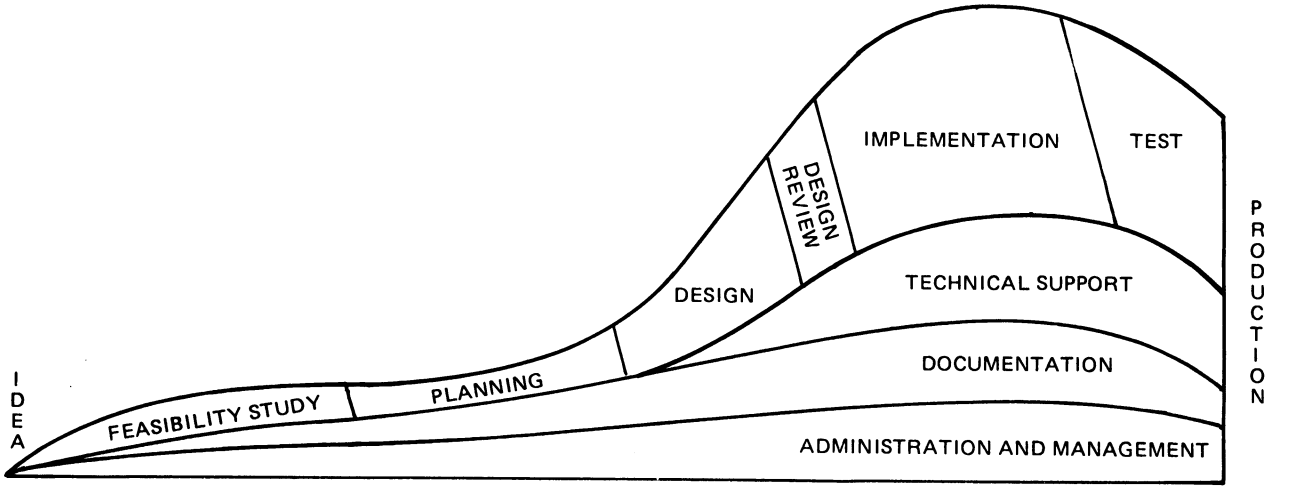


Figure 1-11. The Project Cycle

mandatory for an efficient project implementation and later maintenance. Most organizations already have standards which should be extended into the data base environment. Standards should be available for:

- Naming of data base items such as DBDs, PSBs, segments, fields, and so on.
- 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 1-12 shows a sample 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 because these are largely dependent upon the installation's environment. The following descriptions apply to the activities shown in the PERT chart (Figure 1-12).

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 1-13 shows an example of such a Gantt chart, which 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 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 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.

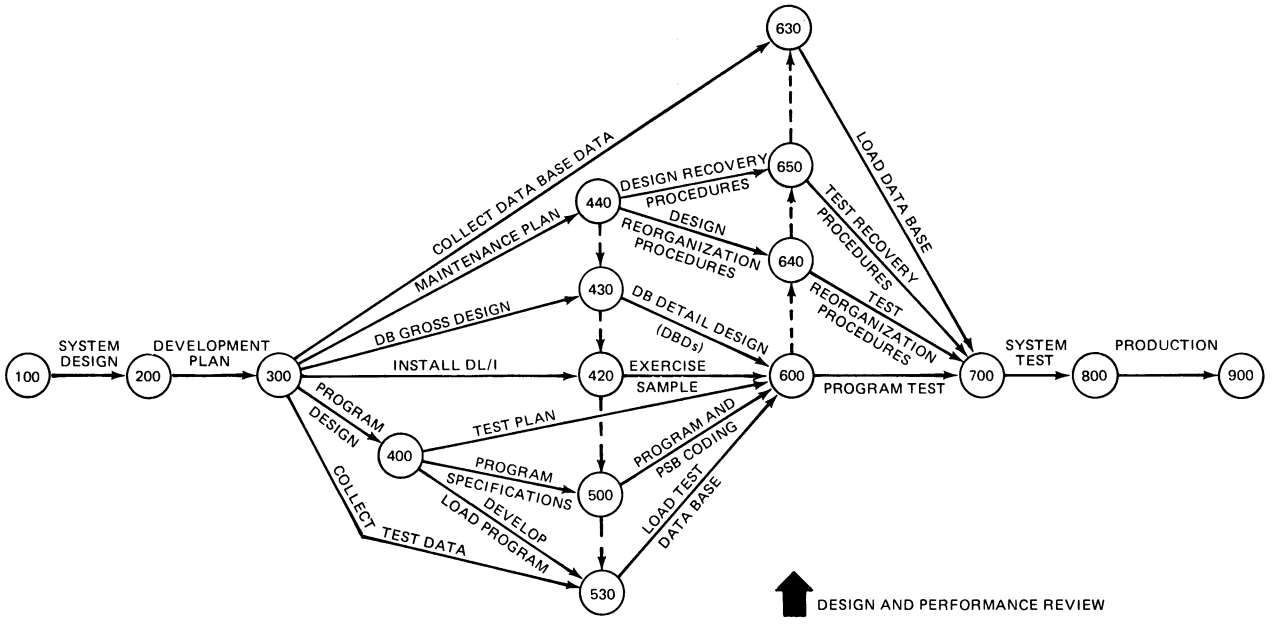


Figure 1-12. DL/I Installation Plan PERT Chart

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 assembled.

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:

- 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.

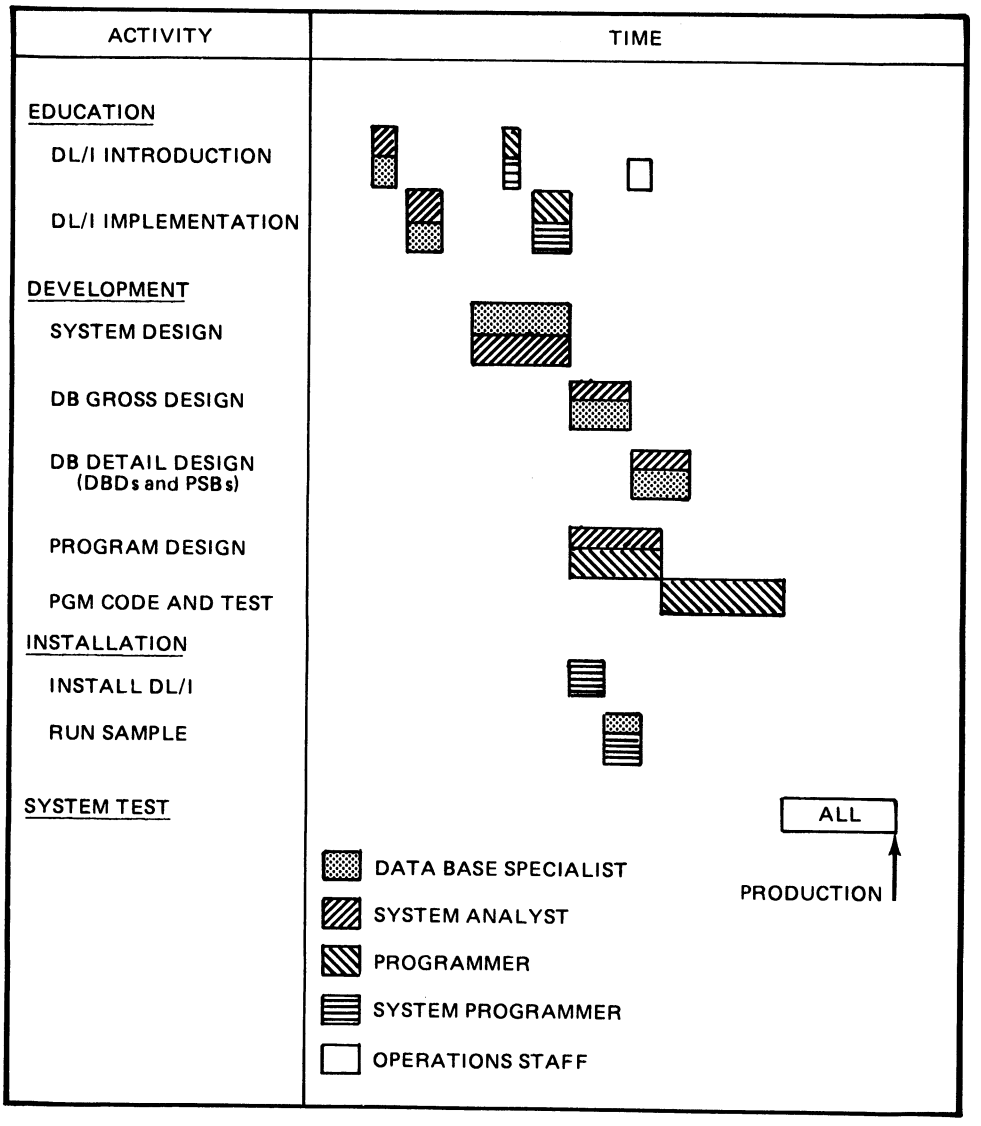


Figure 1-13. Sample Gantt Chart

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 benefits 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
- 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.

Chapter 2: Data Base Design

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.

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 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 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 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 also uses secondary index data bases.

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

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 satisfy the organization's application program data needs while satisfying an organization's data security, integrity, and redundancy objectives.

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 described in the next chapter.

This chapter introduces the concepts, techniques and guidelines for designing DL/I data structures. It is for those individuals 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 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.

Each user must determine the applicability and priority of the design objectives to the current design effort.

Detailed information about data base design can be found in *DL/I DOS/VS Application and Data Base Design.*

Section 1: DL/I Sample Application

The sample application 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, using DL/I.

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 has 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

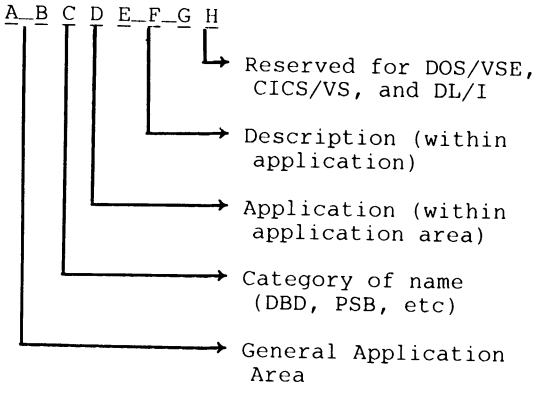
Information about customer orders is managed by the sales department. All order data will be stored in the Customer data base. It consists of one record for each customer order. Each record identifies:

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

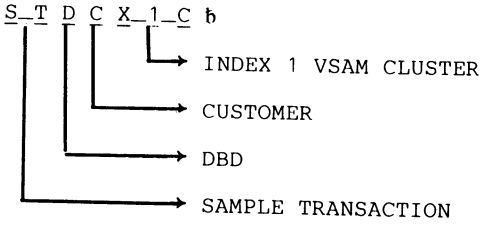
A link is required to the Inventory data base because it is necessary to know which parts are on order by each customer, and vice versa.

Naming Conventions Used in the Sample Application

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



Example:



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 - not used
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
                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)
```

Note: The names for the segments and data elements shown in all data base examples are as they will be used in the final (Phase 3) online application. Therefore some names used in the Phase 1 and Phase 2 examples are not consistent with the naming conventions described.

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.

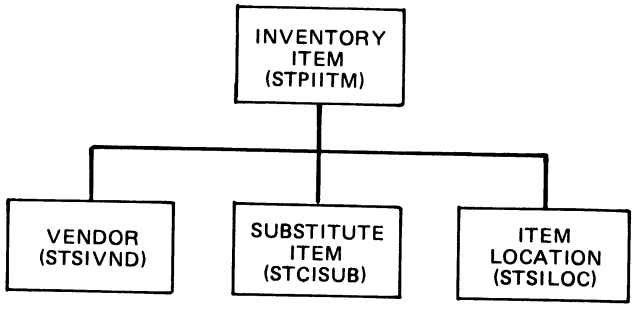


Figure 2-1. Inventory Data Base

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 decimal 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): 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): 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 data base 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.

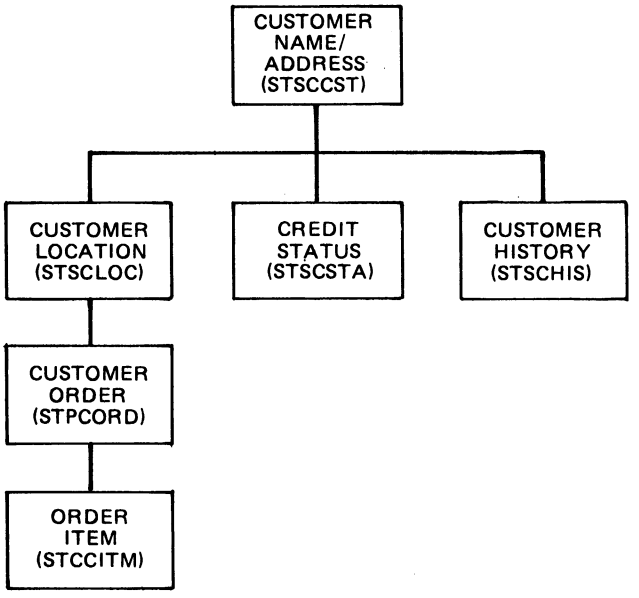


Figure 2-2. Customer Data Base

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	6
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, several sample data bases and sample programs are included to demonstrate the use of DL/I with logical relationships and secondary indexes, allowing 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 and logical 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 listed in Chapter 9.

These programs are interactive DL/I-CICS/VS online applications designed to allow customer order inquiry and customer order entry to the Customer and Inventory data bases defined for this sample application.

HLPI sample programs are written in COBOL (DLZCBL10, DLZCBL20, and DLZCBL30) and in PL/I (DLZPLI10, DLZPLI20, and DLZPLI30). The online sample application also

includes a program that defines the format of the displays to the 3270 screen as used by the online sample programs. See Chapter 9, "DL/I Online Sample Application," for more information.

Section 2: DL/I Data Base Facility

This section 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 Direct (HD)
- Simple Hierarchical Sequential Access Method (Simple HSAM)
- Hierarchical Sequential Access Method (HSAM)

Note: The Hierarchical Direct (HD) access method should be considered as your primary access method. The others are special purpose access methods; for example, Simple HISAM and Simple HSAM can be used as migration tools for existing application programs that will be run in a DL/I environment. (See Section 3 of this Chapter for a description of HD.)

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 first elaborates on some of the basic DL/I concepts introduced in Chapter 1.

DL/I Data Base Record

The DL/I data base record, 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.

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:

- Space reuse of deleted segments

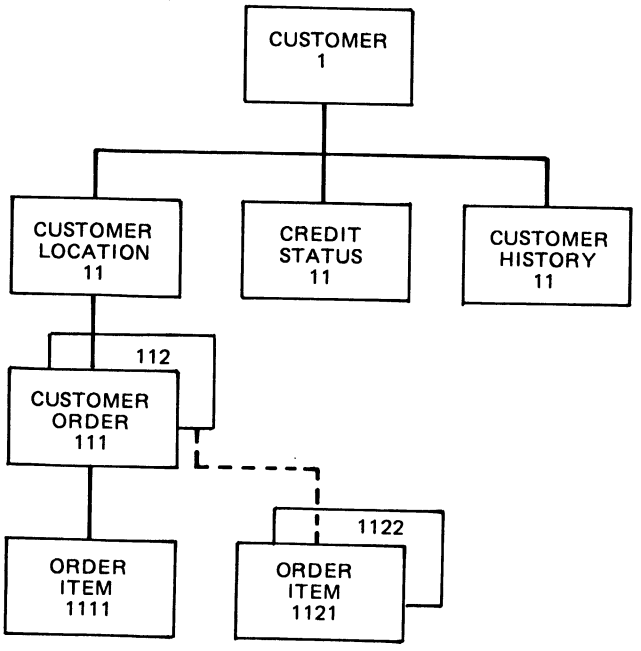


Figure 2-3. A DL/I Data Base Record

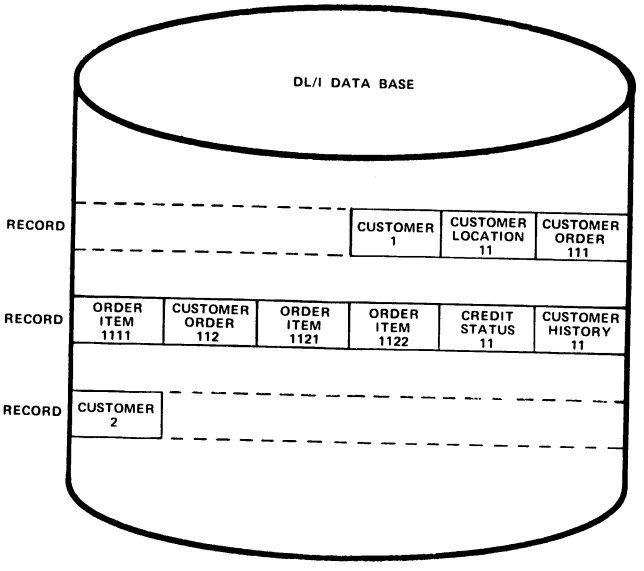


Figure 2-4. A DL/I Data Base Record in Physical Storage

- 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 HD data bases to link the segments within one data base record into 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.

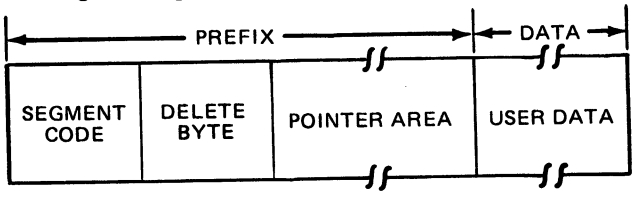


Figure 2-5. Segment Format

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.

Commands and Data Base Positioning

To better understand each particular data base organization, a basic description of the DL/I commands used to process segments in a data base follows.

The segments in a DL/I data base are processed through high level programming interface (HLPI) commands issued by an application program. Commands are issued to get, insert, delete, or replace a segment or a path of segments. A command includes all data required by DL/I to complete the function. Included in the command are a function and, optionally, one or more segment references. The function states the service to be performed, and the segment references define the hierarchical path down to, and including, the segment(s) to be processed. A brief description of DL/I commands follows. For more detailed information, see "Chapter 4: Processing Data Bases", and *DL/I DOS/VS Application Programming: High Level Programming Interface.*

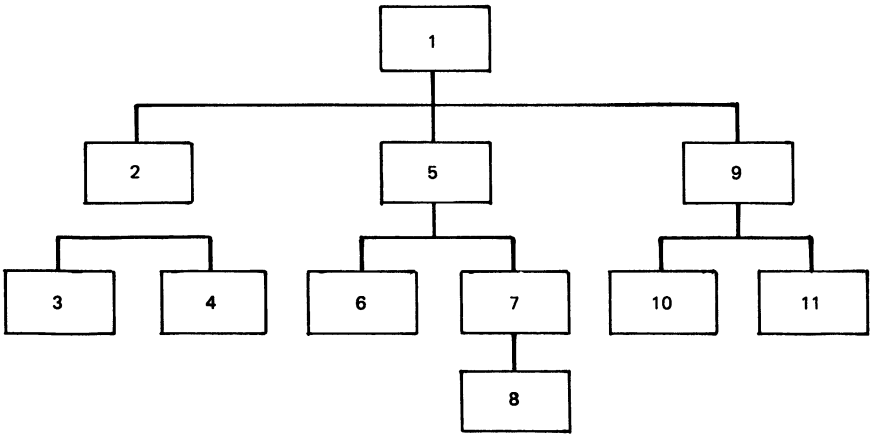


Figure 2-6. Segment Types Numbered in Hierarchical Sequence

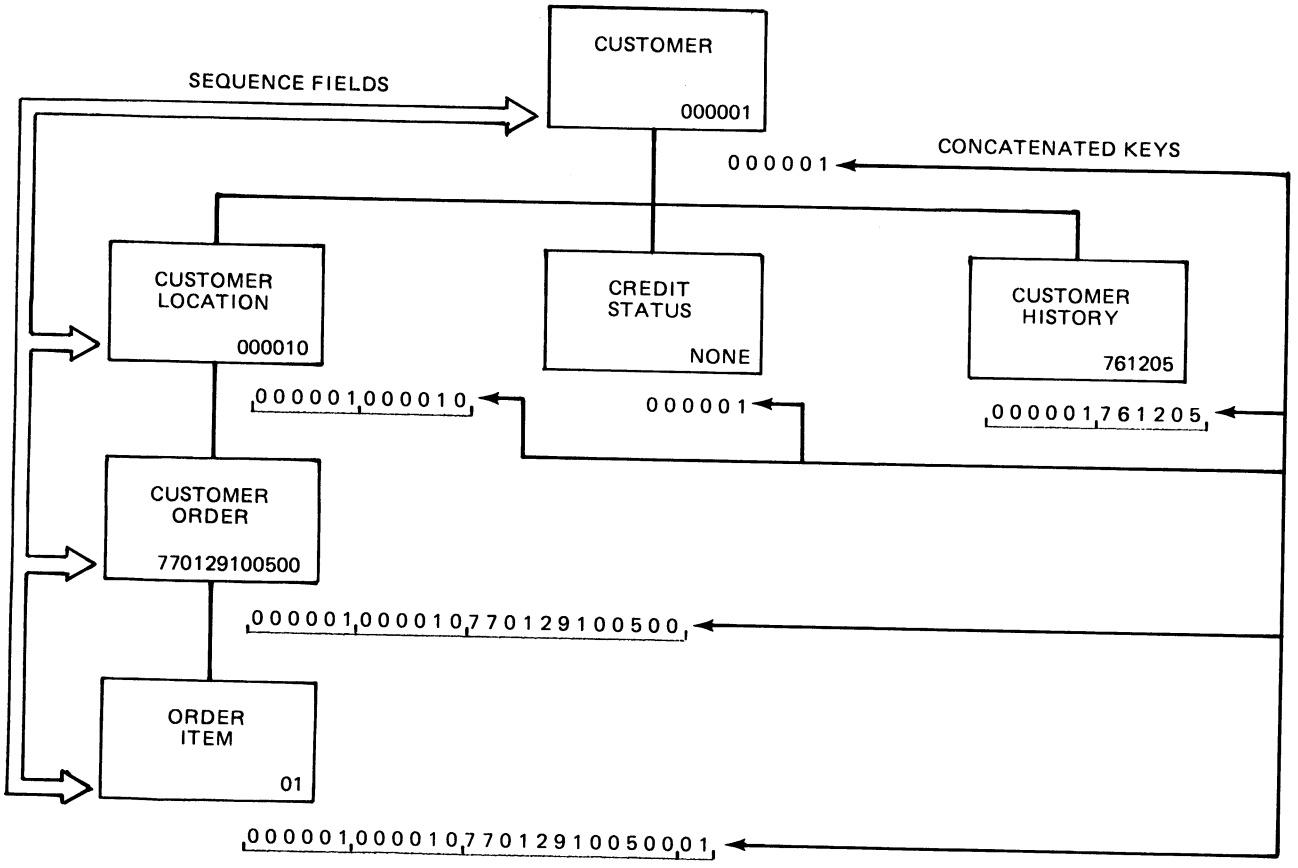


Figure 2-7. Concatenated Keys

The basic direction of movement in a DL/I data base is *top to bottom, left to right, front to back.* 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 commands are:

- GET UNIQUE is used to retrieve a specific segment, or path of segments, from a data base. At the same time, it establishes a position in a data base from which additional segments can be processed in a forward direction.
- GET NEXT is used to retrieve the next desired segment, or path of segments, from a data base. The GET NEXT command normally moves forward in the hierarchy of a data base from the current position. It can be modified to start at an earlier position than the current position in the data base through an option, but its normal function is to move forward from a given segment to the next desired segment in a data base. Options are discussed in Chapter 4.
- GET NEXT IN PARENT is used to retrieve the next desired segment or path of segments within established parentage. Parentage must have been established by a successful GET UNIQUE or GET NEXT command either immediately before this command or at some prior time, provided no other command that changes parentage has intervened. A GET NEXT IN PARENT does not establish parentage.
- INSERT is used to insert a segment or a path of segments into a data base. It is used to add segments to 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.

- DELETE is used to delete a segment from a data base. When a segment is deleted, its dependents, if any, are also deleted. This command must be preceded by a GET command.
- REPLACE 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 command. This command must be preceded by a GET command.
- LOAD is used to initially load segments into the data base. It cannot be used in the online environment.
- CHECKPOINT causes a checkpoint record to be written on the DL/I log as an aid in restart processing.

Segment Options

Segment options, when used, specify a functional variation of the command. Qualified segment selection identifies, through field values, the segment occurrence of the specified segment type. Qualified segment selection specifies 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 satisfies the command.

VSAM (Virtual Storage Access Method)

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

In the first, 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. 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
```

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. An ESDS is considered by VSAM to be a cluster without the index component.

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. Clusters are defined with the Access Method Services DEFINE command.

For information about VSAM, see the *VSE/VSAM General Information* manual, GC24-5143.

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 Sequential Access Method (SAM) file on DASD or tape. Any SAM file defined with RECFORM=FIXUNB may be defined as a simple HSAM data base.

HSAM

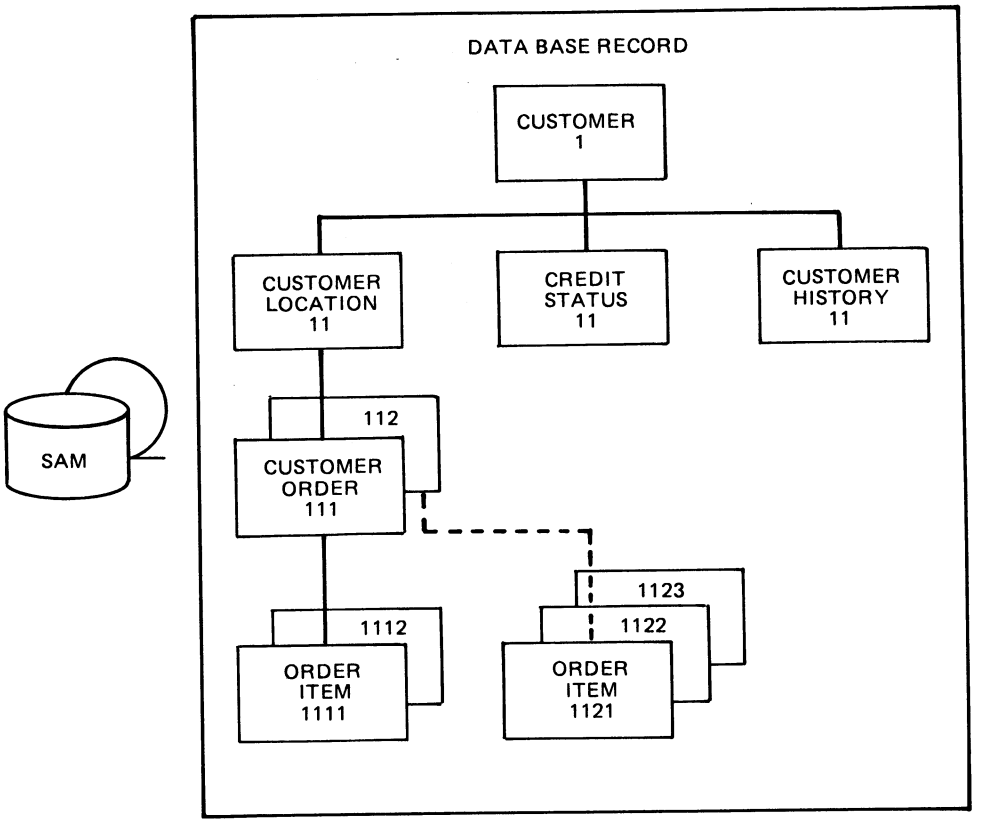
Figure 2-8 shows the HSAM physical storage of the logical data structure. HSAM uses the Sequential Access Method (SAM) data management facility for DASD and tape files. Segments, which contain DL/I prefix information and data, are placed sequentially in a physical record until the remaining space in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

SHISAM and HISAM Functions

- GET: SHSAM and HSAM will accept GET functions
- LOAD: SHSAM and HSAM *will* accept LOAD functions. Loads to an HSAM data base must be in sequence.
- INSERT: SHSAM and HSAM will *not* accept an insert function.
- DELETE: SHSAM and HSAM will *not* accept a delete function.
- REPLACE: SHSAM and HSAM will *not* accept a replace function.

Simple HISAM

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. 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 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.



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

HISAM

The HISAM data base access method is used for indexed sequential access. VSAM provides data management capabilities. 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 are 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,

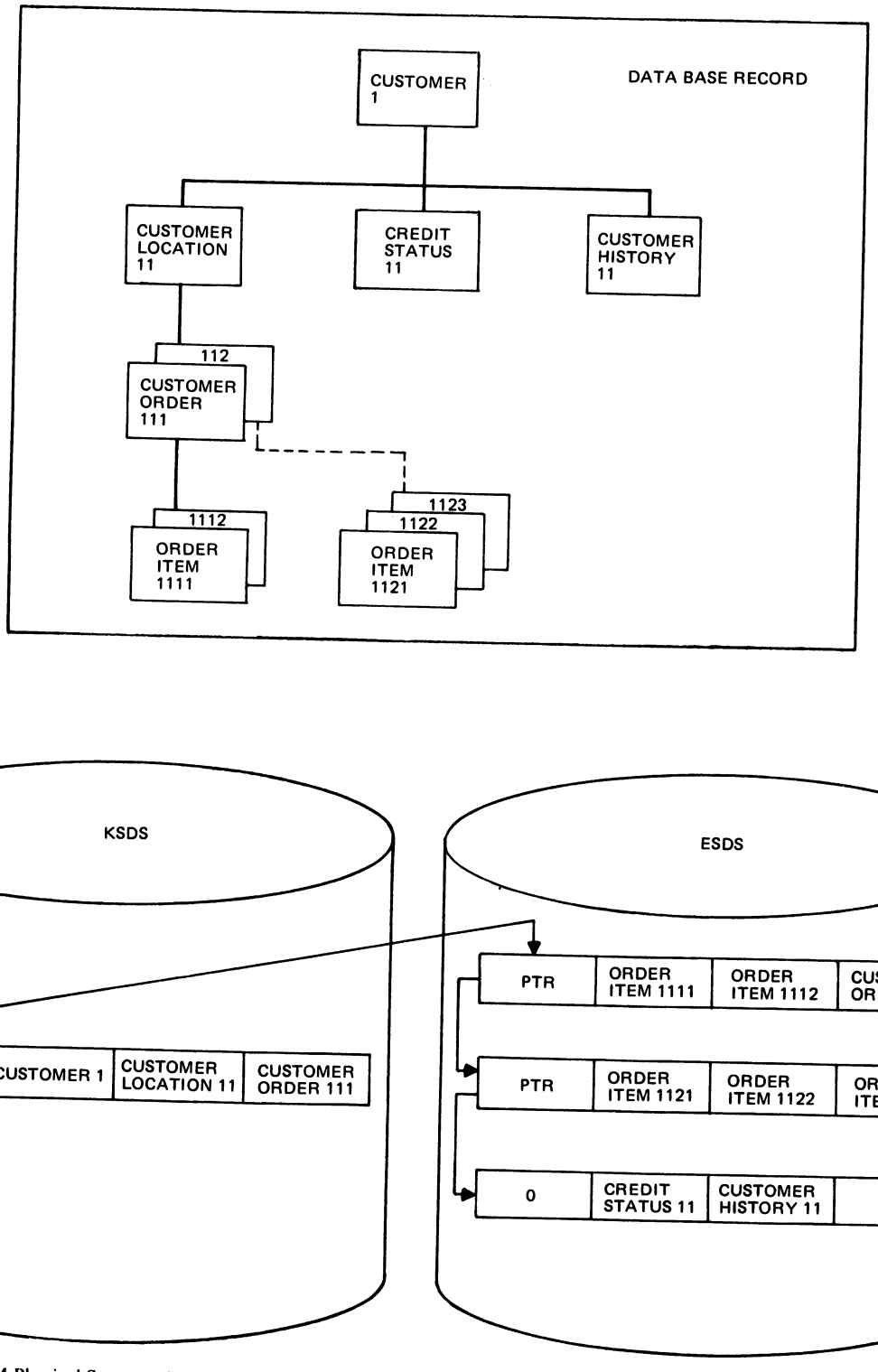


Figure 2-9. HISAM Physical Storage of a Data Base Record

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, INSERT, and REPLACE commands are not valid for HSAM.

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

HISAM data bases have these limitations:

- No support for variable length segments, secondary indexes, or logical relationships.
- Less efficient use of DASD space than direct organizations (for example, no space is reclaimed on delete processing). Space is reclaimed during a reorganization.

Direct (HD)

Direct access is the locating of any data base record in the data base directly, without searching sequentially through the records from the beginning. Direct access in DL/I is accomplished by using a randomizing routine or an index. DL/I can find any data base record that you want, independently of the sequence of data base records in the data base. Direct access methods can give good results with either direct or sequential processing.

HD Characteristics

Two primary advantages of an HD data base are space reuse and the ability to directly access segments within the data base.

The segment storage organization used for an HD data base is through a randomizing module or through an index. To access a given root segment, 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 through an indexed data base, the 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.

Randomized Access: Figure 2-10 shows that an HD data base with random access consists of one ESDS. To access the data, 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, the data base, into an address for the root segment.

The ESDS is divided into two areas:

- The *primary 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 primary 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 primary area. This parameter limits the number of segments of a data base record that can be consecutively inserted into the primary area.

When consecutively inserting a root and its dependents, each segment is stored in the primary 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. This parameter controls only segments consecutively inserted in one data base record. Consecutive inserts are inserts to one data base record with no intervening command to process a segment in a different data base record.

Indexed Processing: An indexed data base on auxiliary storage is actually comprised of two data bases; the HD primary index data base and the main HD data base. Only the main data base need be defined. DL/I automatically generates the definition for the HD primary index data base.

The HD primary index data base is used to locate the data base record stored in a data base. Under HD, two DBDs are generated: one for the index and one for the primary data base. When the main HD data base is defined at DBDGEN, a unique

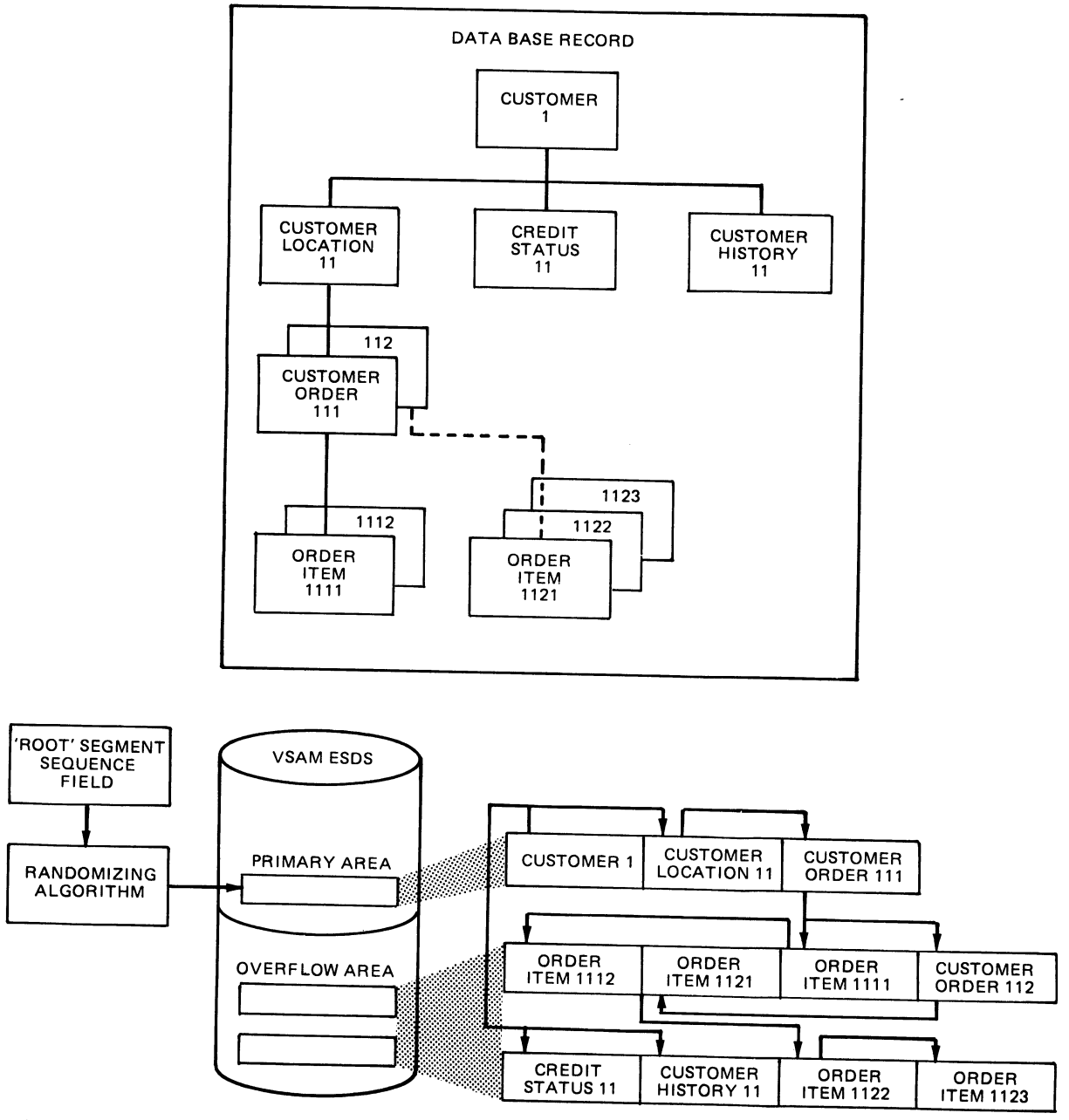


Figure 2-10. HD Data Base Record In Physical Storage For Randomized Processing

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 HD primary index data base contains, in its prefix, a pointer to the root segment in the main HD data base.

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

Inserts and Deletes in HD

The techniques used to insert or delete segments are the same for both HD randomized processing and HD indexed 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 after deletion 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. You can also specify free space in the DBD for an HD data base.

Direct Access Pointers in HD

Refer to Figure 2-12 for the following description of pointers.

Physical Child/Physical Twin Pointers: Every parent segment in the data base has a pointer to the first occurrence of each child segment type. This is the *physical child*

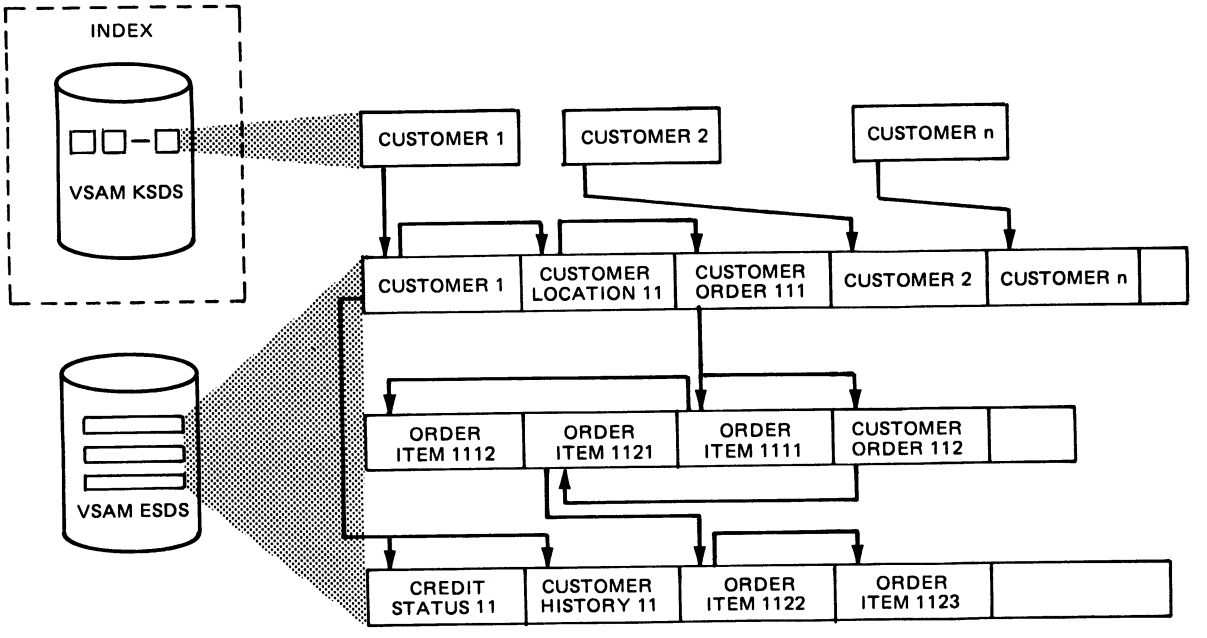


Figure 2-11. HD Indexed Data Base Record in Physical Storage

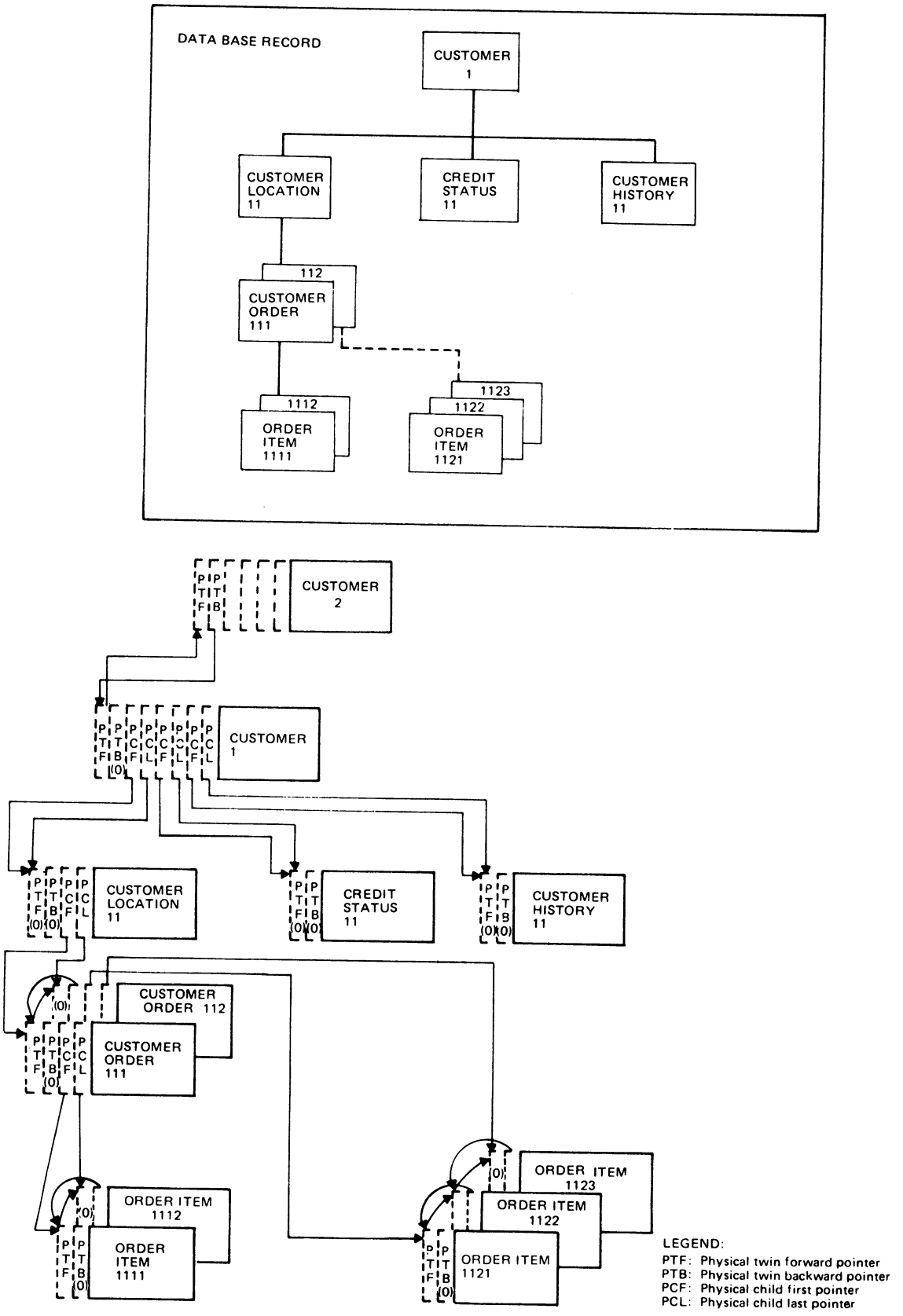


Figure 2-12. Direct Access Pointers in HD

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 improves segment insert performance of that child if that segment has no sequence field defined.

Every segment in an HD 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 an HD indexed data base, they enable sequential processing across data base records without intervening references to the index. When only physical twin forward pointers are specified for the root segment type of an HD indexed data base, sequential processing across data base records requires intervening references to the 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.

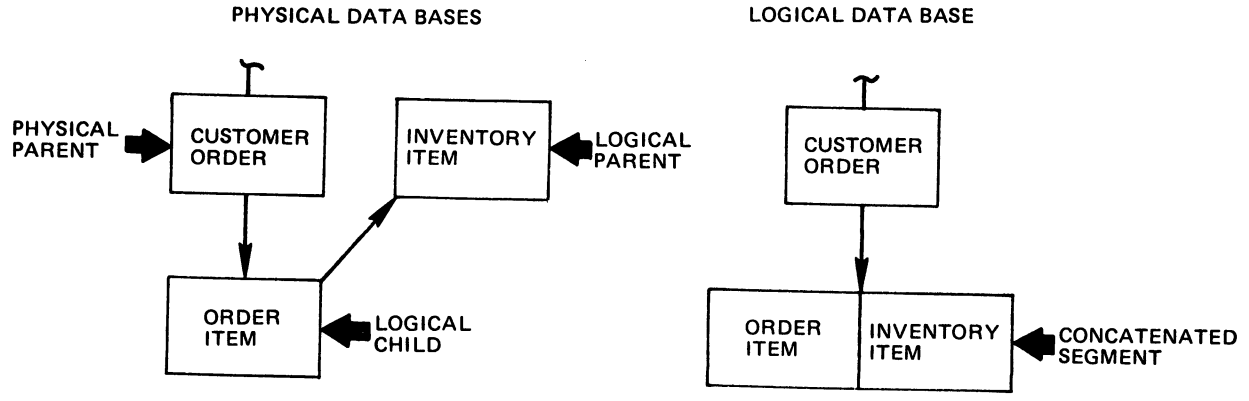


Figure 2-13. Segment Types Involved in Logical Relationships

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).

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 are supported for HD data bases only, and are implemented using direct address pointers, which are all 4-byte relative byte address pointers similar to other pointers.

The Virtual Logical Child Segment (VLC)

DL/I uses a special segment type to define the relationship between the logical parent and its logical children. 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.

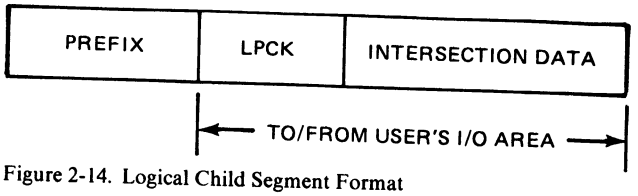
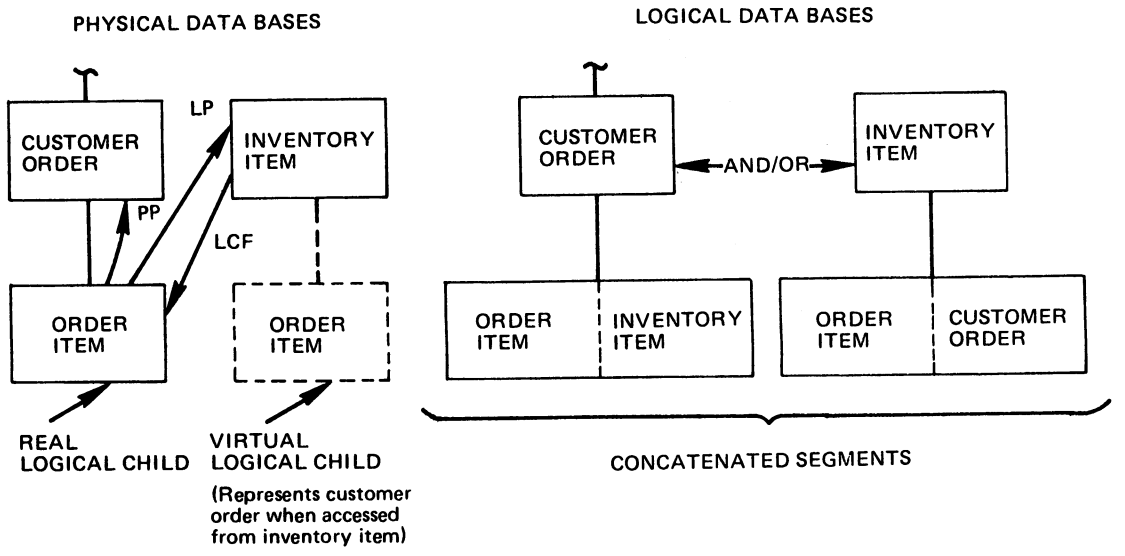


Figure 2-14. Logical Child Segment Format



KEY:

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

Figure 2-15. Virtual Paired Bidirectional Logical Relationship

When accessed, the virtual logical child contains the concatenated key of the physical parent of the real logical child, plus the intersection data of the real logical child. So the virtual logical child 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 HD 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 appear in the DBDs, as discussed later.

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.

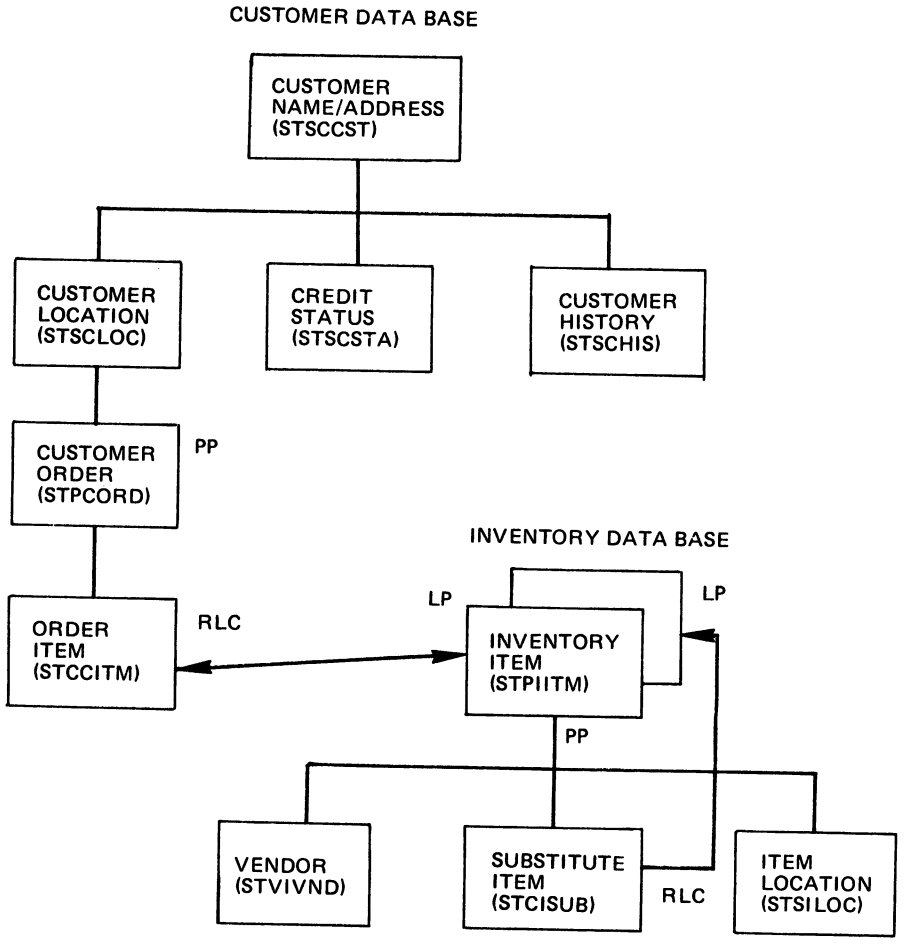


Figure 2-16. The Phase 2 Physical Data Bases

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.

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:

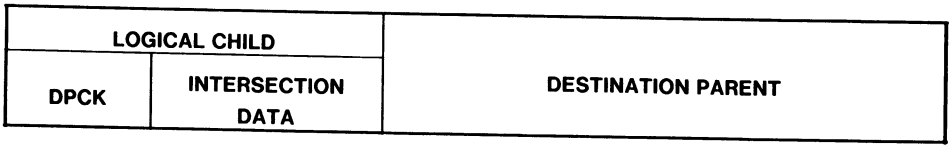


Figure 2-17. Concatenated Segment Format

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.

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.

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/I, 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 only one physical parent segment, 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.

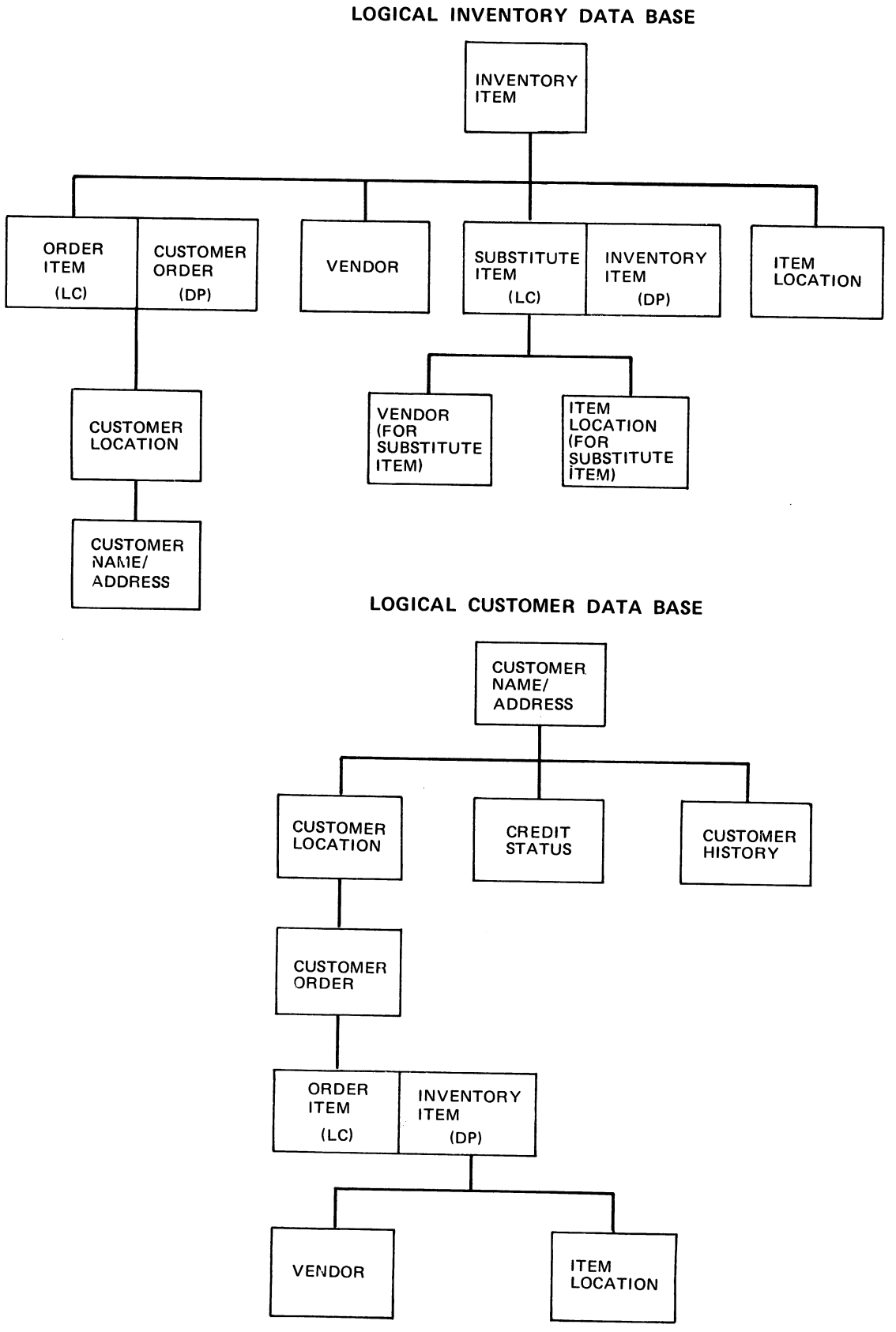


Figure 2-18. Phase 2 Logical Data Bases

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.

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 of a typical DL/I application. They merely show the different possible combinations.

Processing Logically Related Segments

The segments involved in logical relationships can be accessed through their physical data bases; insert, delete, or replace operations may be performed through either the logical or physical access path. 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 been previously 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 *DL/I DOS/VS Data Base Administration.* A general discussion follows.

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

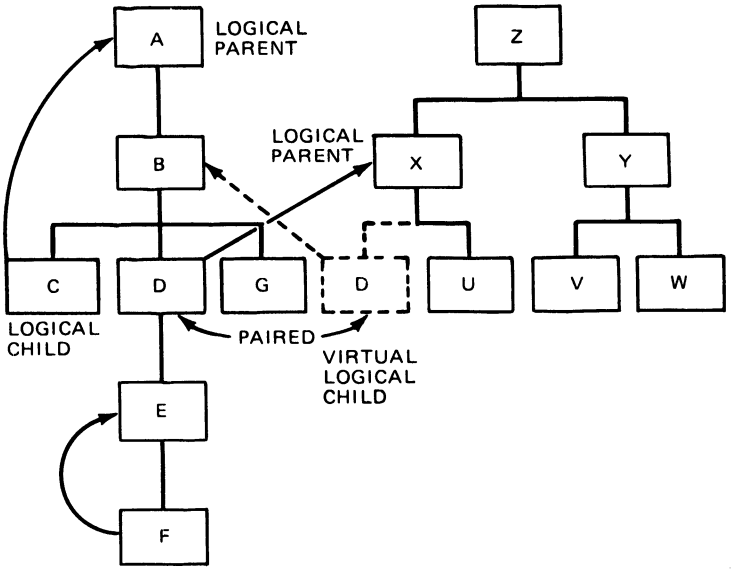


Figure 2-19. Using Multiple Logical Relationships (Part 1 of 2)

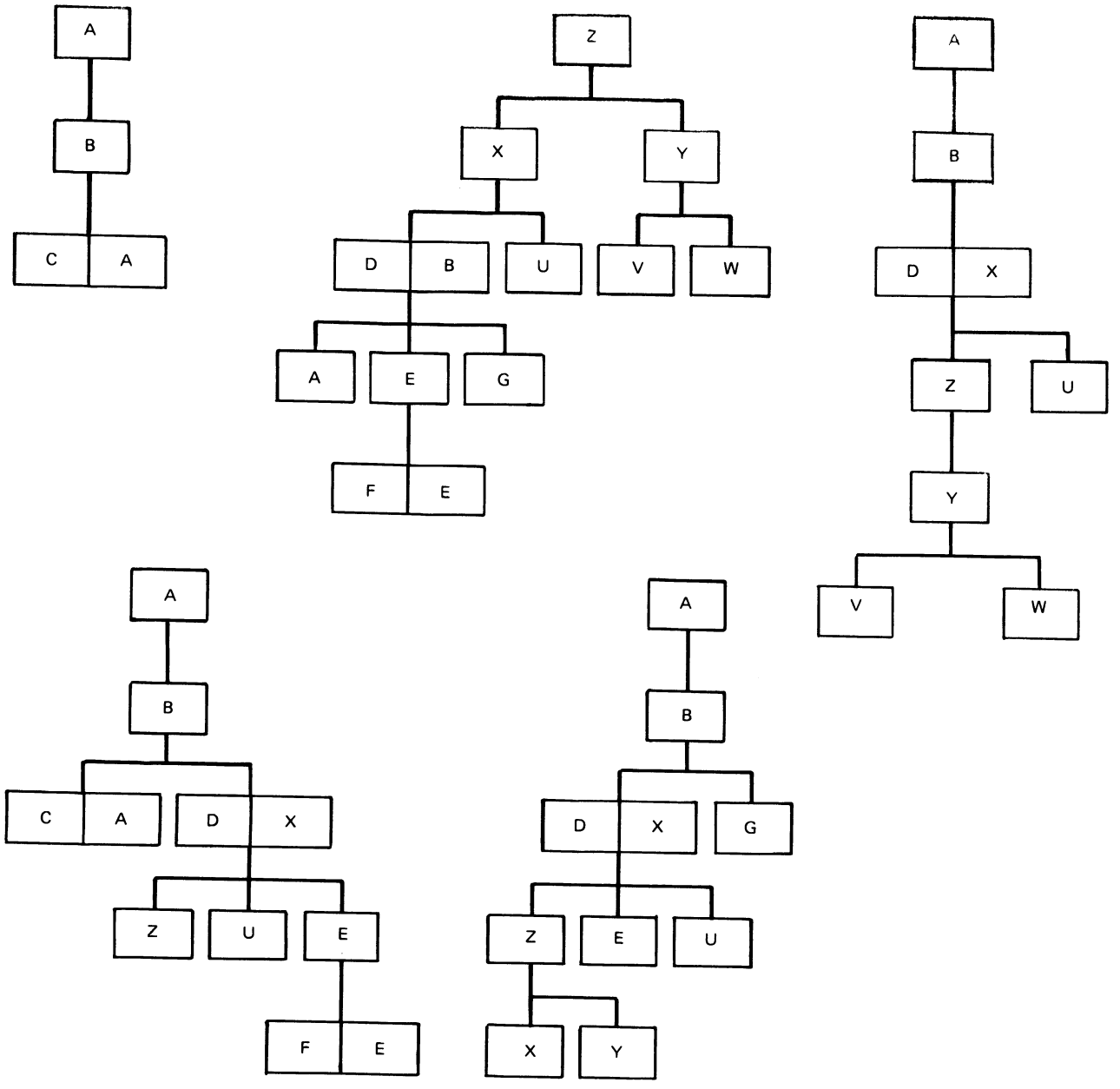


Figure 2-19. Using Multiple Logical Relationships (Part 2 of 2)

The rules for the logical relationships are coded as xxx in the DBD, 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:

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.

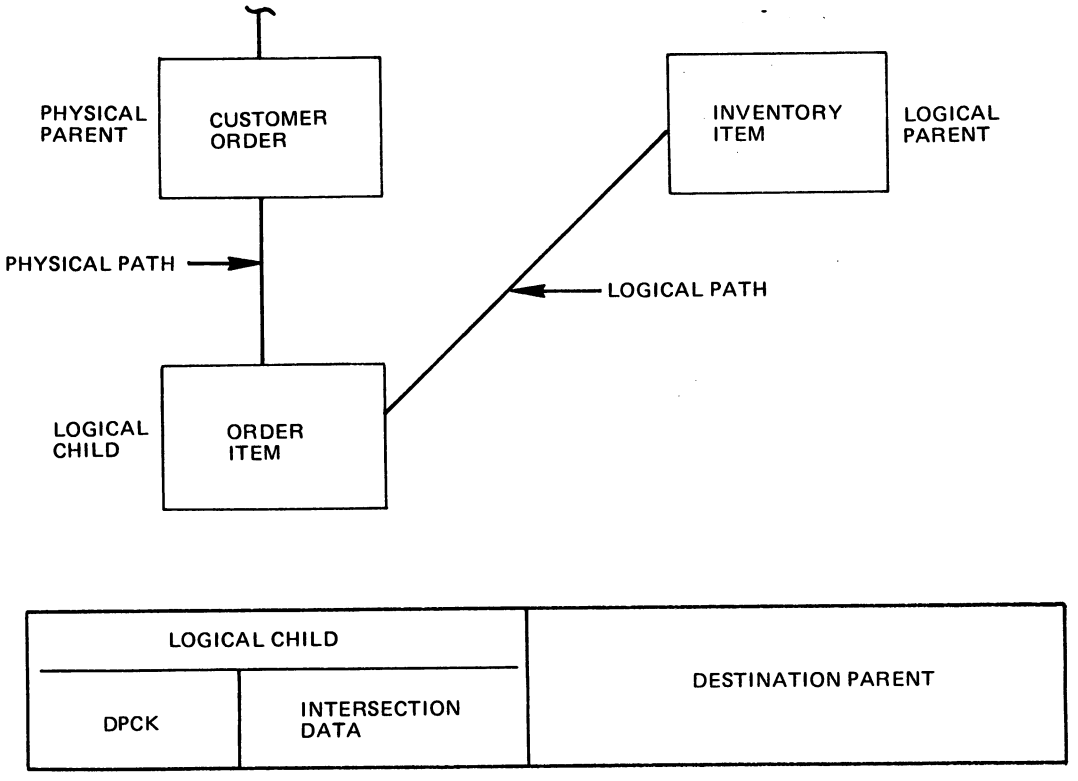


Figure 2-20. Virtual Paired Bidirectional Logical Relationship

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 command 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 command 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 caution 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 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

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. This 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. This 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. This pointer is optional.

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

DL/I secondary indexing 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 located in an auxiliary data base associated with an HD data base.

- Index Pointer Segment

  A segment that contains data and a pointer to the index target segment which 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 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 indexes. The secondary index provides an alternate processing sequence. For example, by using secondary indexes, an application program can process the Customer data base in either order number or name sequence.

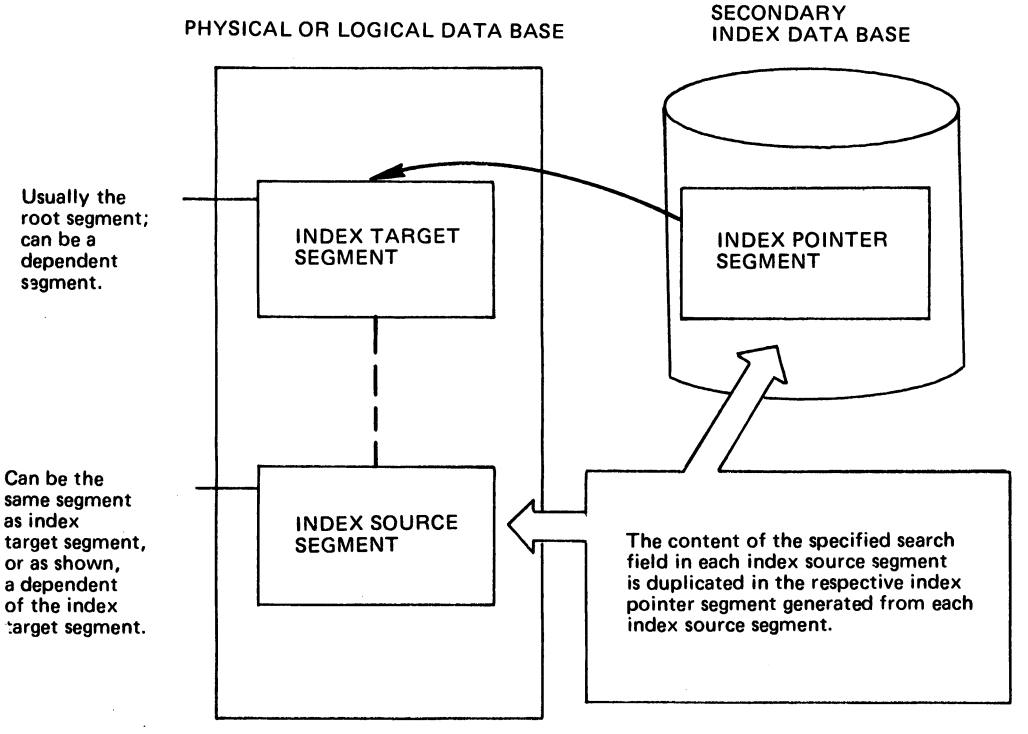


Figure 2-21. Segment Types Associated with a Secondary Index

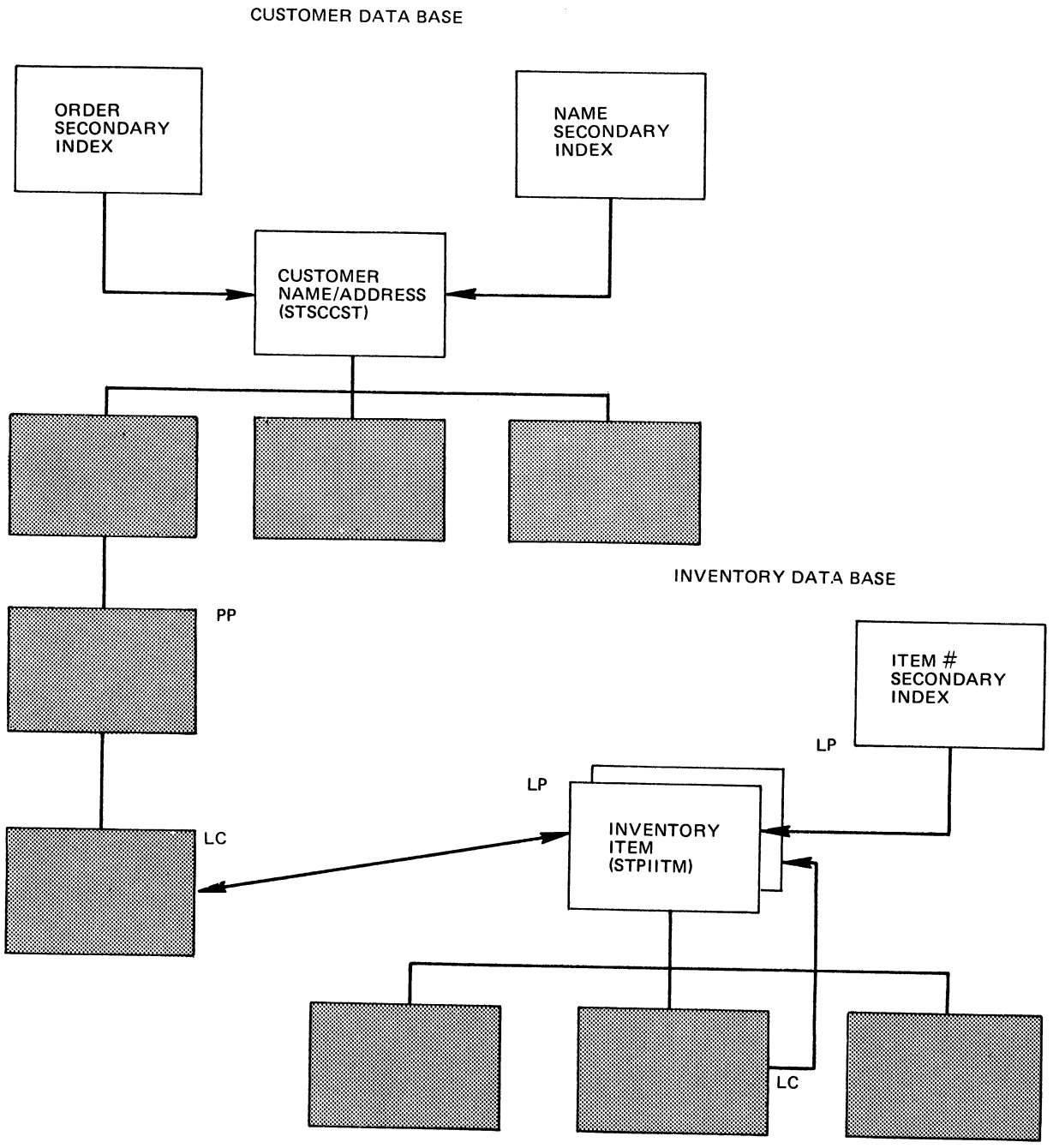


Figure 2-22. Phase 3 Physical Data Bases

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.

Creating a Secondary Index

Secondary indexes are created with the standard DL/I data base reorganization utilities (see Chapter 7). 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 DL/I of the length of data in the segment. Because the size field is in the data portion of a segment, the data length includes the size field. 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 segment 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 and must reside in an HD 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 *DL/I DOS/VS Data Base Administration.*

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 9 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 segments 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 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 *DL/I DOS/VS Data Base Administration.*

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 physical 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 does 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 gives 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 recompiling 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 does 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. For further details see "User Field Exit Routine" later in this section.

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 passed to the exit routine 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.

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 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 occurs 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 an abend of the program with a status code of Kx, where 'x' is the code set by the user routine.

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 use field level sensitivity always view these segments as fixed length. DL/I maintains the two-byte length field. 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 uses field level sensitivity, and the added field does not yet exist (contains no data), DL/I expands the segment with null values (for defined fields) or binary zeroes (for undefined areas) to fit the application program's view.

Additional Field Sensitivity Considerations

- Segment Qualification
  Any field to be used for segment selection, 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 the high level programming interface WHERE option should be in the format of the application program's view of the field. The field identified in the WHERE option, 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 WHERE option are not converted. (See Chapter 4 for WHERE option information.)

Notes:

DL/I does not take field type into consideration for compares for field values. As a consequence, for binary values, 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.

- Fields and Subfields
  You may define a field for the application program's view that contains a number of other fields as subfields. This allows a set of separately processed fields to be referenced as a group and used in segment selection. For purposes of this discussion, we will call this field an "overfield". The following considerations apply:
  - Overfields must be completely defined for the application view by the subfields they contain. These subfields must be contiguous (no holes).
  - Overfields may be defined 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/I 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 consid-

eration. Some examples of entities are: parts, projects, orders, and customers. 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 2-23 illustrates the concepts of data elements and their occurrences in recording the facts about two entities, INVENTORY ITEMS (A) and ORDER ITEMS (B).

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 identification of a particular entity occurrence. It is a special attribute of the entity. Keys are not always unique. In such cases, entities with equal key values are called *synonyms.* For instance, the full name of 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 employee number.

ENTITY A: INVENTORY ITEMS		
DATA ELEMENT	OCCURRENCES	
NAME	VALUE	VALUE
Item Number	200	300
Description	200	300
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 2-23. Concepts of Data Elements

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 interacting with the data base. (See Figure 2-24.)

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 scheduling 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 use the transaction for the data base design. A similar role is set aside for the transaction in program design by adding detailed input, processing, and output descriptions, to the data element usage.

Access Paths

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

For efficient direct access, each access path should use 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 2-25.

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

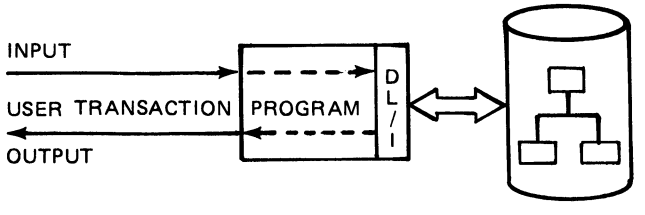


Figure 2-24. The Transaction

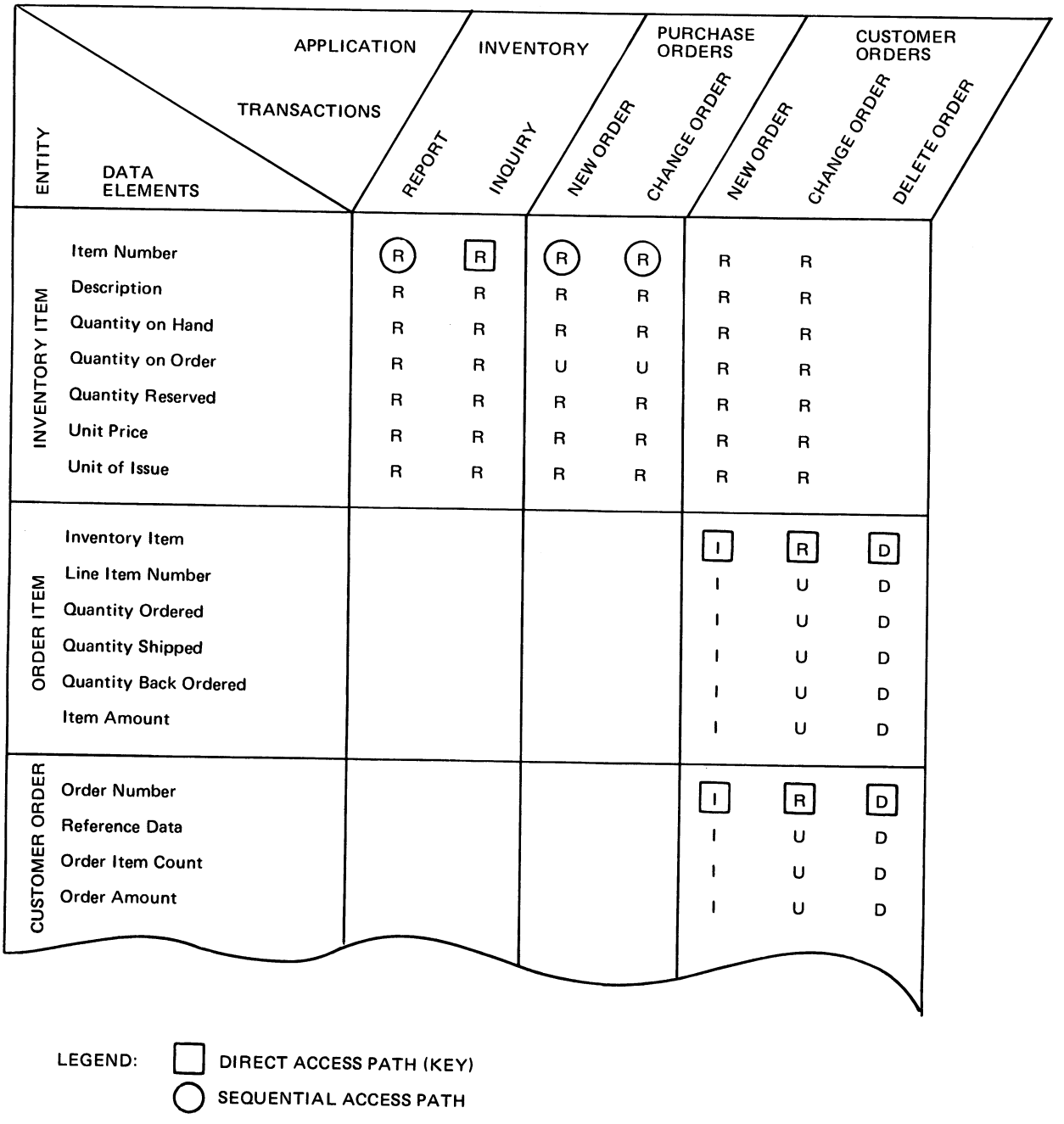


Figure 2-25. The Transaction/Data Element Matrix

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 2-26) can be generally divided into the following tasks:

- Gathering requirements
- Designing application data structures

- Designing physical data structures
- Design and performance evaluations.

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.

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 individual 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 required by the various users.

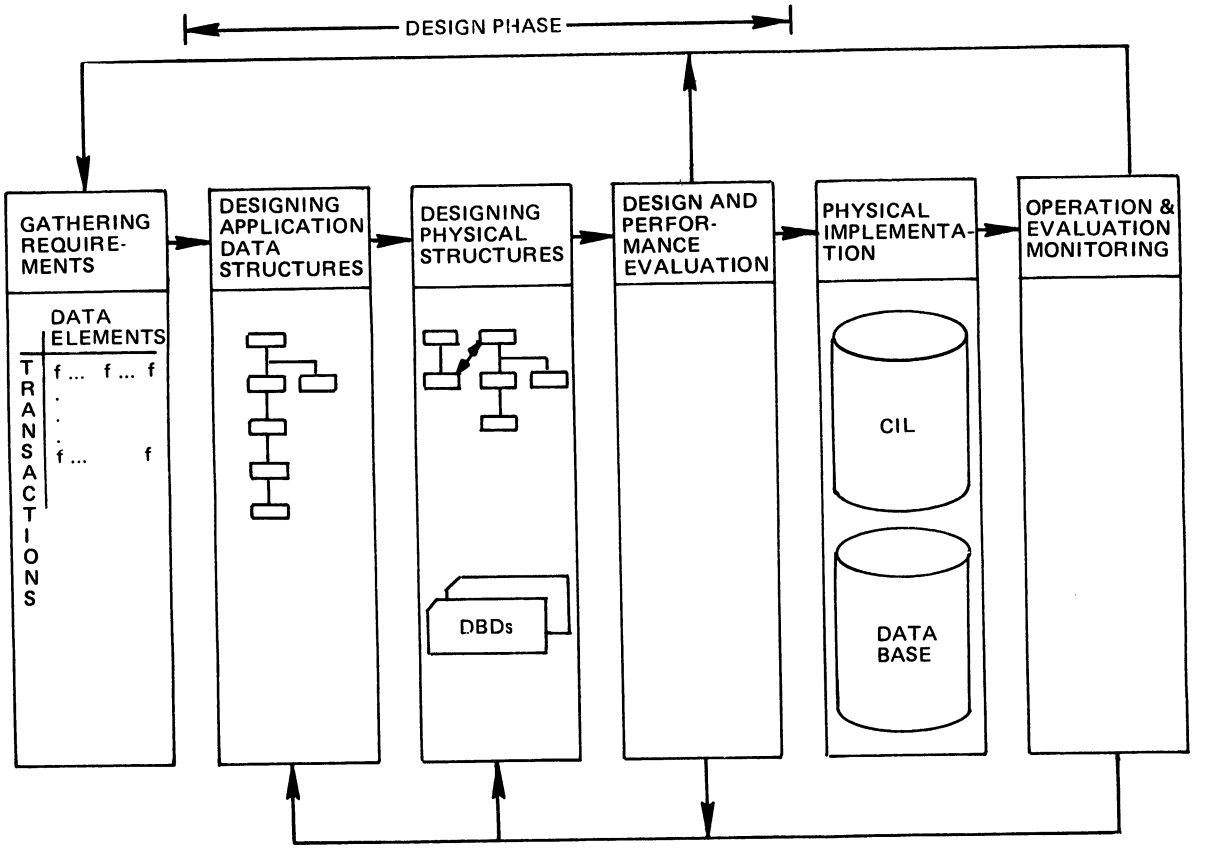


Figure 2-26. The Steps In Data Base Design

- The functions being performed against the data (the transactions).
- The access path 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, 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 future applications and data elements.

Additional columns could be added for miscellaneous information such as:

- 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 COBOL COPY or PL/I %INCLUDE facility for centralized management of segment descriptions. Ultimately, the DB/DC data dictionary system might be used.

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 use 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 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 considerations that may yield changes in the segment design are shown in Figure 2-27.

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 2-27. Grouping Data Elements Into Physical Segments

The numbers shown in Figure 2-27 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, that is, only particular applications may have access to it, it can be stored in a seperate segment 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 Your Primary Access Technique

Hierarchical Direct (HD) Access Method

Direct organization provides for the locating of any data base record in the data base directly, without searching sequentially through the records from the beginning. Direct access in DL/I is accomplished by using either a randomizing routine or an index. DL/I can find any data base record that you want, independent of the sequence of data base records in the data base. HD access can give good results

with either direct or sequential processing.

Direct access uses pointers to maintain the hierarchical relationships between segments of a data base record. By following pointers, DL/I can access a path of segments without first passing through all of the segments in the preceding paths. In direct access, pointers and addresses are maintained internally.

Some of the requirements that direct accessing satisfies are:

- Fast direct processing of roots using an index or a randomizing routine
- Good sequential processing of data base records using the index
- Fast access to a path of segments via pointers.

In addition, when you delete data from a direct access data base, the new space is made available almost immediately. This provides efficient space utilization, and means that you don't have to reorganize the data base often to take advantage of unused space.

Types of Primary Access

The primary access technique for direct organization may either be by index or through a randomizing routine. Primary access through a randomizing routine is efficient for data bases where the records are to be accessed primarily in random order. Data bases for which a significant amount of processing involves sequential access of data base records (sequence based on the sequence field of the root segment) should be defined using primary access through an index.

Note: There is no support for sequential processing of records using access through a randomizing routine. Sequential processing in data bases using a randomizing routine for primary access should be accomplished using a secondary index based on the root sequence field.

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

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

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

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

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

- The block size you use

- The number of RAPs per block
- The pattern for chaining different segments through pointers.

For sequential access of data base records by root key, use a secondary index.

Indexed Access Characteristics: With indexed access, root segments are:

- Initially loaded in key sequence
- Stored wherever space is available after initial loading
- Identified through the index, by the root key value that you supply.

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

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

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

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

If you are going to process data base records sequentially, you can specify special pointers in the DBD so that DL/I doesn't have to go to the index each time to locate the next root segment. These pointers chain the roots together. If you don't chain roots together, HD indexed always goes to the index to locate a root segment. When you process data base records sequentially, HD indexed accesses roots in key sequence in the index. This only applies to sequential processing; when you access a root segment directly, HD indexed uses the index, and not pointers, to find the root segment you've requested.

Defining VSAM Clusters

When 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 (that is, the location of the key in the KSDS record).

Use the VSAM share option 1 and perform a LISTCAT after a DEFINE command to verify that VSAM accepted the parameters.

Data Base Design Checklist

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

- Use a structure no more complex than necessary.
- Keep frequently accessed segments near the top and to the left of the hierarchy.
- Avoid widely varying segment sizes for volatile segments in the same data space.

- Check the requirement for any segment type whose relative frequency under its parent is one.
- Oversegmentation results in many DL/I commands and longer reorganization times.
- 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.
- 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

About This Chapter

This chapter is divided into three parts:

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

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 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, 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.

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 two subjects in concurrence with the three phases:

1. Basic DBDGEN for physical data bases
2. DBDGEN for logical relationships

For each data base, 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 problem program job. The DL/I macro instructions used for DBD generation exist in a 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 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.

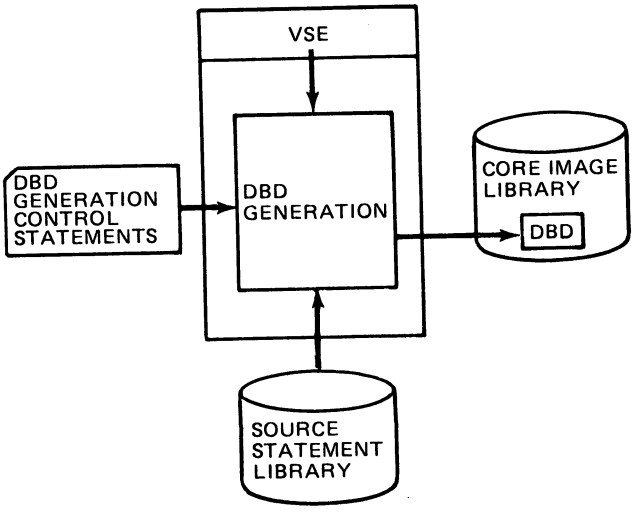


Figure 3-1. Data Base Description Generation

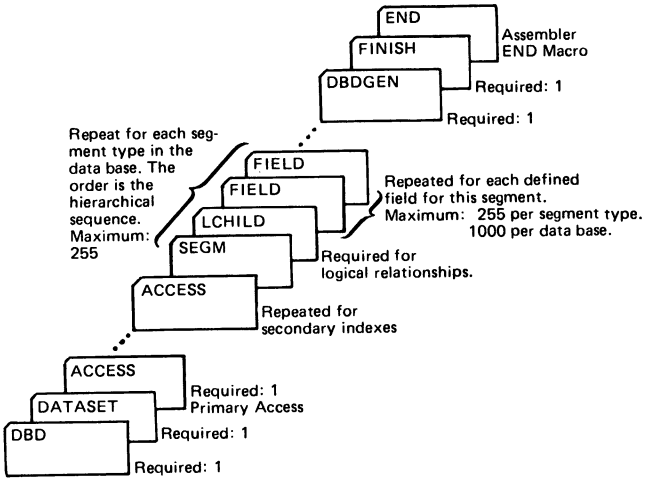


Figure 3-2. DBDGEN Input Deck Structure

DBDGEN Coding Conventions

DBDGEN statements are 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. 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.

[ ] indicates optional operands. The operand enclosed in the brackets (for example [VL]) may be present, depending on whether 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, * ACCESS=HD

> ***How to Create a DBD Interactively . . .***
>
> The DBD creation and generation procedures described in this chapter can also be done interactively on a 3270-type terminal by using IMF (Interactive Macro Facility).
>
> IMF provides the same options that are available through the conventional method of coding DBD control statements (macros). Through interactive dialogue, however, the need for a detailed knowledge of these macros is reduced.
>
> For overview information about IMF, see "Using the Interactive Facilities" in Chapter 1 of this manual.
>
> For more detailed IMF introductory and procedural information, see *DL/I DOS/VS Interactive Resource Definition and Utilities.*

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 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 *DL/I DOS/VS Resource Definition and Utilities.*

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={HD[,CIANPT={1 }][,PRIMCI={OVERLAP }][,RILIM=bytes]} {anch} {control intervals} ,IMSCOMP={NO } {YES}
		The following parameters do not apply to HD data bases and therefore are given only general consideration in this manual.
		{HSAM } {HISAM } {SHSAM } {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.

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.

HD
: specifies the direct access method.

Notes:

- Guidelines for selecting the access method for a particular data base are provided under the topic "Data Base Access Methods" in Chapter 2.
- Parameters for the 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.

CIANPT
: specifies the number of root anchor points (anch) desired in each control interval or block in the root addressable area of an HD 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 greater than 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, then DL/I uses anchor point one in the control interval or block.

PRIMCI

specifies the number of control intervals or blocks in the root-addressable area of the HD data base. This parameter must be an unsigned integer ranging from 1 through 16,777,215 or the characters 'OVERLAP'. The default value is OVERLAP. The value entered for this parameter is passed to the randomizing module for this data base each time the module is called. The value does not represent the total number of blocks that constitute an HD data base, but instead represents only that portion of the data base which is to contain root anchor points from which root segments can be chained. The remainder of the data base may be used for root segment synonyms and dependent segments. In this overflow area, space is initially allocated sequentially.

If 'OVERLAP' is specified, no upper limit check is performed on the relative block number (rbn) created by the randomizing module.

If this parameter is provided, but the user's randomizing module produces an rbn greater than the indicated value, the maximum value is used in place of the generated rbn.

Note: If one of the randomizing modules supplied with DL/I is used, this value must be used. Omitting this value or specifying OVERLAP will cause unpredictable results to occur in the randomizing module during load execution.

RILIM

specifies the maximum number of bytes of a data base record that can be stored in the root addressable area in a series of inserts unbroken by a command to another data base record. This parameter is used to control the amount of primary space used by one 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 interger whose value does not exceed 16,777,215 and is not less than 1.

If, during initial load or during normal operations, a series of uninterrupted inserts occurs where all the inserts are for segments with the same root, and the total size exceeds the specified limit, the segment being inserted will be placed in the overflow area. Any subsequent segments for the same data base record inserted during the uninterrupted sequence are also placed in the overflow area.

Note: Any interruption, such as the retrieval of a segment in another data base record, resets the count.

IMSCOMP

specifies whether the data base is to be created in a format compatible with the Information Management System/Virtual Storage (IMS/VS). YES is recommended if either eventual migration from DL/I to IMS/VS is planned or if this data base is to be alternately accessed by DL/I and IMS/VS. NO means format compatibility is not required. NO is the default value.

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} {3340} ,DEVICE={3350} {3375} {2314} {TAPE} {FBA } [,BLOCK=(blk-fct-1)] [,SCAN={cyls}] {blks} {3 } [,FRSPC=({fbff,fspf})] { 0 0 }
		The following parameters are not used for HD data bases and therefore are given only general consideration in this manual.
		[,RECORD=(rec-len-1 [,rec-len-2])] [,DD2=fname2] [,DEVADDR=(SYSnnn-1,SYSnnn-2)] [,OVFLW=fname3]

DATASET
identifies this statement as the DATASET control statement.

DD1=fname1
identifies the DLBL filename (1-7 characters) used in the job control language 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=HD; 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 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 (HD) 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) are calculated during DBD generation using a control interval size of 2048 bytes wherever possible. For HD, 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 HD 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
Primary Index HD	2 + 4T + 4CI + 8C2 + 4LP1 + 8LP2 + 4LC + 4LP3 + PP	NA	8	10	4068	4096	2048
Primary Random-izing HD		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 CIANPT 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.

FRSPC=

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

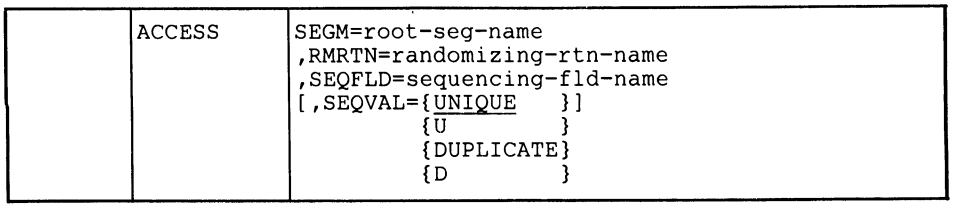
A specification of FRSPC=(5,40) results in a data base load (or reload) in which every fifth block (5, 10, 15, etc.) would be left free and at least 40 percent of all other blocks would also be left as free space. This free space would be used at insert time to place the inserted segments as close to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

ACCESS Statement

This statement specifies the parameters necessary to define the primary randomized and the primary indexed access to a data base. The format of the ACCESS statement is:

Primary Randomized Access



	ACCESS	SEGM=root-seg-name ,RMRTN=randomizing-rtn-name ,SEQFLD=sequencing-fld-name [,SEQVAL={UNIQUE }] {U } {DUPLICATE} {D }

SEGM=

identifies the name of the root segment for this HD organization data base. Segment names are one to eight characters long. The first must be alphabetic; the rest may be alphameric.

RMRTN=

identifies the phase name in the core image library of the randomizing module. The phase name must be one to eight characters long. The first character must be alphabetic; the rest can be alphameric. You can supply your own randomizing module, or you can use one provided by DL/I. (See *DL/I DOS/VS Data Base Administration* for more information on randomizing modules.)

SEQFLD=

identifies the name of the sequence field within the root segment to be used for sequencing. Field names are one to eight characters long. The first character must be alphabetic; the rest may be alphameric.

SEQVAL=

indicates whether each value of the sequence field is unique or if duplicate values occur. This operand is optional. If you omit it, the default is UNIQUE.

Primary Indexed Access

	ACCESS	SEGM=root-seg-name ,SEQFLD=sequencing-fld-name ,REF=index-dbd-name

SEGM=

identifies the name of the root segment for this HD organization data base. Segment names are one to eight characters long. The first must be alphabetic; the rest may be alphameric.

SEQFLD=

identifies the name of the sequence field within the root segment. The maximum size of this field is 236 bytes. Field names are one to eight characters long. The first must be alphabetic; the rest may be alphameric.

REF=

specifies the name you want used as a symbolic file name for the VSAM KSDS and as a DBD name for the index data base. The name must be unique; it must not duplicate any other DBD name. The name must be one to seven characters long. The first character must be alphabetic; the rest may be alphameric. Do not use the character '@'.

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 assumes the segment to be a fixed-length segment and 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 reserves 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 HD data bases only.

- TWINBWD (TB) (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 an indexed data base.

- TWIN (T) (only twin forward pointer)

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

- NOTWIN (NT) (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 *DL/I DOS/VS Data Base Administration.*

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 command, and the new occurrence is inserted before the segment that satisfied the last command.

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 core image library that is assigned.

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

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

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 segment selection. 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 used by an application program in a DL/I command. Duplicate field names must not be defined for the same segment type. The fld-name1 must be a 1- to 8-character alphameric value. It is recommended that you start the field name with an alphabetic character. If you reference this field using the SENFLD statement during PSBGEN, the field name must start with an alphabetic character. See also, the START parameter.

SEQ

the presence of the keyword SEQ as a parameter of this operand identifies this field as a sequence field in a dependent segment type (or root segment for non-HD data bases). As a general rule, a segment can have only one sequence field.

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 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.

Note: M is valid only for root segment types for HSAM or SHSAM.

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, or HD 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)*.

Note: The name of the previously defined field must start with an alphabetic character. 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). (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 commands 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 because 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 statements 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/VSE assembler.

	END	

Execution of DBDGEN (Job Control Language)

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

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

Examples of Physical DBDs

Figure 3-4 shows sample HD data bases and the DBD statements required to assemble them. They are the Phase 1 Inventory and Customer data bases of the batch sample application. The data bases are assumed to reside on a 3340. If the device is other than a 3340, the DATASET statements should be changed.

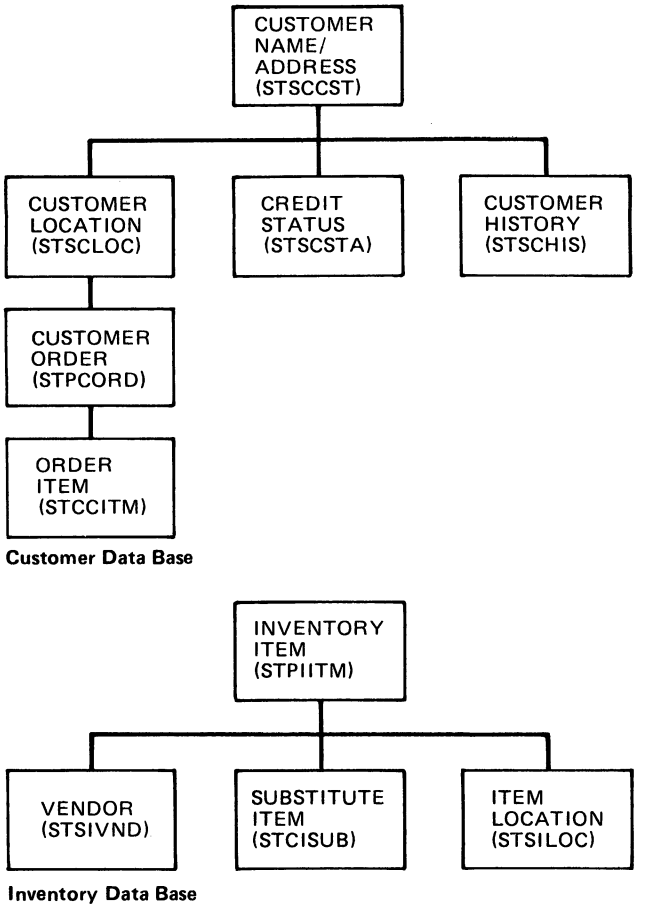


Figure 3-4. DBDGEN for the Phase 1 Data Bases (Part 1 of 4)

```
// 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=HD,                HIERARCHICAL DIRECT ACCESS   X
              CIANPT=3,                 ROOT ANCHOR POINTS PER BLOCK X
              PRIMCI=100,               ROOT ADDR. AREA HI RELATIVE BLK X
              RILIM=600                 INSERT BYTES LIMIT FOR RBA
        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
* THE FOLLOWING ACCESS MACRO IS FOR THE PRIMARY ACCESS POINT
        ACCESS                                                       X
              SEGM=STSCCST,             ROOT SEGMENT                 X
              SEQFLD=STQCCNO,           SEQUENCE FIELD               X
              SEQVAL=U                  UNIQUE VALUE                 X
              RMRTN=DLZHDC10            RANDOMIZING ROUTINE
        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,             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. LOCATION 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 PRECEDING 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=HD,               HIERARCHICAL DIRECT ACCESS  X
               CIANPT=3,                ROOT ANCHOR POINTS PER BLOCK X
               PRIMCI=100,              ROOT ADDR. AREA HI RELATIVE BLK X
               RILIM=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
         ACCESS                                                     X
               SEGM=STPIITM,            ROOT SEGMENT                X
               SEQFLD=STQIINO,          SEQUENCE FIELD              X
               SEQVAL=U,                UNIQUE VALUE                X
               RMRTN=DLZHDC30           RANDOMIZING ROUTINE
         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,            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)

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
ACCESS
FIELD
DBDGEN
FINISH
END

The following statement is extended:

SEGM

The following statement is added:

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= ((seg-name2,{SNGL}) {DBLE} ,(lpseg-name[,V,db-name1])) ,BYTES=bytes ,POINTER=({TWIN }[,[{LTWIN }]]) {TWINBWD} [{LTWINBWD}] {NOTWIN } {P}{P}{P} {,FIRST} ,RULES=([{L}{L}{L}][{,LAST }]) {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

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.

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=
: specifies the length of the data portion of the segment n 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 assumes the segment to be a fixed-length segment and calculates the size of the segment based on the location and length of the fields identified as belonging to it.

  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)
: this parameter is the default for a physical segment or a logical parent segment.

TWINBWD (TB)
: It is recommended that you specify TB. The use of this pointer can improve deletion performance if the segment being deleted is neither the first nor the last occurrence.

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

LTWIN (LT)
: 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 3-5, 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 3-6 and 3-7.

IF AN INSERT COMMAND 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 CONCATENATED SEGMENT IGNORED	X		X			
	DESTINATION PARENT SEGMENT IS INSERTED (*REPLACED)				X*	X	X
	COMMAND 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 3-5. Insertion Rules for Logical Relationships

		DELETE RULE SPECIFIED IN LP						
		P (BI-DIR)		L		V		
IF . . .	A DELETE COMMAND IS ISSUED FOR AN LP	X	X	X	X	X	X	
AND	ALL LOGICAL CHILDREN HAVE BEEN PHYSICALLY DELETED		X					X
	ALL LOGICAL CHILDREN HAVE BEEN LOGICALLY DELETED							X
	LP DEPENDENTS NOT INVOLVED IN AN LR							X
	LC DELETE RULE IS 'V'				X	X		
THEN DL/I	MARKS LOGICAL PARENT AS PHYSICALLY DELETED AND LOGICAL CHILDREN AS LOGICALLY DELETED		X	X	X	X	X	X
	MARKS ALL ACCESSIBLE LOGICAL CHILDREN AS PHYSICALLY DELETED				X	X		
	MARKS LOGICAL PARENT AS LOGICALLY DELETED					X		X
	REJECTS COMMAND WITH A DX STATUS CODE	X						
NOTE: Logical parent delete rules do not apply to destination parent unless it is also an LP.								

Figure 3-6. Logical Parent Deletion Rules

		DELETE RULE SPECIFIED IN LC					
		P		L		V	
IF . . .	A DELETE COMMAND IS ISSUED FOR AN LC	X	X	X	*	X**	
AND	LOGICAL CHILD HAS BEEN LOGICALLY DELETED	X					
	RELATIONSHIP IS BIDIRECTIONAL	X	X	X			X
THEN DL/I	MARKS LOGICAL CHILD AS PHYSICALLY DELETED	X		X		X	X
	MARKS LOGICAL CHILD AS LOGICALLY DELETED				X	X	X
	REJECTS COMMAND WITH A DX STATUS CODE		X				
NOTE: Logical child delete rules establish criteria for removal of LC. V rule is required for LC in uni-directional LR.							

* delete command issued for virtual logical child

** applies to delete command for either real logical child or virtual logical child

Figure 3-7. Logical Child Deletion Rules

Replacement Rules: The replacement rules determine what actions take place when a concatenated segment is presented in a REPLACE command and one or both portions of it are to be altered. Replacement rules, as shown in Figure 3-8, 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.

FIRST (F)
: 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 (L)
: 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 (H)
: assumes the user has determined positioning by a previous DL/I command, and the new occurrence is inserted before the segment that satisfied the last command. 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.

IF A REPLACE COMMAND 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 DL/I	REPLACES THE LC SEGMENT	X		X		X	
	REPLACES THE LP (DP) SEGMENT				*		X
	REJECTS COMMAND 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, command rejected with 'DA' status code

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

Figure 3-8. Replacement Rules for Logical Relationships

Virtual Logical Child: 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, D (or DATA) is for upward source compatibility to IMS/VS and db-name2 is the DBD name of the data base which contains that logical child.

POINTER=
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}
                    ,POINTER={SNGL}
                              {DBLE}
                    ,PAIR=virtchild
                           {FIRST}
                    ,RULES={LAST }
                           {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 the HLPI LAST option 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 }
```

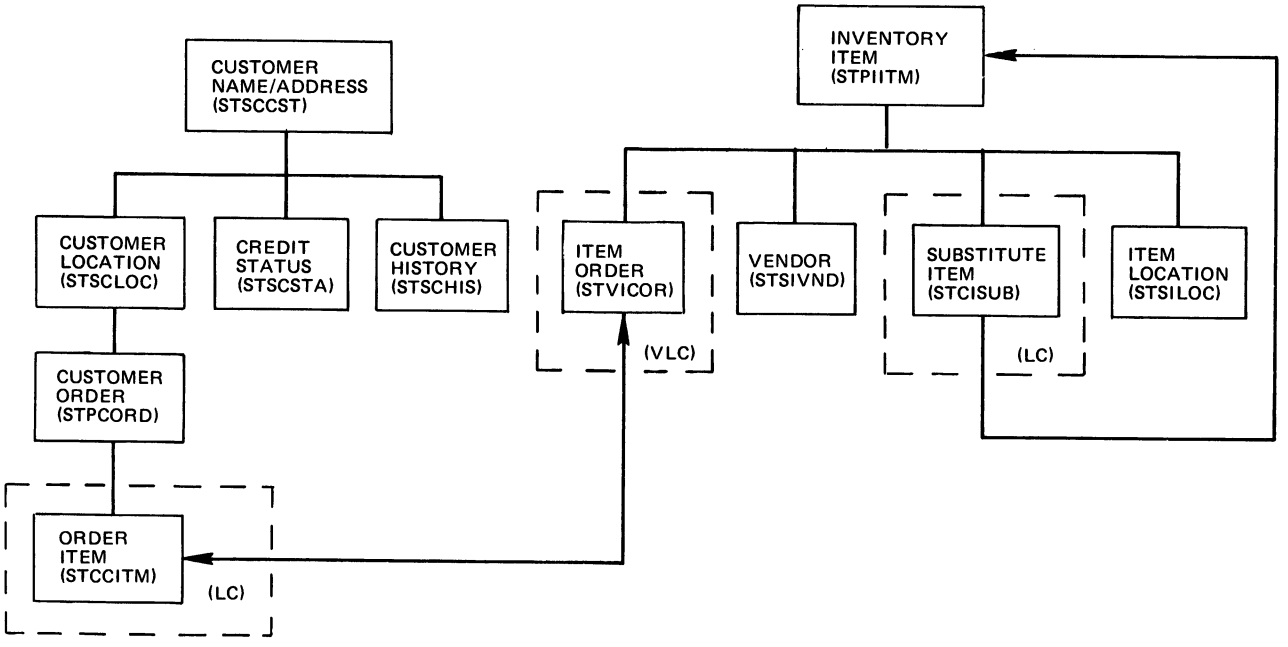
See the preceding discussion of this parameter for an explanation.

Examples of Physical DBDs With Logical Relationships

Figure 3-9 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.

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.



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 ensure that no more orders are pointing to this item before allowing deletion. The only allowable path for deletion is 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 3-9. Phase 2 Physical DBDs (Part 1 of 3)

```
        ACCESS                                                          X
              SEGM=STPIITM,           ROOT SEGMENT                      X
              SEQFLD=STQIINO,         SEQUENCE FIELD                    X
              SEQVAL=U,               UNIQUE VALUE                      X
              RMRTN=DLZHDC30          RANDOMIZING ROUTINE
        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, ITEM 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=STQIINO,BYTES=6,START=1,TYPE=C
        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 PHASE 1 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 3-9. 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 is 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 is replaced if altered.

Figure 3-9. 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 formats 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
|        |      |          {0        }
|        |      | [,PARENT={seg-name2}]
|        |      | ,SOURCE=((seg-name3,D,db-name1)
|        |      |   [,(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.

D (or DATA, the default) is for upward source compatibility to IMS/VS.

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.
- D (or DATA, the default) is for upward source compatibility to IMS/VS.
- db-name2 defines the physical DBD name of the destination parent.

DBDGEN, FINISH, and END Statements

These should be coded as before.

No LCHILD, FIELD, or ACCESS statements are allowed in a logical DBD.

Example of Logical DBDs

Figure 3-10 shows the logical DBD for the phase 2 Customer data base.

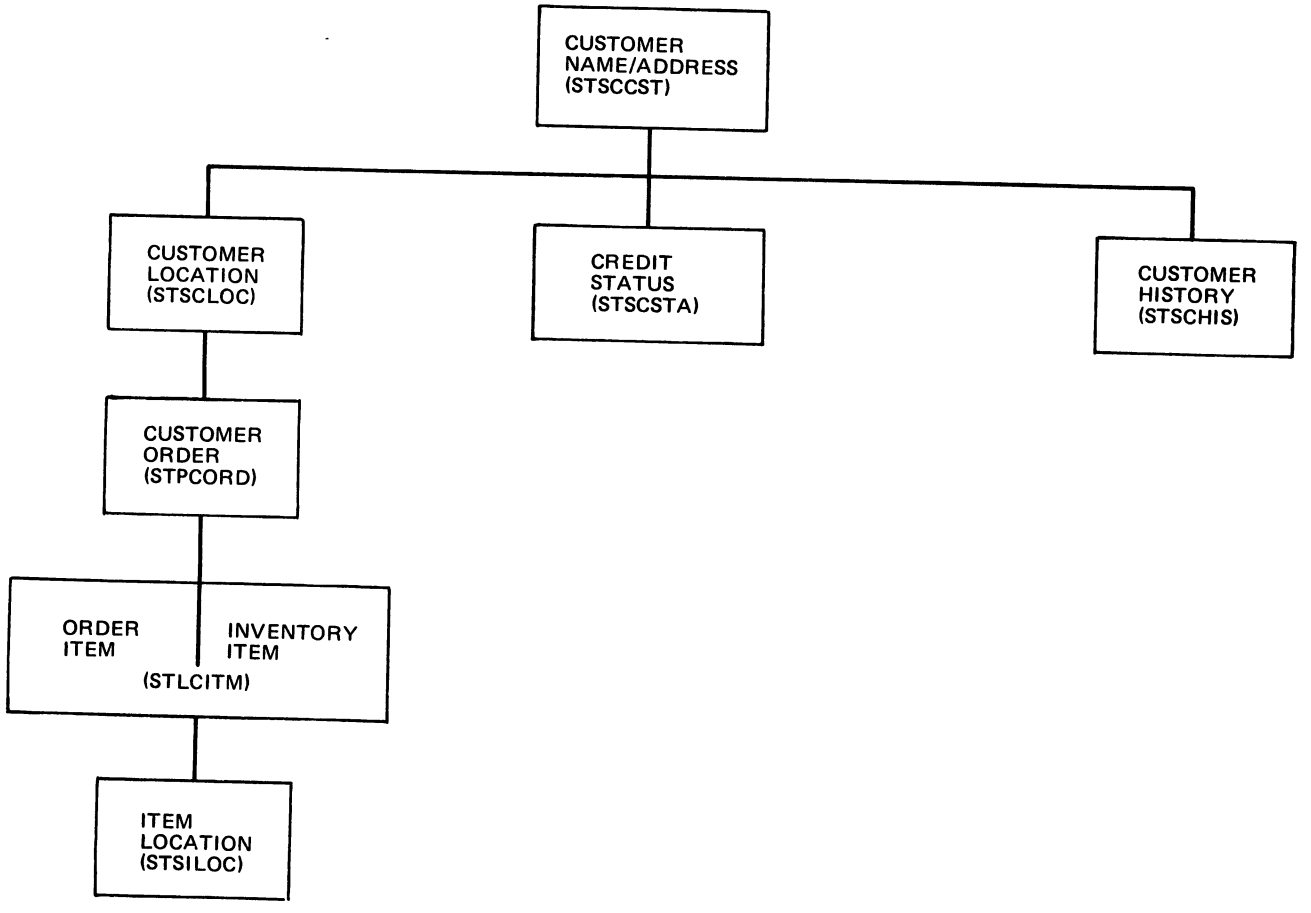


Figure 3-10. Phase 2 Logical DBD for the Customer Data Base (Part 1 of 2)

```
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 3-10. Phase 2 Logical DBD for the Customer Data Base (Part 2 of 2)

Figure 3-11 shows the logical DBD for the phase 2 Inventory data base.

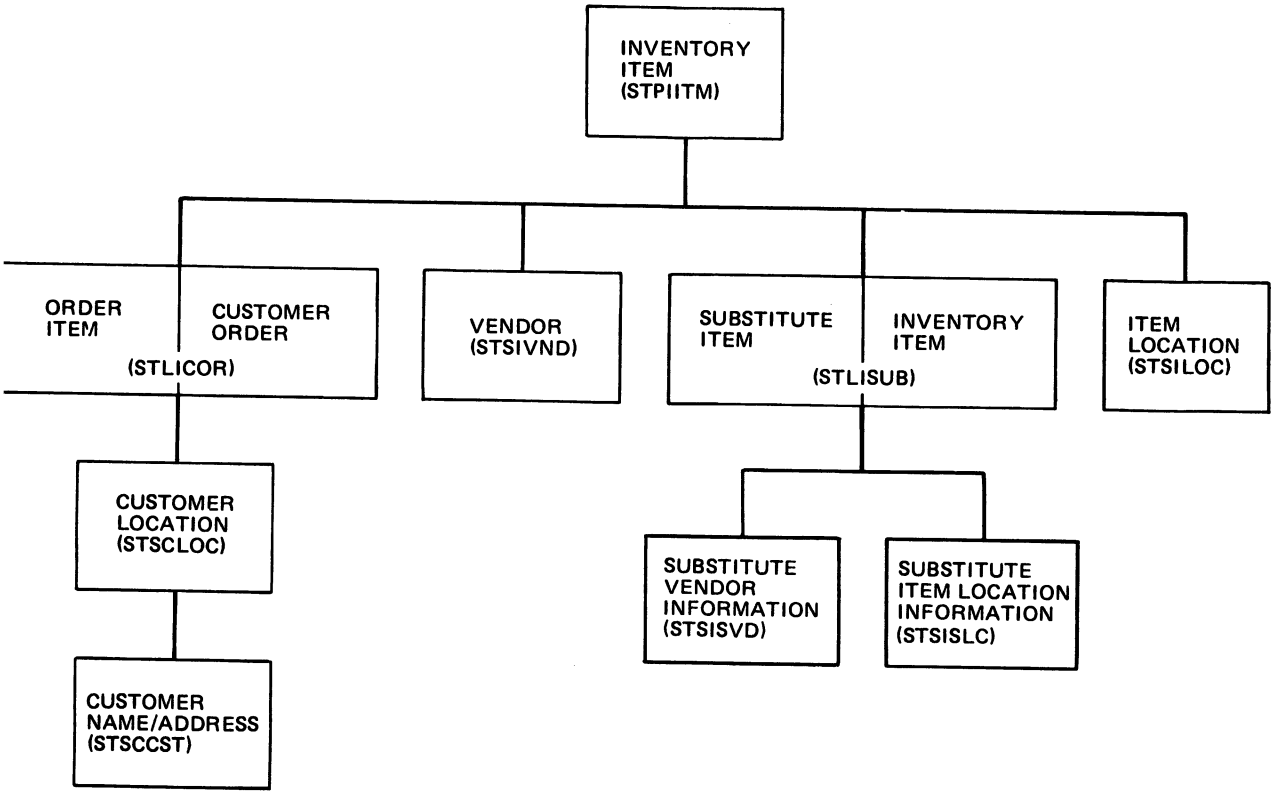


Figure 3-11. Phase 2 Logical DBD for the Inventory Data Base (Part 1 of 2)

```
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 3-11. Phase 2 Logical DBD for the Inventory Data Base (Part 2 of 2)

DBDGEN for Secondary Indexes

To support the secondary index function, the DBDGEN process is extended.

The control statements extended for the secondary index function is ACCESS.

The following control statements are unchanged:

DBD
DATASET
SEGM
FIELD
LCHILD
DBDGEN
FINISH
END

Secondary Index Access Statement

This statement specifies the parameters necessary to define the secondary indexed access to a data base. The format of the ACCESS statement is:

```
          ACCESS    SEGM=target-seg-name
                    ,SEQFLD={fldname                                  }
                            {(fldname1,fldname2[,...,fldname5])}
                    ,REF=index-dbd-name
                    [,SEQVAL={UNIQUE    }]
                             {U         }
                             {DUPLICATE}
                             {D         }
                    [,SEQSEG=source-seg-name]
                    [,SUPVAL=value]
                    [,SUPRTN=rtn-name]
```

SEGM=

specifies the name of the segment that will be the entry point (that is, the target segment) to the HD data base. The segment name must be one to eight characters long. The first character must be alphabetic; the rest can be alphameric.

Note that a target segment can be any segment in the HD data base with two exceptions; it cannot be a logical child segment or the physical dependent of a logical child segment.

SEQFLD=

identifies the field(s) that will be used in sequencing. You must specify at least one field, but as many as five may be named to form a sequencing key. Each field name must be one to eight characters long. It is recommended that each name start with an alphabetic character.

If multiple fields are named, they are concatenated in the order of their specification. The combined length of the specified fields must not exceed 236 bytes.

All fields named in this operand must be defined in the same segment. That segment must be either the target segment named in the SEGM operand, or optionally, in the source segment named in the SEQSEG operand.

REF=

specifies the name you want used as a symbolic filename for the VSAM KSDS and as a DBD name for the index data base. It will also define the field name to be used in the WHERE option to reference the key field during data base references via this access path.

The name you specify for this operand must be unique. It must not duplicate any other DBD name or any field name within the target segment. The name must be one to seven characters long. The first character must be alphabetic; the rest may be alphameric. Do not use the character '@'.

SEQVAL=

indicates whether each value of the sequence field is unique or if duplicate values occur. This operand is optional. The default is UNIQUE.

If duplicate values exist, the order in which the duplicate indexes occur is effectively random (based on the RBA of the source segment).

SEQSEG=

identifies the segment in which the sequence field(s) reside, if other than the target segment. This operand is optional.

To be valid, the sequence segment must have fixed length. Its name must be one to eight characters long. The first character must be alphabetic; the rest can be alphameric. In addition, the sequence segment must be a physical dependent of the target segment. It cannot be a logical child segment, but it may be the physical dependent of a logical child.

SUPVAL=

enables suppressing the creation of index pointer segments when the index source segment data used in the search field of an index pointer segment contains the specified value. This operand is optional.

The value must be a one-byte self-defining term (X'10', C'Z', 5 or B'00101101') or the words BLANK or ZERO. ZERO is equivalent to X'00' or 0, but not C'0'. BLANK is equivalent to C'b' or X'40'. If a packed decimal value is required, it must be specified as a hexadecimal term with a valid number digit and zone or sign digit (X'3F' for a packed positive 3 or X'9D' for negative 9).

No indexing is performed when each field of the index source segment specified in the SRCH= operand has the value of this operand in every byte. For example, if the SUPVAL=C'9' were specified, then the associated index would have no entries indexed on the value C'9999...9'.

There is a slight difference in the case of packed fields. For packed fields, each field which comprises the search field is considered to be a separate packed value. For example, if the SUPVAL=X'9F' were specified in a case where the search field comprised of three two-byte packed source fields, there would be no index entries with the search field value of X'999F999F999F' because only these would be suppressed.

Also, with the same SUPVAL=X'9F', if the search field were one six-byte field, then no indexing would be performed whenever the value of the search field was X'99999999999F'.

The only form of the sign that will be checked is the form specified. For example, if X'9C' is specified, X'9F' will not cause suppression.

SUPRTN=

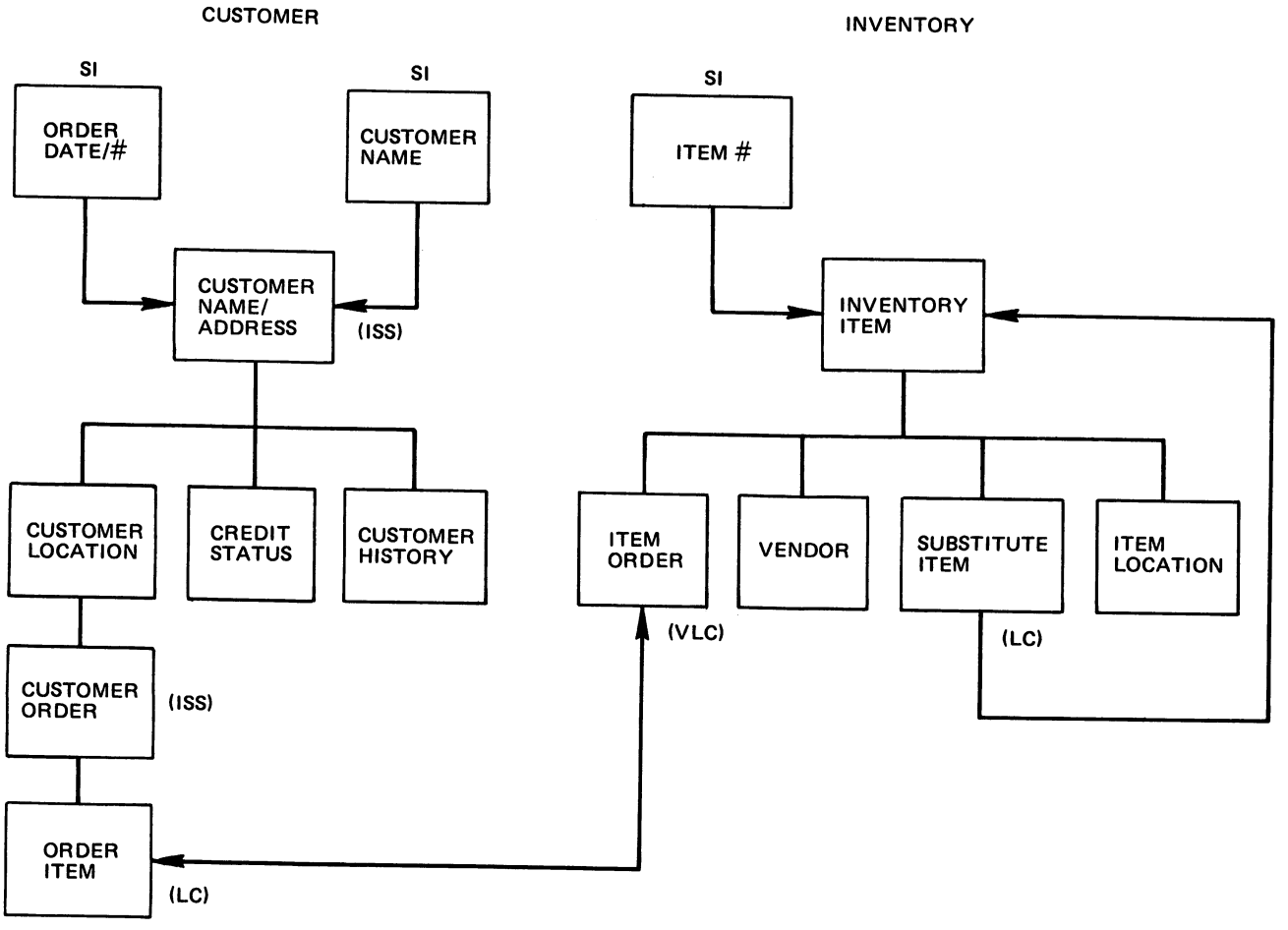
it may be used to suppress creation of indexes based on a dynamic decision made in a user-supplied routine. This operand is optional.

The parameter specified for this operand is the phase name of a user-supplied module, which receives control whenever DL/I attempts to insert, delete, or replace an index entry because of changes occurring in the source segment for the sequence field. This exit routine can inspect the source segment and decide if an index entry should be created.

Note: For more information on user-supplied exit routines, see *DL/I DOS/VS Data Base Administration.*

Physical DBD - Inventory Data Base - Phase 3

Figure 3-12 shows the physical DBDs and the associated secondary index DBDs for the phase 3 sample environment.




```
* 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=HD,CIANPT=3,PRIMCI=100,RILIM=400
         DATASET DD1=STDIDBC,DEVICE=3340,BLOCK=(2048),SCAN=2
         ACCESS  SEGM=STPIITM,SEQFLD=STQIINO,SEQVAL=U,RMRTN=DLZHDC30
* THE FOLLOWING ACCESS MACRO IS FOR A SECONDARY ACCESS POINT
         ACCESS
               SEGM=STPIITM,           TARGET AND SOURCE SEGMENT      X
               SEQFLD=STQIINO,         SEQUENCE FIELD                 X
               SEQVAL=U,               UNIQUE VALUE                   X
               REF=STXININ             REFERENCE NAME
         SEGM    NAME=STPIITM,PARENT=0,BYTES=56,POINTER=TWIN,RULES=(PPV)
         FIELD NAME=STQIINO,BYTES=6,START=1,TYPE=C
```

Figure 3-12. Phase 3 Physical DBDs (Part 1 of 3)

```
* 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
         SEGM   NAME=STVICOR,PARENT=STPIITM,POINTER=PAIRED,            X
               SOURCE=((STCCITM,D,STDCDBP))
         SEGM   NAME=STSIVND,PARENT=STPIITM,BYTES=110,                 X
               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=HD,CIANPT=3,PRIMCI=100,RILIM=400
         DATASET DD1=STDCDBD,DEVICE=3340,BLOCK=(2048),SCAN=2
         ACCESS SEGM=STSCCST,SEQFLD=STQCCNO,SEQVAL=U,RMRTN=DLZHDC10
*   THE FOLLOWING ACCESS MACROS ARE FOR SECONDARY ACCESS POINTS
         ACCESS                                                        X
               SEGM=STSCCST,          TARGET AND SOURCE SEGMENT        X
               SEQFLD=STUCCNM,        SEQUENCE FIELD                   X
               SEQVAL=D,              DUPLICATE VALUES POSSIBLE        X
               REF=STXCMNA            REFERENCE NAME
         ACCESS                                                        X
               SEGM=STSCCST,          TARGET SEGMENT                   X
               SEQSEG=STPCORD,        SOURCE SEGMENT                   X
               SEQFLD=STQCODN,        SEQUENCE FIELD                   X
               SEQVAL=U,              UNIQUE VALUE                     X
               REF=STXCRDN            REFERENCE NAME
         SEGM NAME=STSCCST,PARENT=0,BYTES=106,POINTER=TWIN
         FIELD NAME=STQCCNO,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
         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
```

Figure 3-12. Phase 3 Physical DBDs (Part 2 of 3)

```
* 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 PRECEDING 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
```

Figure 3-12. Phase 3 Physical DBDs (Part 3 of 3)

Program Specification Block Generation (PSBGEN)

For each program that 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 assembly job. The DL/I macro instructions used for PSB generation exist in a source statement library. The result of the PSB generation is the creation of a PSB CSECT. The generated PSB is link edited into a core image library (see Figure 3-13). 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 3-14 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.

Remote Program Specification Blocks (PSBs)

Using CICS/VS intersystem communication support, DL/I application programs can access data bases that are resident in another CPU, or in another CICS/VS partition of the same CPU, referred to as *remote* data bases. The system on which the application program is executing is referred to as the *local* system. The system programmer, when generating the DL/I nucleus for the local system, specifies the location of the remote data base(s) by giving the name of the remote PSB in a DLZACT TYPE=RPSB statement. A remote PSB, and its related DMBs and data bases, must all reside on the same system.

For more information about remote PSBs, see *DL/I DOS/VS Data Base Administration,* SH24-5011. For more information on DLZACT TYPE=RPSB, see *DL/I DOS/VS Resource Definition and Utilities,* SH24-5021.

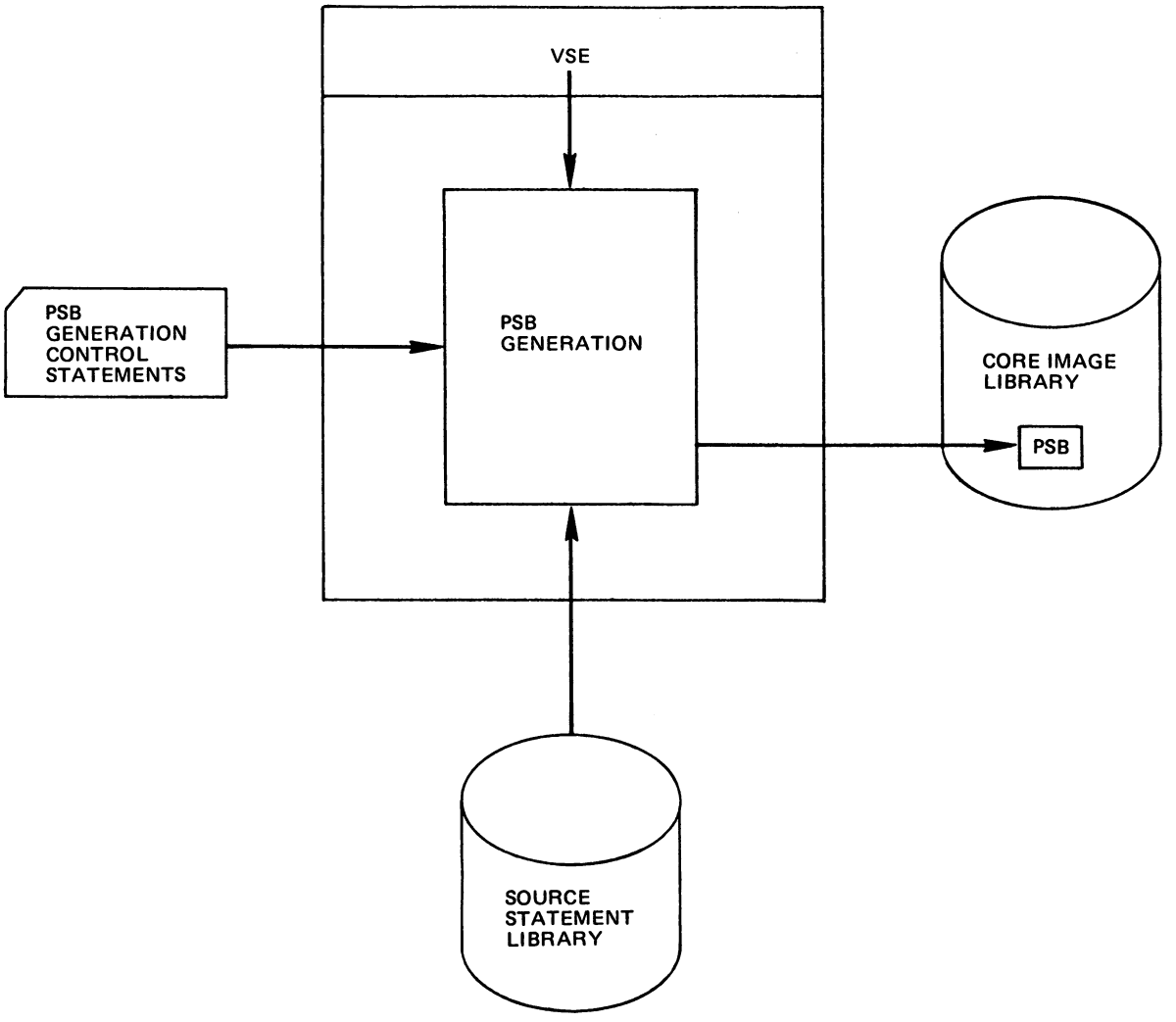


Figure 3-13. Program Specification Block Generation (PSBGEN)

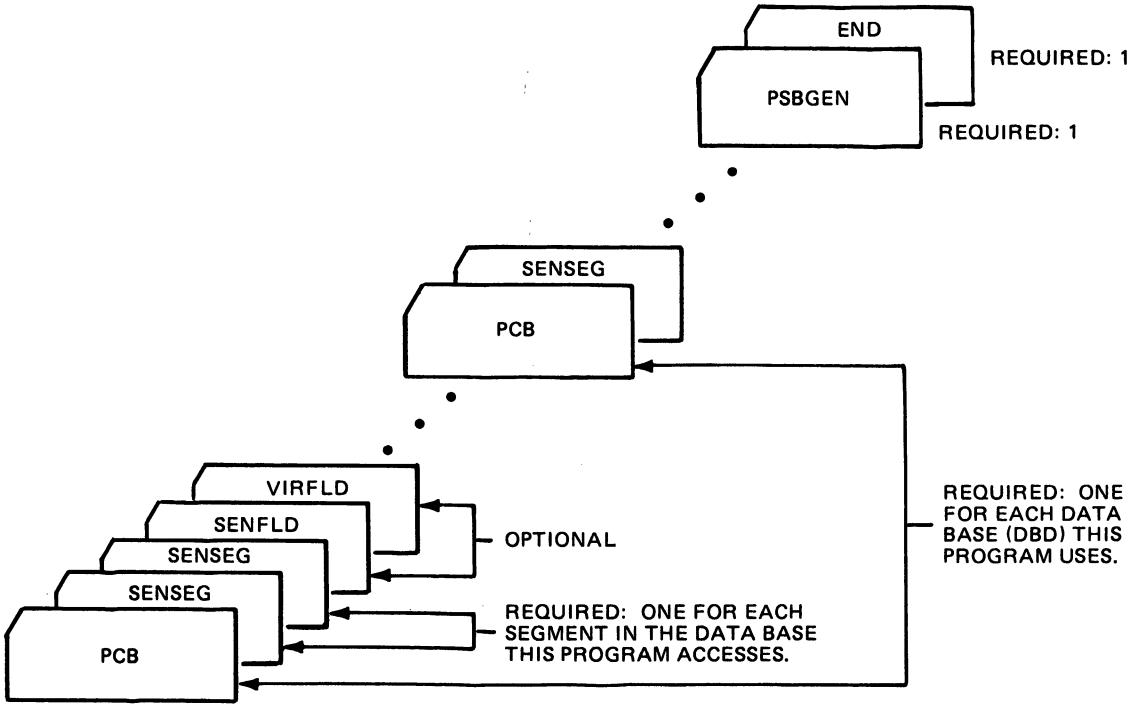


Figure 3-14. PSBGEN Input Deck Structure

How to Create a PSB Interactively . . .

The PSB creation and generation procedures described in this chapter can also be done interactively on a 3270-type terminal by using IMF (Interactive Macro Facility).

IMF provides the same options that are available through the conventional method of coding PSB control statements (macros). Through its interactive dialogue, however, the need for a detailed knowledge of these macros is reduced.

For more detailed IMF introductory and procedural information, see *DL/I DOS/VS Interactive Resource Definition and Utilities.*

Basic PSB Coding

Following are the basic PSB control statement formats.

PCB Statement

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

```
          PCB       TYPE=DB
                    ,DBDNAME=dbdname
                             {{[G][I][R][D]}      }
                    ,PROCOPT={{A           }[P][E]}
                             { L[S]               }
                             {{GO}[P]}
                    ,KEYLEN=value
                          {MULTIPLE}
                    [,POS={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 a path call is to be used on get-type commands or insert commands. Determines the maximum length of the I/O area. To be used 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.

L- Load function for data base loading (except HD with primary index)

LS- Segments loaded in ascending sequence only (HD with primary index, HSAM). This load option is required for HD with index.

O- Inhibits locking (enqueuing) by program isolation during retrievals of the same segment types in a data base. O can be used only 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=

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 is used in attempting to satisfy all subsequent GET NEXT commands. If MULTIPLE or M is specified, DL/I maintains a unique position in each hierarchical path in the data base. For a description of positioning, see *DL/I DOS/VS Application Programming: High Level Programming Interface.*

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}] {0} [,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.

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 [,BYTES=n] [,START=pos] [,TYPE=t] [,RTNAME=prog] [,REPLACE= YES] NO

NAME=

is the name of the related field defined in the physical DBD. This field name must start with an alphabetic character.

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 name of the defined field must start with an alphabetic character.

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 (in most cases) converts 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 15 characters, so the values contained in fields to be converted must be within the range of ±999999999999999.
- 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 the 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 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 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 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' 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 program. 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 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 description of virtual fields. The format of the VIRFLD statement is:

```
        VIRFLD    NAME=fldname
                  [,BYTES=n]
                  [,START=pos]
                  [,TYPE=t]
                  [,VALUE=value]
                  [,RTNAME=prog]
```

NAME=
: specifies the name of the field. This field name must start with an alphabetic character.

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. This field name must start with an alphabetic character.

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 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}
                   {PL/I }
LANG=              {ASSEM}
                   {RPG  }
,PSBNAME=psbname
```

LANG=

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

HLPI application programs can use any PSB regardless of which language is specified.

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 core image library, care must be taken to avoid duplicate names. The generated PSB name is eight characters 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 3-15 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 OR RPG     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 OR RPG     X
               PSBNAME=STBCPHA          PROGRAM SPECIFICATION BLK NAME
         END
```

Figure 3-15. Sample PSBs for Phase 1

Execution of PSBGEN - Job Control Language

PSBGEN is run as a standard job and requires the following 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 messages are contained in *DL/I DOS/VS Messages and Codes.*
- Assembly listing
  A DOS/VSE 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 core image library.

Coding PSBs for Logical Data Bases

When a physical DBD contains logical relationships, the PCB and the application program still refers 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 3-16 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 OR RPG       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
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 OR RPG       X
      PSBNAME=STBCLGA          PROGRAM SPECIFICATION BLK NAME
END
```

Figure 3-16. Sample PSBs for Phase 2

Coding PSBs for Secondary Indexes

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

PCB Statement

	PCB	TYPE=DB,... ,PROCSEQ=refname

PROCSEQ=refname

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. 'refname' corresponds to the REF value of the ACCESS statement for the secondary index.

Note: If non-unique fields are used, then the sequence of root segments with the same index field value will be effectively random, based on the relative byte address of the source segment. This sequence varies during reorganization of the target data base.

Figure 3-17 shows some of the PSBs 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
* NOTE: LANG=ASSM HAS BEEN SPECIFIED ON THE PSBGEN STATEMENTS.
// EXEC ASSEMBLY
         TITLE 'DL/I ONLINE SAMPLE PROBLEM - INVENTORY AND CUSTOMER    X
               LOAD PSBS'
         PRINT NOGEN
*********************************************************************
*                                                                   *
* THIS PSB IS USED BY THE SAMPLE LOAD PROGRAMS TO LOAD THE CUSTOMER *
* AND INVENTORY DATA BASES.                                         *
*                                                                   *
* NOTE THAT THE LANG PARAMETER ON THE PSBGEN STATEMENT IS CODED FOR *
* ASSEMBLER APPLICATION PROGRAM.                                    *
*                                                                   *
* THE LANG PARAMETER ON THE PSBGEN STATEMENT SHOULD BE CHANGED TO THE *
* LANGUAGE OF THE APPLICATION PROGRAM TO BE EXECUTED IF IT USES THE *
* DL/I CALL INTERFACE.                                              *
*                                                                   *
* THE LANG PARAMETER ON THE PSBGEN STATEMENT DOES NOT HAVE TO CHANGE *
* IF THE EXEC DL/I INTERFACE IS USED.                               *
*                                                                   *
* THE FOLLOWING IS A LIST OF THE SAMPLE LOAD APPLICATION PROGRAMS:  *
*                                                                   *
*     DLZSAM40  -  ASSEMBLER DL/I CALL INTERFACE                    *
*     DLZCBL40  -  COBOL DL/I CALL INTERFACE                        *
*     DLZPLI40  -  PL/I DL/I CALL INTERFACE                         *
*     DLZRPG40  -  RPG RQDLI INTERFACE                              *
*     DLZCBL10  -  COBOL HLPI EXEC DL/I INTERFACE                   *
*     DLZPLI10  -  PL/I HLPI EXEC DL/I INTERFACE                    *
*                                                                   *
*********************************************************************
         PRINT NOGEN                        NO MACRO EXPANSION PRINTING
*********************************************************************
*                                                                   *
*          THE FOLLOWING PCB IS FOR THE INVENTORY DATA BASE         *
*                                                                   *
*********************************************************************
         PCB                                                       X
               TYPE=DB,                 REQUIRED                   X
               DBDNAME=STDIDBP,         FROM DBD MACRO IN DBD ASSEMBLY X
               PROCOPT=L,               LOAD PSB                   X
               POS=S,                   SINGLE POSITIONING  (DEFAULT) X
               KEYLEN=50                CONCATENATED KEY LENGTH
*                                       THE LONGEST CONCATENATED KEY IN
*                                       THE CUSTOMER DATA BASE IS 26.
*                                       50 LEAVES ROOM FOR EXPANSION.
         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 ARE *
* NOT INCLUDED.                                                     *
*********************************************************************
```

Figure 3-17. PSBs Used for the Phase 3 Sample Application (Part 1 of 3)

```
         PCB    TYPE=DB,DBNAME=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
         END
/*
// EXEC LNKEDT
```

Inventory and Customer Data Base PSBs - Logical

```
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
         TITLE 'DL/I ONLINE SAMPLE PROGRAM - INVENTORY AND CUSTOMER    X
               PSB - LOGICAL'
         PRINT NOGEN
*********************************************************************
* THIS PSB IS FOR THE LOGICAL DATA BASES USED BY THE SAMPLE PRINT   *
* PROGRAMS.                                                         *
*                                                                   *
* 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.                            *
*                                                                   *
* THE FOLLOWING IS A LIST OF THE SAMPLE PRINT APPLICATION PROGRAMS: *
*                                                                   *
*     DLZSAM50  -  ASSEMBLER DL/I CALL INTERFACE                    *
*     DLZCBL50  -  COBOL DL/I CALL INTERFACE                        *
*     DLZPLI50  -  PL/I DL/I CALL INTERFACE                         *
*     DLZRPG50  -  RPG RQDLI INTERFACE                              *
*     DLZCBL20  -  COBOL HLPI EXEC INTERFACE                        *
*     DLZPLI20  -  PL/I HLPI EXEC INTERFACE                         *
*                                                                   *
*********************************************************************
         PCB    TYPE=DB,DBNAME=STDIDBL,PROCOPT=G,KEYLEN=50,POS=S,     X
                PROCSEQ=STXININ
         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,DBNAME=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
         SENSEG NAME=STSCSTA,PARENT=STSCCST
         SENSEG NAME=STSCHIS,PARENT=STSCCST
         PSBGEN LANG=ASSEM,PSBNAME=STBICLG
         END
/*
// EXEC LNKEDT
```

Figure 3-17. PSBs Used for the Phase 3 Sample Application (Part 2 of 3)

PSB for the Online Application - Update

```
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
         TITLE 'DL/I ONLINE SAMPLE PROGRAM - CUSTOMER AND INVENTORY    X
               PSB - UPDATE'
         PRINT  NOGEN
*********************************************************************
* THIS IS THE PSB WHICH ALLOWS THE ONLINE SAMPLE PROGRAMS TO UPDATE  *
* THE CUSTOMER AND INVENTORY DATA BASES.  UPDATE CAPABILITY VIA PATH *
* CALL IS SPECIFIED BY PROCOPT=AP.                                  *
*                                                                   *
* LISTED BELOW ARE THE VARIOUS ONLINE SAMPLE APPLICATION PROGRAMS   *
* AND THE CHANGES REQUIRED TO THIS PSB FOR EACH:                    *
*                                                                   *
*     DLZSAM60  -  NO CHANGES REQUIRED                              *
*     DLZCBL60  -  CHANGE TO LANG=COBOL,PSBNAME=STBCUCU             *
*     DLZPLI60  -  CHANGE TO LANG=PL/I,PSBNAME=STBCUPU              *
*     DLZRPG60  -  CHANGE TO LANG=RPG,PSBNAME=SRGCUSU               *
*     DLZCBL30  -  NO CHANGES REQUIRED                              *
*     DLZPLI30  -  NO CHANGES REQUIRED                              *
*                                                                   *
*********************************************************************
         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 3-17. 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 problem program and accepts control statements as input. The PSB for the application program and its related DBD(s) are loaded from a 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 core image library (see Figure 3-18). 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.

Using the DL/I Documentation Aid

When the DL/I Documentation Aid facility is specified, DBD and PSB data information is collected and written to SQL/DS tables. This information can subsequently be accessed directly by ISQL. This provides an easy-to-use facility to monitor the structure of the data base and its definitions. This information can be used to assist in diagnosis of trouble areas, help locate incorrect data, and isolate problems caused by incorrect or incomplete data base defintions.

The DL/I Documentation Aid feature is only available to DL/I users who have SQL/DS and ISQL installed on their VSE system. This facility cannot be run unless sufficient VSAM space has previously been defined for the SQL/DS tables.

For additional information on the DL/I Documentation Aid facility, see the *DL/I DOS/VS Resource Definition and Utilities* manual.

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 comprised of one or more PSB names and, optionally, control parameters for an output destination and/or a DMB generation and/or a userid for SQL/DS. 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.

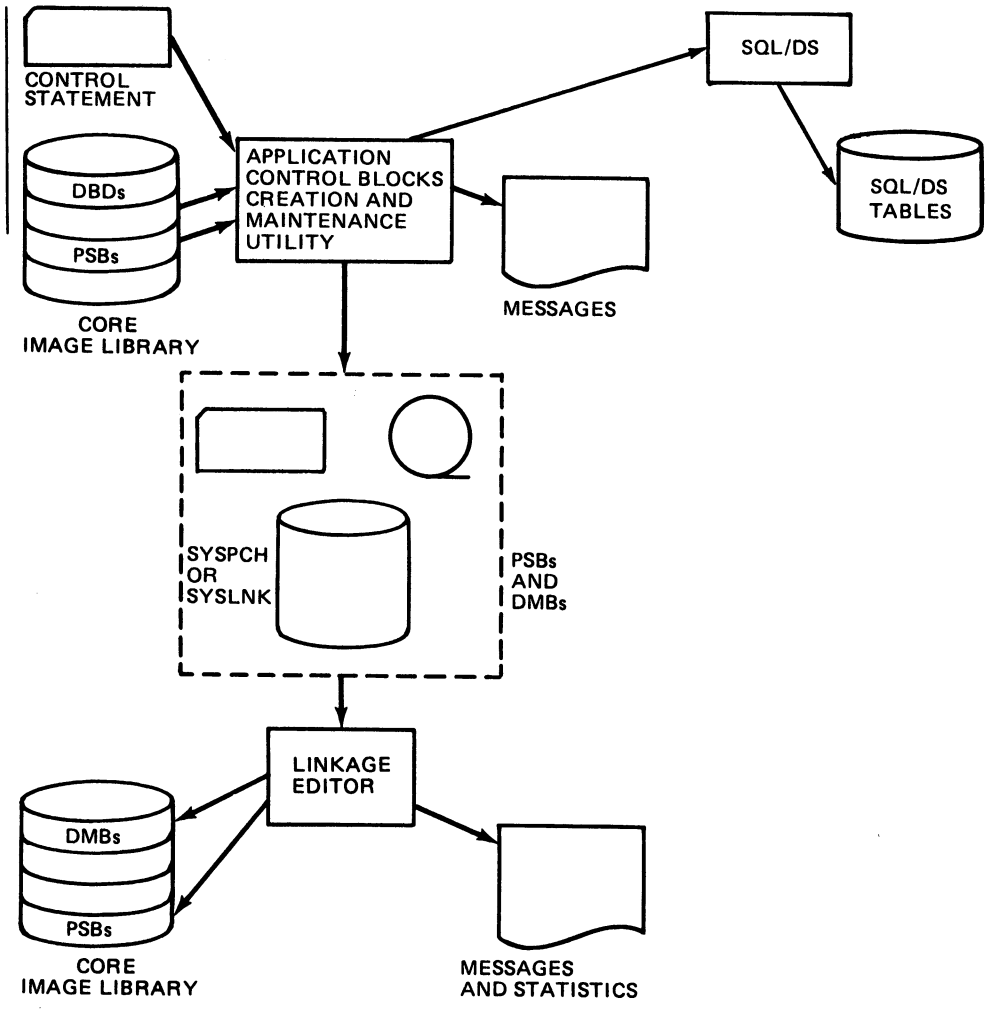


Figure 3-18. DL/I ACB Creation and Maintenance for Each PSB

The format of the control statement is:

```
[label]   BUILD   PSB=(psbname,...)
                  [,OUT=LINK]
                  [,DMB= {COND}]
                         {YES }
                         {NO  }
                  [,USERID=SQLDBA/password]
```

label
: is optional and is useful only for documentation purposes. If specified, it must be a one to eight 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 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.

USERID=SQLDBA/password

If specified in any BUILD statement, data base description (DBD) and program specification block (PSB) data information is to be created and inserted into the DL/I SQL/DS tables. If information already exists for the DBD or PSB in the tables, then the old data information is replaced by the new data information. This parameter is required to initiate the DL/I Documentation Aid. The password is the SQL/DS password currently assigned to the SQLDBA userid.

Notes:

1. This program creates PSB and DMB object modules that contain 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 is on SYSPCH, the necessary 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.
7. Multiple BUILD statements used in a single execution of this utility should all be coded with the same DMB= option. Different options will override each other as follows: NO cancels YES and/or COND, and YES cancels COND.

Job Control Language Requirements

The application control blocks creation and maintenance utility is executed as a standard application program. If you do not specify OUT=LINK in the BUILD statement, a job stream including 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, an 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 VSE core image library. The USERID= parameter indicates that the DL/I Documentation Aid facility is to be run. The job stream indicates that it is to execute in an SQL/DS multi-partition mode.

```
// JOB STJACBGN GENERATE ALL ACBS
// OPTION CATAL,NODECK,DUMP
// EXEC DLZUACB0,SIZE=AUTO
   BUILD PSB=(STBICLD,STBCUSR,STBCUSU),OUT=LINK,DMB=YES
   BUILD PSB=(STBICLG),OUT=LINK,DMB=YES,USERID=SQLDBA/SQLDBAPW
/*
// EXEC LNKEDT
/*
/&
```

ISQL Extract Defines Utility

The DL/I ISQL Extract Defines Utility eliminates the need to manually describe the organization and structure of the DL/I data necessary to retrieve information from DL/I data bases with the Extract Facility of SQL/DS.

This utility creates and stores an ISQL routine composed of ISQL Extract Define commands from data previously gathered and stored in tables with the DL/I Documentation Aid, that is described above. Once the routine is created, it can be run under ISQL to define the necessary DL/I information to the Extract Facility of SQL/DS.

The ISQL Extract Defines Utility reduces the possibility of errors and saves time because the need to manually code the required DL/I information is removed and performed automatically.

For additional information on the ISQL Extract Defines Utility, see the *DL/I DOS/VS Resource Definitions and Utilities* manual.

Control Statement Requirements

The control statement requirements for this program are free form. All parameters are keyword parameters and may continue on additional cards if a non-blank character appears in column 72. Parameter statement and parameters on additional cards may begin in any column. Embedded blanks within the parameter statement are not allowed and characters following a blank after the statement are treated as comments. Comment cards have an asterisk in column 1. More than one EXTRACT parameter statement may be included in a single jobstream.

The format of the control statement is:

[label]	EXTRACT	PSBNAME={psbname } {(psbname,pcbnumber)} ,PCBNAME=pcbname ,DLIPROC=procname ,USERID=SQLDBA/password [,REPLACE]

PSBNAME= {psbname|(psbname,pcbnumber)}

psbname identifies the PSB that is available for extract operations and pcbnumber identifies the PCB within the PSB for extract operations. If the pcb-number is omitted, the pcbnumber defaults to 1.

PCBNAME=pcbname

identifies the DL/I data base view that will be known by the Extract Facility and also the name assigned to the ISQL routine to be executed by the ISQL RUN command.

DLIPROC=procname

identifies the cataloged procedure that will be used to obtain the Job Control Statements necessary to execute the Extract utility of SQL/DS.

USERID=SQLDBA/password

is the userid and password assigned for SQL/DS CONNECT authority. The userid must be SQLDBA.

REPLACE

specifies that if an ISQL routine already exists in the ISQL ROUTINE table that matches the pcbname, the ISQL routine will be deleted from the ISQL ROUTINE table and replaced with an updated ISQL routine.

If REPLACE is not specified, this is the first request to create the ISQL Extract Defines routine in the ISQL ROUTINE table.

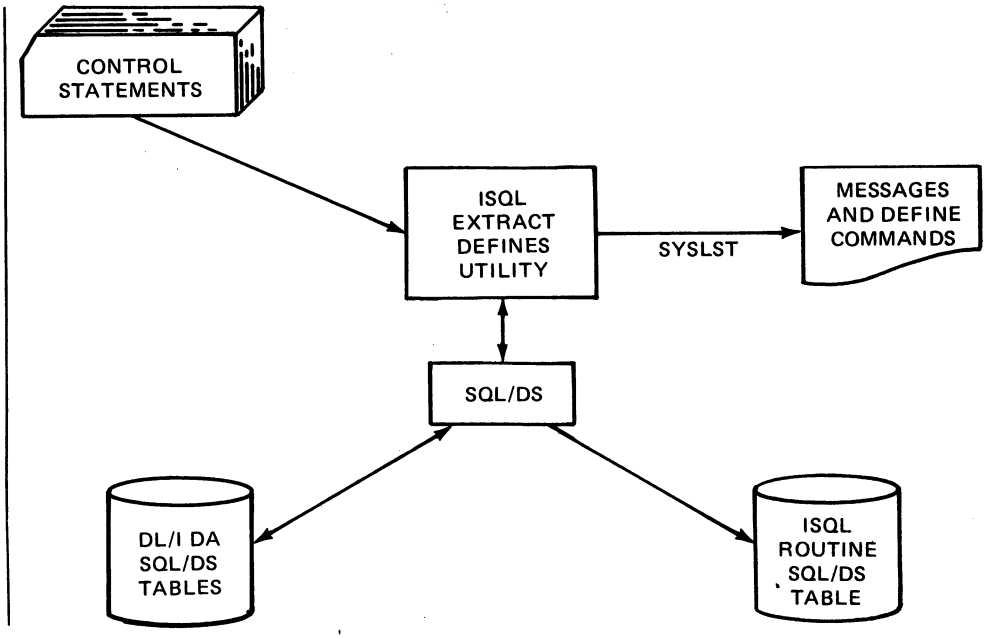


Figure 3-19. ISQL Extract Defines Utility

Job Control Language Requirements

The ISQL Extract Defines Utility is executed as a batch SQL/DS application program. Job control statements for execution are:

- Multiple Partition Mode:

```
// JOB     EXTRACT DEFINES UTILITY
// EXEC    PGM=DLZEXDFP,SIZE=AUTO
(utility parameter statements)
/*
/&
```

- Single Partition Mode:

```
// JOB  EXTRACT DEFINES UTILITY
// EXEC PROC=(identification statement for SQL/DS data base)
// EXEC PGM=ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,PROGNAME=DLZEXDFP'
(utility parameter statements)
/*
/&
```

VSAM Requirements

Before your data bases can be loaded, they must first be defined to VSAM using the Access Method Services utility functions. The Access Method Services must be used to define a VSAM catalog, data space, and 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 the 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 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 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 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.

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 3-20 shows the catalog entries made when the data set (cluster) is defined.

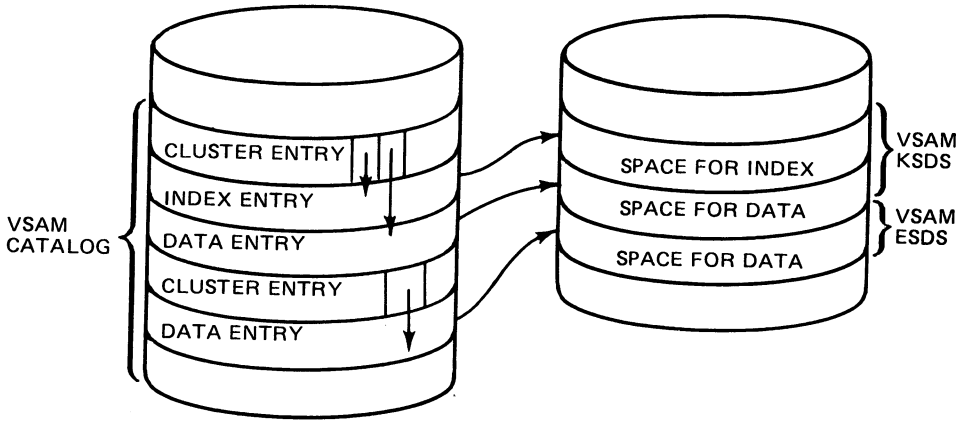


Figure 3-20. Defining VSAM Data Sets

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(29 10)  )                -
                 INDEX (                              -
                        VOLUMES(111111)               -
                        NAME(SAMPLE.CUDX1.INDEX))     -
                 DATA  (                              -
                        NAME(SAMPLE.CUDX1)            -
                        VOLUMES(111111)               -
                        CYL(1 1)                      -
                        FREESPACE(10 10)              -
                        CNVSZ(2048)                   -
                        RECSZ(40 40) )
        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(24 24))
/*
/&
```

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 and its associated secondary index is:

```
171+*,CONTROL INTERVAL SIZE FOR THIS DATA SET IS 2048
182+*,..NR BLKS IN TRK...6..IN CYL...114...
521+*,VSAM DATA SET DESCRIPTIONS
522+*,
523+*,DATA BASE NAME.....................  STDIDBP
524+*,DATA BASE ORGANIZATION.............  HD
525+*,DEVICE TYPE........................  3340
526+*,
527+*,ESDS DATA SET NAME..................  STDIDBC
528+*,CONTROL INTERVAL SIZE..............  2048
529+*,NUMBER RECORDS IN CI................  1
530+*,RECORD LENGTH......................  2038
531+*,
581+*,CONTROL INTERVAL SIZE FOR THIS DATA SET IS 2048
666+*,VSAM DATA SET DESCRIPTIONS
667+*,
668+*,DATA BASE NAME.....................  STXININ
668+*,DATA BASE ORGANIZATION.............  INDEX
670+*,DEVICE TYPE........................  3340
671+*,
672+*,KSDS DATA SET NAME..................  STXININ
673+*,CONTROL INTERVAL SIZE..............  2048
674+*,NUMBER RECORDS IN CI................  113
675+*,RECORD LENGTH......................  18
676+*,KEY LENGTH.........................  6
677+*,RELATIVE KEY POSITION..............  10
678+*,
```

- For the inventory data base, STDIDBP, 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.
- For the secondary index data base, STXININ, the CONTROL INTERVAL SIZE and RECORD LENGTH are used to specify the parameter values for CNVSZ and RECSZ in the DEFINE CLUSTER command for SAMPLE.INVDX. Additionally, the KEY LENGTH and RELATIVE KEY POSITION are also used to specify the parameter values for the KEYS parameter of the DEFINE CLUSTER command for SAMPLE.INVDX.
- The above explanations also apply to the Customer data base and its indexes. Using the output listing of the DBDGEN for parameter information assures 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 load commands. This is job STJLDCST in the sample jobstream (DLZSAM40). Because it is necessary to use DL/I commands to load a data base, the load program will be discussed in Chapter 4, following the presentation of the DL/I commands.

Chapter 4: Processing Data Bases (Batch Considerations)

About This Chapter

This chapter is divided into seven parts.

- General introduction to DL/I data base processing.
- Introduction of the basic DL/I High Level Programming Interface (HLPI) commands. Formats of the HLPI commands and sample presentations of commands for COBOL and PL/I are provided.
- Restrictions for the planning and implementation of application programs.
- CICS/VS translator and job control statements for COBOL and PL/I for batch and MPS batch applications.
- Loading data bases with logical relationships and loading HD indexed and HD randomized data bases.
- Processing of logical data bases, which are implemented with the DL/I logical relationships function.
- Processing with secondary indexes.

Introduction to Data Base Processing

Data base processing is transaction oriented. 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

An HLPI application program is written in either COBOL or PL/I. A translation step is required prior to compilation to convert HLPI commands to COBOL or PL/I statements. The program is then compiled through the user-selected language

compiler and placed in the appropriate program library, after it is link-edited with the appropriate language interface module. (See *DL/I DOS/VS Application Programming: High Level Programming Interface.*)

Interface Components

A DL/I batch application program executes in a manner similar to any other 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 system 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 uses and the kind of access (retrieve, update, insert, delete) the program is allowed to do. The PSBs are maintained in the core image library. The coding and generation of PSBs is described in Chapter 3 of this manual.

During initialization, DL/I loads the application program and its associated PSB from the library. The DL/I modules interpret and execute data base requests issued by the program.

The application program interfaces with DL/I in the following ways:

- A PSB scheduling statement.
- DL/I Interface Block (DIB) variables, which are defined in the application program by the HLPI translator, and which are updated by DL/I after each command is completed, receive return information from DL/I.
- I/O areas for passing data between data bases and application programs.
- DL/I commands specifying processing functions.
- A termination statement.

The DIB and I/O areas are described in the program's data declaration portion. The DIB declaration is generated automatically by the translator. Program entry and DL/I commands are described in the program's procedural portion. Figure 4-2

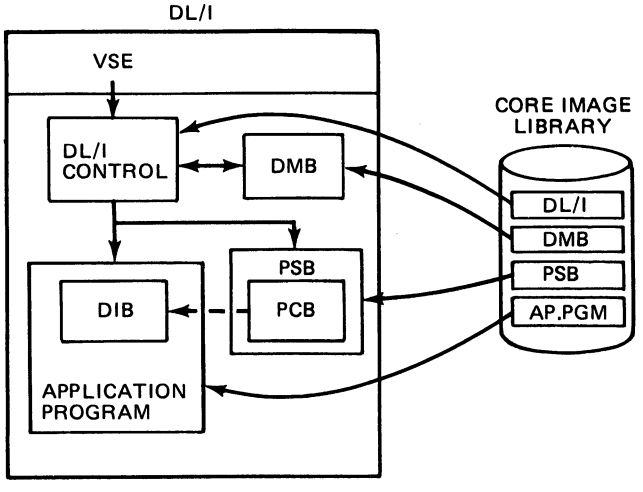


Figure 4-1. DL/I Interface with an Application Program

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 the operating system gives control to the DL/I control facility, the DL/I control program in turn passes control to the application program (through the entry point as defined below).

COBOL: The following statement must be the first in the procedure division.

```
ENTRY 'DLITCBL'.
```

PLI: There is no special requirement.

High Level Programming Interface

The DL/I High Level Programming Interface (HLPI) is an easy-to-use method of performing all of the functions necessary for processing DL/I data bases in a batch, multiple partition support (MPS) batch, or CICS/VS online environment. It consists of commands similar to those in the CICS/VS command language. You can code these commands in application programs written in either COBOL or PL/I Optimizer Language.

You must have CICS/VS to use the HLPI because the HLPI commands are translated by the CICS/VS translator into call statements to DL/I. Before compiling a program using the HLPI, you must execute the CICS/VS EXEC translator, as a separate job step, to convert the commands into statements appropriate to the host language.

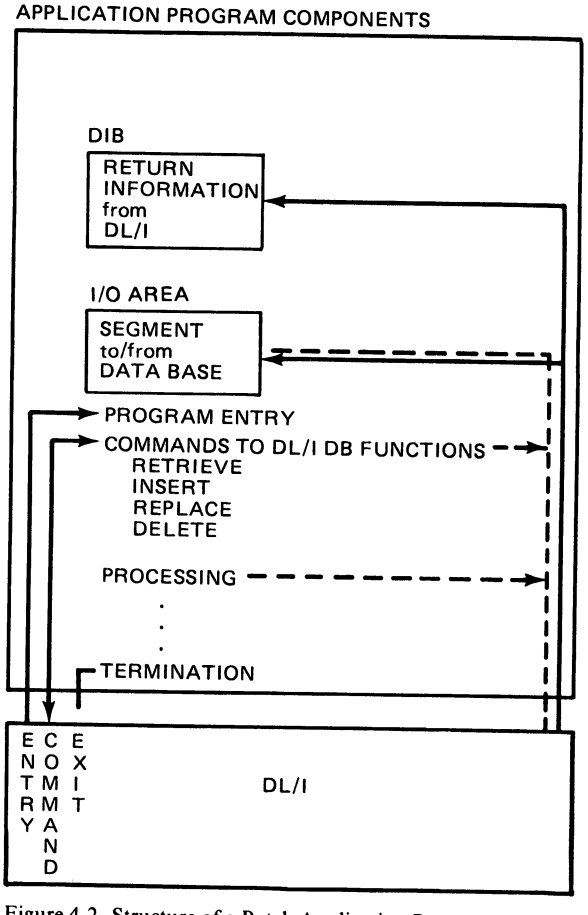


Figure 4-2. Structure of a Batch Application Program

DL/I Functions and Associated HLPI Commands

The primary unit of data in a DL/I data base is called the segment. A number of different segment types comprise a data base. For application programs to distinguish between them, each segment type is assigned a name. Any number of occurrences of a particular segment type can be stored in the data base. Segments, which are placed in a hierarchical structure in a data base, are always referenced in the hierarchical sequence of top-to-bottom, left-to-right, front-to-back, as indicated by the numbers 10 through 21 in Figure 4-3.

There are several functions that DL/I must perform on these segments as the application program processes information stored in data bases. A brief description of these functions, and the format used to code the HLPI commands, are described below.

HLPI Commands

The DL/I High Level Programming Interface is made up of commands similar in syntax to those in the CICS/VS command language. You can code these commands in an application program written in either COBOL or PL/I Optimizer Language.

The syntax of HLPI commands is:

```
trigger  function  [options and arguments] command-delimiter
```

- Trigger identifies the statement to the translator as being a DL/I HLPI command.
- Function names the operation the command is to perform.

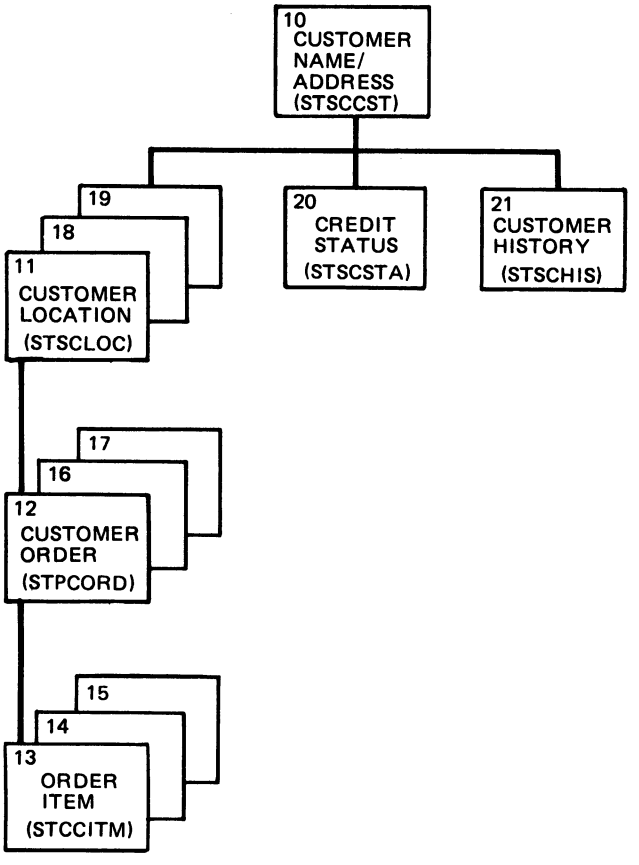


Figure 4-3. Customer data base record

- Options provide information needed to perform the operation named by function.
- Arguments are included, within parentheses, with certain options to pass names, constants, or variable values to the translator.

This is an example of a command:

```
EXECUTE DLI  TERMINATE   ;
|_________|  |________|  |_|
  trigger    function    command-
                         delimiter
```

Here is an example of another command (notice the free-form layout):

```
EXECUTE DLI  GET NEXT  USING PCB(1) SEGMENT(SKILL)
                         WHERE(CODE=ARTIST) INTO(IOAREA)  END-EXEC
|_________|  |_______|  |_____________________________________|  |_______|
  trigger    function          options with arguments            command-
                                                                 delimiter
```

Every command must include a trigger, a function, and a command-delimiter. In addition, most commands include options and arguments. (See *DL/I DOS/VS Application Programming: High Level Programming Interface* for a description of the options and arguments.)

Elements of the HLPI Command Language

The following represents the elements of the HLPI commands.

Trigger—Execute DLI

The trigger is a unique identifier for an HLPI command. You must code it at the beginning of every command. You may use either EXECUTE DLI or EXEC DLI. These keywords are reserved for this use only. You must not use either combination of them for any other purpose in your program.

Functions

Each command must have a function. Functions are coded within a command to specify the operation you wish to perform. The associated options and arguments coded with the function provide the information necessary to complete the operation.

The HLPI functions are:

- GET NEXT — sequential retrieval of segments
- GET NEXT IN PARENT — retrieval of segments under a specific parent
- GET UNIQUE — retrieval of a specific segment
- INSERT — inserting new segments into a data base
- REPLACE — replacing obsolete or incorrect segments
- DELETE — deleting obsolete or incorrect segments
- LOAD — loading (initially) segments into an empty data base.

The function that takes a DL/I checkpoint is:

- CHECKPOINT — checkpointing a data base during program execution.

The functions that schedule and terminate PSBs in an online operation are:

- SCHEDULE — scheduling a PSB for the use of a data base
- TERMINATE — terminating a PSB to release a data base.

Note: See Chapter 5 for a description of the SCHEDULE and TERMINATE commands.

Retrieving Segments

Once segments have been loaded into a data base, you can retrieve them by issuing one of three different commands called the Get commands. They include: GET NEXT, GET NEXT IN PARENT, and GET UNIQUE.

GET NEXT: To retrieve segments sequentially from a data base, DL/I maintains a position pointer that remembers your current position in the data base. Each time you request DL/I to sequentially retrieve a segment, DL/I retrieves the segment pointed to by the position pointer and updates the pointer to the next segment.

The GET NEXT command retrieves the next segment—the one pointed to by the position pointer. If your program executed a continuous series of GET NEXT commands, without specifying a particular segment, it would eventually retrieve, in hierarchical order, every segment in the data base. A GET NEXT command, as coded in a PL/I application without specifying a particular segment, looks like this:

```
EXECUTE DLI GET NEXT INTO(AREA);
```

If this was the first command to be executed in your program, the segment that would appear in AREA would be the root segment of the first record in the data base. If this was the customer data base as shown in Figure 4-3, this segment would be the customer name/address segment (STSCCST) for the first data base record. The position pointer would then point to the customer location segment (STSCLOC). If the same GET NEXT command was executed again, that segment would appear in AREA. The position pointer would then point to the first customer order segment. Executing the GET NEXT again would retrieve that segment (STPCORD). This process, if continued, would proceed through the data base from top-to-bottom, left-to-right, front-to-back, in hierarchical order.

A GET NEXT command, as coded in a PL/I application with a particular segment specified, looks like this:

```
EXECUTE DLI GET NEXT SEGMENT(STSCLOC) INTO(AREA);
```

In this case, the segment that would appear in AREA would be the first customer location segment (STSCLOC). If the same GET NEXT command were executed again, the next customer location (18) would appear in AREA. If the GET NEXT command was continuously executed, the same process would continue until all customer location segments are retrieved.

GET NEXT IN PARENT: The HLPI provides a variation of the GET NEXT command that is called GET NEXT IN PARENT. Segments higher than others in the hierarchical structure of a data base are called parent segments of those below them in the same hierarchical path. The GET NEXT IN PARENT command makes it possible for you to sequentially retrieve the data from all of the segments that are children of a particular parent segment. The parentage that applies to this command is that established by the last previous Get command that was not a GET NEXT IN PARENT command.

For example, if the GET NEXT command for SEGMENT(STSCLOC) was executed successfully, a GET NEXT IN PARENT would use STSCLOC as the designated parent. A GET NEXT IN PARENT command can be coded with or without a SEGMENT option. Coded without the SEGMENT option, it looks like this:

```
EXECUTE DLI GET NEXT IN PARENT INTO(AREA);
```

In this case, with parentage established at the customer location (STSCLOC), continuous execution of this command would retrieve the customer order (12), the order items (13, 14, and 15), and the remaining orders (16 and 17) in this order, for that particular customer location.

A GET NEXT IN PARENT command coded with the SEGMENT option looks like this:

```
EXECUTE DLI GET NEXT IN PARENT SEGMENT(STPCORD) INTO(ORDIO);
```

In this case, with parentage established at the customer location (STSCLOC), continuous execution would retrieve all the customer order segments (12, 16, and 17), in this order for that particular location.

GET UNIQUE: The third type of Get command is GET UNIQUE. This command provides random retrieval of any segment in the data base. When you want to retrieve segments sequentially, GET NEXT provides that function. A GET UNIQUE command allows you to establish position in a data base for subsequent GET NEXT or GET NEXT IN PARENT commands.

Unlike these two GET NEXT commands, GET UNIQUE does not depend on the position pointer; thus, it is able to retrieve a segment from any location in the data base—even from levels higher in the hierarchy than the position pointer indicated before the GET UNIQUE command was executed. You must code the SEGMENT option and argument in a GET UNIQUE command to indicate the type you want to retrieve. This is called unqualified segment selection and looks like this:

```
EXECUTE DLI GET UNIQUE
        SEGMENT(STSCCST) INTO(STSCCST);
```

In this case, the first occurrence of the segment STSCCST is retrieved.

When you need to retrieve a specific occurrence of a segment type, you must qualify the segment selection. You can do this by coding the WHERE option after the SEGMENT option. The WHERE option provides the information DL/I needs to locate the specific segment you want. (See "Qualified Segment Selection" later in this Chapter.)

A GET UNIQUE command with qualified segment selection looks like this:

```
EXECUTE DLI GET UNIQUE
        SEGMENT(STSCCST) INTO(STSCCST) WHERE(STQCCNO=CUSTVAR);
```

where STQCCNO is the name of a field in the customer segment STSCCST and CUSTVAR is the name of a variable previously defined in the host language of your program.

Inserting Segments

The INSERT command is used to add new segments to an existing data base. Your program builds the data to be contained in the new segment in the area you specify in the INSERT command. When the command is executed, DL/I takes the data from the area you specified and adds the segment to the data base. The format of the INSERT command is:

```
EXECUTE DLI INSERT SEGMENT(STSCLOC) FROM(LOCIO);
```

Notice the option FROM, which is used in place of INTO to indicate that data is to be transferred from an area in your program to the data base.

Before the INSERT command can be executed, position in the data base should be established. This can be done by coding one of the three types of Get commands or with a preceding INSERT command, provided the position you desire has been established.

Replacing Segments

The REPLACE command replaces the data in a segment in the data base with different data. Before executing a REPLACE command, your program must have retrieved the existing data from the data base with a Get command. Your program can then modify that data and execute the REPLACE command to place the modified data in the data base. The format of a REPLACE command (assuming the preceding GET command was executed) is:

```
EXECUTE DLI REPLACE SEGMENT(STSCLOC) FROM(LOCIO);
```

Deleting Segments

The DELETE command enables you to delete segments from a data base when the information they contain becomes obsolete or incorrect. Before your program executes a DELETE command, it must have retrieved the segment to be deleted by executing one of the Get commands. The DELETE command then removes the named segment and all of its dependent children from the data base. Because of this deletion of dependent children, you must use caution when you issue the DELETE command. The format of the DELETE command (assuming a Get command was executed that specified the same segment) is:

```
EXECUTE DLI DELETE SEGMENT(STSCLOC) FROM(CUSTIO);
```

The customer location segment and its dependent children—customer order, and order item segments (numbers 11 through 17) under it (Figure 4-3), would be removed from the data base.

Loading a Data Base

The LOAD command is used in a batch application program to load initially segments into an empty data base. No other HLPI commands are allowed in this program. The LOAD command is not used in any batch or online application. (See "Data Base Load Processing" later in this Chapter.)

Data must be loaded into the data base in the form of segments. The format of the LOAD command is:

```
EXECUTE DLI LOAD SEGMENT(STSCCST) FROM(SEGDATA);
```

Checkpointing a Data Base

If your program fails abnormally before its execution is complete, any changes you made to the data base to that point should be backed out (restored to their previous state). This eliminates the possibility that the data base is left in a partially updated condition for access by other application programs. Backout is performed by a DL/I utility program for batch application programs, and by CICS/VS dynamic transaction backout for MPS batch and online application programs.

If your program is a long-running one, you can reduce the amount of backout that might be necessary by taking checkpoints. When a logically complete set of updates has been completed, and your program is about to begin another set of updates, take a checkpoint by issuing the CHECKPOINT command. This signals the backout utility to stop at this point.

The formats of the CHECKPOINT command are:

```
EXECUTE DLI CHECKPOINT ID(CHKPID);
```
and
```
EXECUTE DLI CHECKPOINT ID('CHKP0007);
```

where CHKPID is the name of an eight-byte character string that uniquely identifies this checkpoint. Alternatively, the CHECKPOINT ID can be coded in the commands, enclosed in single quotes, as in the second format.

Specifying the PCB

You can specify the particular PCB that defines the view of a data base to be used for a command performing a data base function. (A data base function is one that requires the data base to be accessed.) You do this in the command by coding the PCB option. If you do not specify the PCB option, DL/I uses the first PCB defined in the currently scheduled PSB. If you code the PCB option, it follows the function and looks like this:

```
EXEC DLI
     GET NEXT USING PCB(3)                    (PCB specification)
     SEGMENT(STSCCST) INTO(STSCCST) SEGLENGTH(106)
END-EXEC.
```

This example indicates the use of the third PCB in the currently scheduled PSB. The number specified must not be greater than the number of PCBs generated for the PSB. The first PCB is PCB(1), the second is PCB(2), and so on.

Selection of Segments

In each command that accesses a data base, you must specify to DL/I the segment, or segments, that you want it to operate on (except in GET NEXT and GET NEXT IN PARENT commands, where it is optional).

Object Segments

The object segment in a data base function command is the segment at the lowest hierarchical level in which you are interested.

Object segment selection looks like this:

```
EXEC DLI
     GET NEXT USING PCB(3)
     SEGMENT(STSCLOC) INTO(STSCLOC) SEGLENGTH(106) (object segment)
END-EXEC.
```

where STSCLOC is the name of the object segment. INTO or FROM (described under "Segment I/O Area" below) must be coded with the object segment selection specification to identify an I/O area for the segment data. The object segment selection specification must be coded after the last of any parent segment selection.

Parent Segments

A parent segment is a segment used to help identify the hierarchical path leading to the segment at the lowest level in which you are interested (the object segment). A maximum of 14 parent segments can be specified in one data base function command corresponding to the 14 possible hierarchical levels above the lowest level in a DL/I data base hierarchy.

Parent segments, when specified, must be coded in hierarchical order from top to bottom, the highest level parent segment first. It is not necessary to specify a parent segment on every hierarchical level.

Parent segment selection looks like this:

```
EXEC DLI
     GET UNIQUE USING PCB(3)
     SEGMENT(STSCCST) WHERE(STQCCNO=NUMSAV) (parent segment)
     SEGMENT(STSCLOC) INTO(STSCLOC);        (object segment)
```

where STSCCST is the name of the parent segment.

Segment I/O Area

To indicate that you want a segment to be transferred to or from a segment I/O area, you must code the INTO or FROM option following the corresponding SEGMENT option in a data base command. INTO is coded in commands that retrieve segments from the data base (GET UNIQUE, GET NEXT, and GET NEXT IN PARENT) into a segment I/O area. FROM is coded in commands that modify the data base (INSERT, REPLACE, DELETE, and LOAD) with data from a segment I/O area.

The INTO or FROM argument identifies an area in your program large enough to contain the segment being transferred. When data transfer is requested for one or more parent segments in this manner, that request is known as a *path call.* Coding the SEGMENT option for a parent segment, without also coding INTO or FROM, indicates that the named segment is to be used in establishing the hierarchical path to the object segment named in the command, but that you do not want data for that segment to be transferred. Within a command, SEGMENT options both with and without FROM or INTO can be used. Data is transferred for only those segments with FROM or INTO specified.

A path call using INTO looks like this:

```
EXEC DLI
     GET UNIQUE USING PCB(3)
     SEGMENT(STSCCST) WHERE(STQCCNO=NUMSAV) INTO(STSCCST)
     SEGMENT(STSCLOC) WHERE(STQCLNO=LOCSAV) INTO(STSCLOC);
```

A "path" is established from the parent segment (STSCCST) to the object segment (STSCLOC). At this point, data is trasferred INTO the I/O areas (STSCCST and STSCLOC) defined in your application program.

In case of a segment-not-found condition in a path call, data is transferred for all segments in the path, for which FROM or INTO was specified, above that which could not be found.

A similar path call using FROM would look like this:

```
EXEC DLI
     REPLACE USING PCB(3)
     SEGMENT(STSCCST) FROM(STSCCST)
     SEGMENT(STSCLOC) FROM(STSCLOC);
```

When a REPLACE command follows an associated path call Get command, FROM must be coded in the REPLACE command for each segment, of those previously retrieved, that is to be replaced.

Processing option "P" must be specified during PSB generation for any segment to be used in path calls. Otherwise, the program is terminated with an AM status code.

You can insert multiple segments in a hierarchical path with one INSERT command. However, this is not permitted if a logical child segment is present in the path (see *DL/I DOS/VS Data Base Administration*).

Segment Length

The SEGLENGTH option defines the length of the segment named by the SEGMENT option.

In an application program using COBOL as the host language, you must code the SEGLENGTH option if you code the INTO or FROM option. This "length" requirement in an HLPI COBOL program is due to a COBOL language restriction. This restriction makes it impossible to determine the declared length of an IOAREA at application program execution time. PL/I does not have this restriction. In a PL/I program, SEGLENGTH is optional. If you do not code SEGLENGTH in PL/I, the length defaults to the length of the area named in the INTO or FROM option.

The length of the segment is defined in the host language. This length can be any expression that converts to the integer data type. If the expression is a variable, it should be declared as a binary halfword value.

You code the SEGLENGTH option following the SEGMENT option like this:

```
EXEC DLI
     GET UNIQUE PCB(3)
     SEGMENT(STSCCST) INTO(STSCCST) SEGLENGTH(106)
END-EXEC.
```

where the length of the segment is 106 bytes.

You are responsible for ensuring that the value of SEGLENGTH, if specified, is the proper value. On a GET command, storage in your application program can be overlaid if the I/O area is not large enough to contain the retrieved segment. Invalid data can be placed in the data base if the I/O area length is incorrectly specified in an INSERT or REPLACE command.

Qualified Segment Selection

Qualified segment selection causes DL/I to search for an occurrence of a segment that contains a field with data that you specify.

Any segment field defined to DL/I can be used in segment selection. However, for performance reasons, qualification of root segments using fields other than the key field should be avoided. DL/I has to scan the data base sequentially for such requests. Qualification of dependent segments on non-key fields is acceptable and should be used as required.

Qualified segment selection is coded after the associated SEGMENT option and looks like this:

```
EXEC DLI
     GET UNIQUE USING PCB(3)
     SEGMENT(STSCCST) WHERE(STQCCNO=NUMSAV) FIELDLENGTH(6)
     SEGMENT(STSCLOC) WHERE(STQCLNO=LOCSAV) FIELDLENGTH(6)
     SEGMENT(STPCORD) WHERE(STQCODN=ORDSAV) INTO(STPCORD)
     FIELDLENGTH(12) SEGLENGTH(55)
END-EXEC.
```

The WHERE option provides segment qualification and operates as follows:

DL/I examines the data in the field you name, in each occurrence of the segment type specified in the SEGMENT option, and compares it with the value specified in the fields defined in your application program. DL/I stops its examination when it finds a value that satisfies the relationship or reaches the last occurrence of the named segment type.

In this example, the field STQCCNO in the segment STSCCST is compared for an equal value in NUMSAV, a variable defined in the host language. When a match is found, the field STQCLNO in segment STSCLOC is compared for an equal value in the field named LOCSAV, and so on. The length of the segment's field must be equal to the length of the field you specified in your application program.

FIELDLENGTH must be specified when the host language is COBOL, but need not be coded when the host language is PL/I. When not specified in PL/I, the value of FIELDLENGTH defaults to the length of the field named in your application program. When you code FIELDLENGTH, the expression you use can be any expression in PL/I that converts to an integer data type. In COBOL, the expression must be a constant or a variable that converts to an integer data type. If the expression is a variable, it should be declared as a binary halfword value.

When comparing fields of the segment with the name you specified, the operator may be any one of the following:

>	(greater than)
<	(less than)
=	(equal to)
¬=	(not equal to)
<=	(less than or equal to)
>=	(greater than or equal to)
=<	(equal to or less than)
¬>	(not greater than)
=>	(equal to or greater than)
¬<	(not less than)

The WHERE option may also use the Boolean operators AND and OR. The AND operator connects qualification conditions to form a set. The OR operator separates the sets and marks the beginning of a new set of conditions. For example, the following WHERE clause could be used in the EXEC DLI step:

```
SEGMENT (STSCCST) WHERE (STQCCNO = NUMSAV AND STQCLNO < LOCSAV
AND STQCLNO ¬< C800 OR STQCODN ¬> ORDSAV)
```

In this example, DL/I looks at the first occurrence of the named segment type. It verifies that STQCCNO equals the value stored in NUMSAV, that STQCLNO is less than the value stored in LOCSAV, and that STQCLNO is not less than the value stored in C800. If these requirements are all met, DL/I does not look any further. The segment is retrieved and DL/I looks at the next segment of this named type. If these requirements are not all met, DL/I then looks at the next set which is started by the OR operator. The next set requires that STQCODN not be greater than the value stored in ORDSAV. If this condition is met, the segment will be retrieved. If not, then DL/I looks at the next segment of this named type and begins the Boolean comparison again. This continues until the last occurrence of the named segment type.

A single WHERE clause may have a maximum of 12 qualification conditions connected by a maximum of 11 Boolean (AND, OR) operators in any combination.

Parentheses are not allowed within the WHERE clause and the Boolean operators AND and OR must be surrounded by blanks.

Command—Delimiter

Every DL/I HLPI command that you code must include a command-delimiter to identify the end of the command to the translator. The particular delimiter you code depends on which host language you are using. For ANS COBOL programs, the command-delimiter is:

```
END-EXEC
```

This allows the command to be included within a THEN clause without being the only statement of the THEN clause.

In PL/I Optimizer programs, the command-delimiter is the semicolon, as with all other PL/I statements.

Terminating the Program

Batch and MPS Batch

In the batch and MPS batch environments, at the completion of the execution of your program, control must be passed back to DL/I. This is done by coding an exit statement in your program that looks like this, depending on the host language you are using:

```
COBOL:  GOBACK.
PL/I:   RETURN;
```

The GOBACK or RETURN statement returns control to DL/I. After all DL/I resources are released and the data bases are closed, DL/I returns control to the operating system.

Note: STOP RUN cannot be used in a COBOL program as an exit statement, because control would not be returned to DL/I to allow it to release its resources and close the data bases and log. This will probably result in lost data. However, it can be coded after GOBACK. This prevents the compiler from giving a warning message and automatically generating a STOP RUN.

Status Codes

After processing an HLPI command, control is returned to your application program at the next sequential instruction following the command. So that you can check that the command has completed successfully, and has performed the operation you intended on the data you specified, DL/I returns a two-character status code in the DIB (DL/I Interface Block).

The first thing you should do in your program after each command is to test DIBSTAT for the various status codes and take appropriate action. The status codes that could be returned in DIBSTAT are GA, GB, GE, GK, II, LB, NE, TG, and bb.

A complete list of status codes is shown in Figure 4-4. An example of how to test the DIBSTAT for a status code follows:

```
EXEC DLI...
IF DIBSTAT = '  ' NEXT SENTENCE
   ELSE IF DIBSTAT = 'GE' GO TO LISCUS
        ELSE GO TO REALER.
```

It is more convenient to directly test the regular exceptions inline instead of branching to a status code check routine. 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.

STATUS CODE	COMMANDS										COMMAND COMPLETED	ERROR IN CMD or CONVERSION	I/O or SYSTEM ERROR	DESCRIPTION
	GET UNIQUE	GET NEXT	GET NEXT IN PARENT	DELETE	REPLACE	LOAD	INSERT	CHECKPOINT	SCHEDULE	TERMINATE				
AB	X	X	X	X	X	X	X					X		SEGMENT I/O AREA REQUIRED, NONE SPECIFIED IN COMMAND
AC	X	X	X			X	X					X		HIERARCHICAL ERROR IN SEGMENT SELECTION
AD						X			X	X		X		INVALID FUNCTION PARAMETER
AH						X	X					X		COMMAND REQUIRES SEGMENT SELECTION, 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 QUALIFIED SEGMENT SELECTION FORMAT
AK	X	X	X			X	X					X		INVALID FIELD NAME IN COMMAND
AM	X	X	X	X	X	X	X					X		COMMAND 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 COMMAND
DX				X								X		VIOLATED DELETE RULE
GA		★	★								★			CROSSED HIERARCHICAL BOUNDARY INTO HIGHER LEVEL (RETURNED ONLY ON COMMANDS WITHOUT SEGMENT SELECTION)
GB		★												END OF DATA SET, LAST SEGMENT REACHED
GE	★	★	★				★							SEGMENT OR PARENT SEGMENT NOT FOUND
GK		★	★								★			DIFFERENT SEGMENT TYPE AT SAME LEVEL RETURNED (RETURNED ON UNQUALIFIED COMMANDS ONLY)
GP			X									X		A GNP COMMAND 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						X								KEY FIELD OF SEGMENTS OUT OF SEQUENCE
LD						X								NO PARENT FOR THIS SEGMENT HAS BEEN LOADED
LE						X								SEQUENCE OF SIBLING SEGMENT NOT THE SAME AS DBD SEQUENCE
NA					X							X		DATA IN SEARCH OR SUBSEQUENCE FIELD HAS BEEN CHANGED
NE				★	★		★				★		★	INDEX MAINTENANCE CANNOT FIND SEGMENT
NI				X	X	X	X						X	INDEX MAINTENANCE UNABLE TO OPEN INDEX DATA BASE
					X	X	X					X		DUPLICATE KEY FOUND FOR INDEX DATA BASE
NO				X	X	X	X						X	I/O ERROR
						X								LOADING DUPLICATE SECONDARY INDEX POINTER SEGMENT
RX					X							X		VIOLATED REPLACE RULE
TA									X				X	PSB NOT IN DIRECTORY

★ Indicates status code returned in DIB.

X Indicates status code that could be expected as an error situation.

Figure 4-4. DL/I Status Codes (Part 1 of 2)

STATUS CODES	COMMANDS													DESCRIPTION
	GET UNIQUE	GET NEXT	GET NEXT IN PARENT	DELETE	REPLACE	LOAD	INSERT	CHECKPOINT	SCHEDULE	TERMINATE	COMMAND COMPLETED	ERROR IN CMD or CONVERSION	I/O or SYSTEM ERROR	
TC									X			X		TASK ALREADY SCHEDULED
TE									X				X	PSB INITIALIZATION ERROR
TF									X				X	PSB NOT AUTHORIZED
TG										★				TERMINATE ATTEMPTED WHEN PSB NOT SCHEDULED
TH	X	X	X	X	X	X	X	X				X		DATA BASE COMMAND ATTEMPTED WHEN PSB NOT SCHEDULED
TI								X				X		INVALID PATH INSERT
TJ	X	X	X	X	X	X	X	X	X	X			X	DL/I NOT ACTIVE
TK									X				X	DATA BASE NOT ACTIVE
TL									X				X	SCHEDULING CONFLICT WITH MPS TASK
TN	X	X	X	X	X	X	X	X	X	X		X	X	INVALID SYSTEM DIB ADDRESS
TO					X							X		PATH REPLACE ERROR
TP	X	X	X	X	X	X	X	X				X		INVALID PCB INDEX
V1					X	X	X					X		INVALID LENGTH FOR VARIABLE LENGTH SEGMENT
V2	X	X	X	X	X	X	X					X		SEGLENGTH MISSING OR INVALID
V3	X	X	X				X					X		FIELD LENGTH MISSING OR INVALID
V4	X	X	X	X	X	X	X					X		INVALID LENGTH FOR VARIABLE LENGTH SEGMENT
V5	X	X	X		X		X					X		INVALID OFFSET
V8	X	X	X									X		KEY FEEDBACK LENGTH MISSING OR INVALID
XD								X						ERROR DURING DATA BASE BUFFER WRITE OUT
XH								X						DATA BASE LOGGING NOT ACTIVE
XR								X						ERROR DURING CHECKPOINT PROCESSING FOR MPS RESTART
bb	★	★	★	★	★	★	★	★	★	★	★			COMMAND COMPLETED SUCCESSFULLY

★ Indicates status code returned in DIB.

X Indicates status code that could be expected as an error situation.

Note: Information about the the DL/I Status Codes is contained in the *DL/I DOS/VS Messages and Codes* manual.

Figure 4-4. DL/I Status Codes (Part 2 of 2)

DIB (DL/I Interface Block)

DL/I returns a status code, and other information, to your program through the DIB (DL/I Interface Block). The status code should be checked to ensure that the function was performed as expected.

There is one DIB provided for each external procedure. The contents of the DIB, at any given moment, reflects the status of the last HLPI command executed in that procedure. DIB information required by an external procedure that has not issued an HLPI command must be passed to that procedure by your application program.

Labels you can use to access the variables in the DIB are automatically generated in your program by the CICS/VS translator. (These labels are reserved and you must not redefine them in your program.)

The way the DIB variables are defined for each host language is:

For COBOL:

```
DIBVER    PICTURE X(2)
DIBSTAT   PICTURE X(2)
DIBSEGM   PICTURE X(8)
DIBFLAG   PICTURE X(1)
DIBSEGLV  PICTURE X(2)
DIBKFBL   COMP PIC S9(4)
```

For PL/I:

```
DIBVER    CHAR(2)
DIBSTAT   CHAR(2)
DIBSEGM   CHAR(8)
DIBFLAG   CHAR(1)
DIBSEGLV  CHAR(2)
DIBKFBL   FIXED BIN(15)
```

- DIBVER is the version of the translator used to translate the application program.
- DIBSTAT is the DL/I status code.
- DIBSEGM is the name of the lowest level segment retrieved. DIBSEGM should be ignored following a CHECKPOINT, SCHEDULE, or TERMINATE command.
- DIBFLAG is a flag indicating that an online task has to wait for a resource owned by an MPS batch task (DIBFLAG=X'FF').
- DIBSEGLV is the hierarchical level of the lowest level segment retrieved. DIBSEGLV should be ignored following a CHECKPOINT, SCHEDULE, or TERMINATE command.
- DIBKFBL is a field containing a numerical value representing the actual length of the concatenated key in the PCB when used with Key Feedback.

Presentation of Command Examples

The HLPI commands are illustrated in two examples, one for COBOL and the other for PL/I, in Figure 4-5 and Figure 4-6. The status codes that can be returned for the commands are shown in Figure 4-4. Status codes for each command are discussed in *DL/I DOS/VS Application Programming: High Level Programming Interface.*

```
1    CBL XOPTS(DLI)...
     ID DIVISION.
     PROGRAM-ID. DLICOBLA.
           •
           •
     DATA DIVISION.
     WORKING-STORAGE DIVISION.
     01  STSCCST.
         03  .......
2    01  STSCLOC.
         03  .......
     01  STPCORD.
         03  .......
     01  STLCITM.
         03  .......
     01  SAVE-AREAS.
3        02  LOCSAV   PIC X(6).
         02  ORDSAV.
             03  .......
         02  NUMSAV   PIC X(6).
         02  ITMSAV   PIC XX.
           •
           •
     PROCEDURE DIVISION.
4    ENTRY 'DLITCBL'.
           •
           •
     MOVE CUSTNUM TO NUMSAV.
           •
           •
     * GET UNIQUE COMMAND
     *
     EXEC DLI
5         GET UNIQUE USING PCB(1)
          SEGMENT(STSCCST) WHERE(STQCCNO=NUMSAV) INTO(STSCCST)
          FIELDLENGTH(6) SEGLENGTH(106)
     END-EXEC.
     IF DIBSTAT NOT = '  ' GO TO LSTCUS.
     IF DIBSTAT NOT = 'GB' TO TO REALER.
           •
           •
     * GET NEXT COMMAND
     *
     EXEC DLI
6         GET NEXT USING PCB(1)
          SEGMENT(STSCCST) INTO(STSCCST) SEGLENGTH(31)
     END-EXEC.
     IF DIBSTAT NOT = '  ' GO TO REALER.
           •
           •
     MOVE NEWDATA TO STSCCST.
           •
           •
7    * REPLACE COMMAND
     *
     EXEC DLI
          REPLACE USING PCB(1)
          SEGMENT(STSCCST) FROM(STSCCST) SEGLENGTH(31)
     END-EXEC.
           •
           •
8    GOBACK.
     STOP RUN.
```

Figure 4-5. COBOL command examples

COBOL Batch Program Structure

The example in Figure 4-5 illustrates the use of the HLPI commands, using the sample data base. The following explanation relates to the reference numbers along the left side of the figure.

1. This is the control statement that signifies to the CICS/VS translator that HLPI commands are present in your host language application program.
2. A 01 working-storage entry defines the program segment I/O area. Separate I/O areas may be allocated for each segment type.
3. A 01 work-storage entry defines the field variables used in the WHERE option clauses to qualify the segment selection. This area can also be further defined.
4. This is the standard entry point in the procedure division of a COBOL 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.
5. This is a typical command to retrieve data from a data base using the WHERE option to qualify the segment selection. The WHERE variable (NUMSAV) should be set to a predefined value. Immediately following the command, a test should be made of the status-code field of the DIB to determine if the command executed successfully.
6. This is a typical command to retrieve data from a data base for a subsequent replace function.
7. This statement replaces data in the data base with data from a COBOL batch program. Prior to the issuing of a REPLACE command, a GET command must be executed and data changed in an I/O area.
8. The GOBACK statement causes the COBOL batch program to return control to DL/I.

Note: STOP RUN cannot be used in a COBOL program as an exit statement, because control would not be returned to DL/I to allow it to release its resources and close the data bases and log. This will probably result in lost data. However, it can be coded after GOBACK. This prevents the compiler from giving a warning message and automatically generating a STOP RUN.

```
1    *PROCESS XOPTS(DLI)...;
     DLIPLIJB: PROCEDURE OPTIONS(MAIN);
          •
          •
     DCL 1 ORDSAV         CHAR (12) DEFINED FILL_ORDSAV POSITION (1);
     DCL 1 STSCCST,
           3 ......;
2    DCL 1 STSCLOC,
           3 ......;
     DCL 1 STPCORD,
           3 ......;
     DCL 1 STLCITM,
           3 ......;
     DCL 1 SAVE-AREAS,
3         3 LOCSAV        CHAR (6),
          3 NUMSAV        CHAR (6),
          3 ITMSAV        CHAR (2),
     DCL 1 FILL_ORDSAV,
          3 .....;
          •
          •
     EXEC DLI                              /* GET NEXT IN PARENT COMMAND    */
4         GET NEXT IN PARENT USING PCB(1)
          SEGMENT(STSCLOC) INTO(STSCLOC);
     IF DIBSTAT ¬= '  ' THEN               /* WAS THERE A DL/I ERROR?       */
          CALL REALER;                     /* CALL ERROR ROUTINE            */
     ELSE                                  /* NO. CONTINUE                  */
          •
          •
     EXEC DLI
          DELETE USING PCB(1)
          SEGMENT(STSCLOC) FROM(STSCLOC);
     IF DIBSTAT ¬= '  ' THEN               /* WAS THERE A DL/I ERROR?       */
        CALL REALER;                       /* YES. CALL ERROR ROUTINE       */
     ELSE                                  /* NO. CONTINUE                  */
          •
          •
     STLCITM=NEWDATA;                      /* MOVE SEGMENT DATA TO I/O AREA*/
5    EXEC DLI                              /* INSERT COMMAND                */
          INSERT USING PCB(1)
          SEGMENT(STLCITM) FROM(STLCITM);
     IF DIBSTAT ¬= '  ' THEN               /* WAS THERE A DL/I ERROR?       */
        CALL REALER;                       /* YES. CALL ERROR ROUTINE       */
     ELSE                                  /* NO. CONTINUE                  */
          •
          •
          •
6    RETURN;
     END DLIPLIJB;
```

Figure 4-6. PL/I command examples

PL/I Batch Program Structure

The example in Figure 4-6 illustrates the use of the HLPI commands, using the sample data base, to delete obsolete or incorrect data and to insert the new data into a master data base. The following explanation relates to the reference numbers along the left side of the figure.

1. This is the control statement that signifies to the CICS/VS EXEC translator that HLPI commands are present in your application program.
2. The segment I/O areas are defined.
3. The field variables are defined for use in the WHERE option clauses to qualify the segment selection. These fields can be further defined.
4. This is a typical command to retrieve data from a data base. This command is used here for a subsequent delete function. Immediately following the command, a test should be made of the status-code field in the DIB to determine if the command was successful.
5. This command is used to insert data into the data base from your PL/I application program. Before an INSERT command can be issued, segment data must be moved into an I/O area.
6. This RETURN statement causes the batch application program to return control to DL/I.

Data Base Positioning

To satisfy a request, DL/I relies on the established position in the data base as set by the previous command against the PCB.

The data base position is the knowledge by DL/I of the last segment retrieved and all segments above it in the hierarchy. This position is maintained by DL/I for each PCB. When an application program has multiple PCBs for a single data base, these positions are maintained independently.

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 direct access processing, this is not necessarily the root segment with the lowest key value.

The following basic rules apply for data base positioning after an HLPI command:

- A Replace command does not change current position in the data base.
- Data base position after a successful INSERT command is immediately after the inserted segment.
- Data base position after the return of an II status code is immediately prior to the duplicate segment. This positioning allows the duplicate segment to be retrieved with a GET NEXT command.
- Data base position after a successful DELETE command is immediately after all dependents of the deleted segment.
- Data base position is unchanged by an unsuccessful DELETE command.
- After an unsuccessful (partial) retrieve command, the DIB (DL/I Interface Block) reflects the lowest level segment which satisfied the request. (The DIB is described later in this chapter.)

In considering the current position pointer in the data base, remember that DL/I must first establish a starting position to be used in satisfying the request. This starting position is the current position in the data base for GET NEXT commands.

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 command is the start of the data base.
- If a GET UNIQUE command 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 command. Two exceptions are:
  1. When the end of the data base is reached, and
  2. For direct processing, where it is the next segment in physical sequence.

You can always establish your data base positioning with a GET UNIQUE command, specifying all the segment key values in the hierarchical path. It is recommended that you use a GET UNIQUE command after each "not found" condition.

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 translate, compile, and link-edit the program, as shown below, are used. The use of "PL/I-SORT" programs, using the sort program product, is not affected 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.

On Mixed Use of Interfaces

Two DL/I interfaces are available for use in writing application programs: the DL/I HLPI and the DL/I CALL interface. Either can be used in an application program written for the batch, MPS batch, or CICS/VS online environment. Also, within the online environment, either interface can be used with statements in the CICS/VS macro level or the CICS/VS command level interface. However, statements in the two DL/I interfaces must not appear in the same program; neither can both interfaces be used in different programs that exchange control during the life of a given task. This means that if control is passed to programs other than the one that initially acquired the PSB, these programs should access the DL/I data base using the same interface as the original program.

On Host Language Use and Features

Certain features of the PL/I Optimizer and ANS COBOL languages cannot be used in application programs used with CICS/VS. These restrictions and other techniques are described in the discussion on programming techniques and restrictions in the *CICS/VS Application Programmer's Reference Manual (Command Level).*

On Use of Reserved Keywords and Labels

Two pairs of keywords are reserved for use as triggers by the translator and can be used only within HLPI commands. They are:

- EXECUTE DLI
- EXEC DLI

The translator generates labels for the DIB, DL/I default values, and its own internal variables. To avoid creating duplicately defined symbols, you must not define the following:

- Labels beginning with the characters "DLZ"
- Labels beginning with the characters "DIB"
- Labels beginning with the characters "DFH" in COBOL programs
- Labels beginning with the characters "DFH" in PL/I programs

The *CICS/VS Application Programmer's Reference Manual (Command Level)* lists CICS/VS reserved labels.

CICS/VS Translator

The HLPI commands that you coded in your application program must be translated into calls to DL/I in the host language of your application program.

The translation is done by the CICS/VS EXEC translator. The translator is executed as a separate job step before compilation of your program. CICS/VS supplies cataloged procedures you can use to set up the translation step. They are described in the *CICS/DOS/VS Installation and Operations Guide.*

The translator generates an initialization call at the beginning of each external procedure. It also generates, for each external procedure, the labels for the DIB fields and other fields that may be needed by statements subsequently generated.

The translator searches your code for triggers corresponding to the XOPTS options specified on the PROCESS (PL/I) or CBL (COBOL) statements at the beginning of your program. (See *CICS/VS Application Programmer's Reference Manual (Command Level)* for details on translator options.)

Possible triggers are:

```
{EXECUTE CICS}        {EXECUTE DLI}
{EXEC    CICS}        {EXEC    DLI}
```

Either type of command, or both, may be present in an online program.

When the translator recognizes an HLPI command, it scans the command looking for proper syntax. If an error is found, a message is printed. The translator proceeds to generate one or more CALL statements to DL/I. The translator replaces the command you coded in your program with the calls and parameter lists.

Translate, Compile, and Link-Edit

Job Control

The output of the translator must be compiled by the appropriate host language compiler and link-edited into a core image library for execution as a separate job. DL/I application programs cannot be executed in a compile-link-go environment, since they run as a subprogram of the DL/I initialization program.

The appropriate DL/I or CICS/VS interface module must be link-edited with your program. The module to be included is determined by the host language and the operating environment.

Batch and MPS batch programs must also have an entry statement included in the input to the linkage editor.

The following examples illustrate job control statements needed for various combinations of environment and host language. Further examples and use of translator cataloged procedures appear in the *CICS/DOS/VS Installation and Operations Guide*.

For COBOL

BATCH AND MPS BATCH

```
// JOB COBSAMPL
// DLBL IJSYSPH,'COBOL TRANSLATION',yy/ddd
// EXTENT SYSPCH,balance of extent information
ASSGN SYSPCH,DISK,VOL=volid,SHR
// EXEC DFHECP1$,SIZE=...
 CBL LIB,XOPTS(DLI)
       •
       •
SOURCE DECK
       •
       •
/*
CLOSE SYSPCH,PUNCH
// DLBL IJSYSIN,'COBOL TRANSLATION',yy/ddd
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=volid,SHR
// OPTION SYM,ERRS,NODECK,CATAL
   PHASE COBSAMPL,*
   INCLUDE DLZLICBL
   INCLUDE DLZBPJRA
// EXEC FCOBOL,SIZE=...
   ENTRY CBLCALLA
// EXEC LINKEDT
/&
// JOB RESET
CLOSE SYSIPT,00C
/&
```

For PL/I

BATCH AND MPS BATCH

```
// JOB PLISAMPL
// DLBL IJSYSPH,'PL/I TRANSLATION',yy/ddd
// EXTENT SYSPCH,balance of extent information
ASSGN SYSPCH,DISK,VOL=volid,SHR
// EXEC DFHEPP1$,SIZE=...
*PROCESS INCLUDE,XOPTS(DLI);
       •
       •
SOURCE DECK
       •
       •
/*
CLOSE SYSPCH,PUNCH
// DLBL IJSYSIN,'PL/I TRANSLATION',yy/ddd
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=volid,SHR
// OPTION CATAL
   PHASE PLISAMPL,*
// EXEC PLIOPT,SIZE=...
   INCLUDE DLZLIPLI
   INCLUDE IBMBPJRA
   ENTRY DLZLIPLI
// EXEC LINKEDT
/&
// JOB RESET
CLOSE SYSIPT,00C
/&
```

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:

```
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}
```

Parameters can be entered from SYSIPT or SYSLOG. However, continuation statements, if required, can only be entered from SYSIPT. Continuation statements are not permitted from SYSLOG.

Continuation is indicated by a non-blank character in column 72 of the statement being continued. The parameter statement can be stopped in or before column 71 and be continued in a continuation statement.

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 *DL/I DOS/VS Data Base Administration.*

HDBFR
: describes one DL/I subpool (See *DL/I DOS/VS Data Base Administration.*)

- 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 exceeding 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 *DL/I DOS/VS Data Base Administration.*

- 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 causes 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 causes 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 causes an abnormal termination.
- dbdname3 is the name of the HISAM or SHISAM 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 *DL/I DOS/VS Data Base Administration.*

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 to the DL/I system log file.
	= 1	DL/I system log function inactive
Bit 7	= 0	Set exit (STXIT) linkage to DL/I for abnormal task termination.
	= 1	STXIT inactive

UPSI byte settings in the online environment have different meanings than those for batch. See Chapter 5 of this manual. 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,cuu
// TLBL    LOGOUT
```

If disk logging is used, a DLBL statement as shown below is required. The log file must have been previously defined with a DEFINE command because it is a VSAM file.

```
// DLBL {DSKLOG1},'cluster-name',,VSAM
       {DSKLOG2}
```

The execution job stream must contain DLBL or TLBL statements that define the data base(s) to be processed. When initially loading a data base, additional DLBL 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 *VSE/Advanced Functions System Control Statements* 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 4-7).

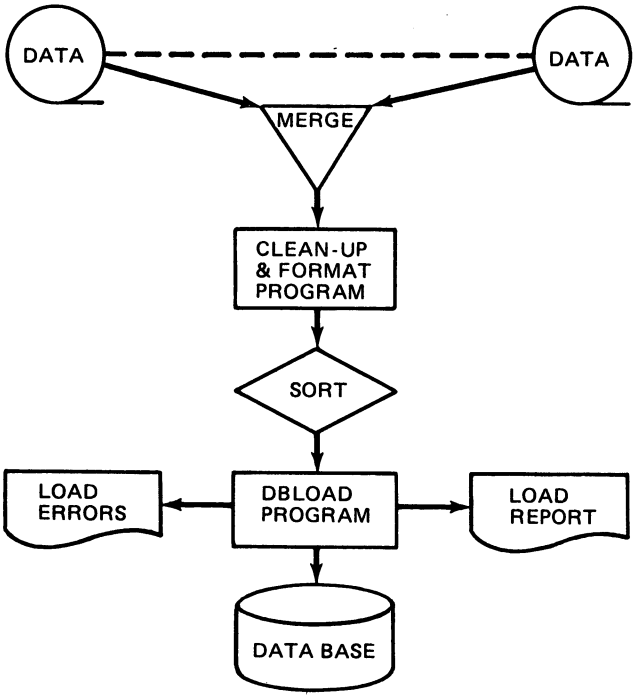


Figure 4-7. Basic Data Base Load Process

Loading Data Bases With Logical Relationships

DL/I provides a set of utility programs to establish the logical relationships during initial load of data bases with logical relationships. 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 automatically creates 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 7, "Data Base Reorganization/Load Processing".

Sample Data Base Load Program

The sample data base load programs, DLZCBL10 (COBOL) and DLZPLI10 (PL/I), are used to load the sample data bases. This sample is provided as a guide in writing your own data base load program(s).

Loading an HD Indexed Data Base

For initial loading of an HD indexed data base, you must specify PROCOPT=LS in the PCB. Data base records must be inserted in ascending root key sequence. Dependents of the root segment must be inserted after it, in hierarchical sequence.

If you need to sort dependent segments into hierarchical sequence by segment type, you have to write a program to prepare your input data for the sort operation by

adding a sort control field to each segment. The sort control field defines the hierarchical path to the particular segment and the sequencing within the path. The sort control field (Figure 4-8) is made up of:

- The root segment key field.
- The key fields of the segments in the hierarchical path between the root segment and this segment (one segment at each hierarchical level).
- The one-byte segment codes of those same segments.

When loading an HD indexed data base, DL/I also loads the primary index data base automatically. As each root segment is stored, DL/I generates the index segment for it and stores the index segment in the index data base.

The index data base is a KSDS. Index segments created during and after initial loading are placed in this data set.

The data stored in an HD indexed data base is stored in the ESDS. DL/I uses the HD space search algorithm to locate the most suitable space available for inserting each segment.

Loading an HD Randomized Data Base

When initially loading an HD randomized 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 passing each root key through the randomizing module for address conversion, and then sorting on the generated address plus the root key value.

DL/I DOS/VS Buffer Pool Characteristics Report

DL/I prints 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, 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 suppressed by assigning SYSLST to IGN.

ROOT KEY	LEVEL 2 SEGMENT CODE	LEVEL 2 KEY	LEVEL 3 SEGMENT CODE	LEVEL 3 KEY	etc.

Notes:

1. The segment code is one byte containing a binary number from 1 to 255 that is used by DL/I, instead of a segment name, to identify a segment type. The values are assigned to the segment types in ascending sequence, starting with the root segment type and continuing through all dependent segment types, following the hierarchical sequence you defined in the DBD.
2. For every level, the key field length should be equal to the length of the longest segment key field on that level. Keys shorter than that should be left-adjusted and padded on the right with characters having a low value in the sort sequence.
3. If no sequence field has been defined in the segment, provide one. Its value could be a simple dependent-segment counter provided by a user-written "clean-up and format" program. Segments on the lowest level need not have a key field if no sequence field was defined; however, their sequence below their parent might be different after the sort.

Figure 4-8. Control Field for Sorting Segments into Hierarchical Sequence

Processing With Logical Relationships

Generally, there is no difference between the processing of physical data bases and logical data bases; all command functions are available for both. Some considerations do apply when accessing a logical child or a concatenated segment. For a definition of these terms see "DL/I Logical Relationships" in Chapter 2.

Accessing a Logical Child in a Physical DBD

When accessing a logical child in a physical DBD, 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 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 typically happens when you forget the LPCK in front of the LCHILD.

Note: In general, don't use physical data bases when processing logical relationships.

Accessing Segments in a Logical DBD

The considerations that apply for each command function when accessing segments in logical DBDs are 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 *DL/I DOS/VS Data Base Administration.*

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 always maintains 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 sets when processing the main data base. So, the label information statements for the index data sets must be supplied in the job control statements of every job, which could change the secondary index.

Accessing Segments Via a Secondary Index

Retrieving Segments

The same commands are used as before. However, the index search field, defined by the REF parameter of an ACCESS statement in the DBD is used in the WHERE option for the GET UNIQUE of the root segment.

An example of a GET UNIQUE command for an INVENTORY ITEM using the secondary processing sequence is shown below. After the successful completion of this command, the DIB and IOAREA look the same as after the basic GET UNIQUE command.

```
EXEC DLI
     GET UNIQUE USING PCB(1)
     SEGMENT(STPIITM) INTO(IOAREA) SEGLENGTH(56)
     WHERE(STXININ=VALININ) FIELDLENGTH(6)
END-EXEC.
```

When using the secondary processing sequence, consecutive GET NEXT commands 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, use two PCBs.

Replacing Segments

To replace segments in the indexed data set, a combination of GET and REPLACE commands can be used as before. No sequence fields or index search fields may be changed while the secondary index is used as the processing sequence. However, the search fields can be changed if the access path is one other than the secondary index. If an index search field is changed, DL/I automatically updates the index pointer segment with a DELETE old and INSERT new pointer segment.

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, 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 INSERT command functions 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 set is created with the DL/I reorganization utilities. No application program is required for this creation. Chapter 7 covers this.

Chapter 5: Online and MPS Considerations

About This Chapter

This chapter contains the DL/I considerations for online programs. The parts include:

- Multiple partition support and the differences between batch, MPS, and online DL/I.
- CICS/VS system generation and the CICS/VS tables requiring specific entries for DL/I MPS support within CICS/VS.
- Application Control Table (ACT) functions: (1) providing DL/I with environmental information and, (2) defining the valid application program and PSB combinations for the online system.
- CICS/VS-DL/I table example of the parameters required in the various CICS/VS and DL/I tables to support DL/I and MPS.
- Online initialization with the UPSI byte settings.
- Formats of the HLPI SCHEDULE and TERMINATE commands and their use in COBOL and PL/I examples.
- Minimizing online data base contention through program isolation.
- The parameters available in the CICS/VS and DL/I system that affect performance by controlling the number of active tasks.

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/VSE. Special transactions in the CICS/VS partition receive these requests, issue the appropriate DL/I requests 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 updating during this time. Without MPS, the data bases would have to be closed in the online system while the batch programs are run.

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, command formats, and status codes.

Batch	Online	MPS
The application program is a subprogram to DL/I.	The application programs are CICS/VS tasks, utilizing the common DL/I code.	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.	The online program must explicitly issue a scheduling call for a PSB.	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.	Code must be quasi-reentrant (unless RELOAD=YES is specified in the CICS/VS PPT).	Code need not be reentrant.

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 with 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 CICS/VS dynamic transaction backout.

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). It is recommended that this log be assigned to the CICS/VS system journal. However, it may be 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.

DL/I also supports a restart facility for MPS batch users. MPS Restart uses the VSE checkpoint/restart facility in conjunction with the DL/I CHECKPOINT command to provide restart capability. Information on using the MPS Restart facility is provided in Chapter 8.

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; that is, 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; that is, not defined in the CICS/VS partition.

VSAM Data Set Share Options

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.

VSAM share option 2 is used for data sharing of data bases between DL/I subsystems in one host or across host systems. The data sets for the data bases the user wants to be shared must be defined to VSAM with share option 2. (See *DL/I DOS/VS Data Base Administration* for information on data sharing.)

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. Since 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 commands.

CICS/VS System Generation

There are no parameters for CICS/VS system generation specifically for DL/I MPS. The parameter DLI=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 some 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,DTB=YES,                  X
        TRANSID=CSDB,TWASIZE=488
DFHPCT  TYPE=ENTRY,PROGRAM=DLZBPC00,DTB=YES,                  X
        TRANSID=CSDC,TWASIZE=256
DFHPCT  TYPE=ENTRY,PROGRAM=DLZMSTP0,                          X
        TRANSID=CSDD
DFHPCT  TYPE=ENTRY,PROGRAM=DLZSTTL,                           X
        TRANSID=CSDE
*DFHPCT TYPE=ENTRY,PROGRAM=DLZMPUR0,                          X
        TRANSID=CSDP
```

* Needed for MPS Restart

These transactions are used as follows:

CSDA	Start MPS operation
CSDB	Master Partition Controller
CSDC	Batch Partition Controller
CSDD	Stop MPS Operation
CSDE	Run and Buffer Statistics
*CSDP	Purge Temporary Storage

* Needed for *MPS Restart

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, CSDD, CSDE, and CSDP 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 Partition Controller program ignores the request; however, no message is returned to the terminal.

Since the CSDA, CSDD, CSDE, and CSDP 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 program isolation 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.

If MPS Restart is used, the CSDB and CSDC transactions must be marked for dynamic transaction backout (DTB=YES).

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, or PL/I).

Several PPTs can be assembled and identified by suffix characters.

The programs referenced by the DL/I MPS transactions must be defined in the PPT. They are coded as follows:

```
DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTR0,RES=PGOUT
DFHPPT TYPE=ENTRY,PROGRAM=DLZMPC00,RES=YES
DFHPPT TYPE=ENTRY,PROGRAM=DLZBPC00,RES=YES
DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTP0,RES=PGOUT
DFHPPT TYPE=ENTRY,PROGRAM=DLZSTTL
*DFHPPT TYPE=ENTRY,PROGRAM=DLZMPUR0,RES=PGOUT
```

* Needed for MPS Restart.

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, DLZMSTP0, and DLZMPUR0 are low usage, they must be specified as RES=PGOUT. DLZMPC00 and DLZBPC00 must be specified as RES=YES. 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. Because 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 DLI=YES or DLI=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.

To use the MPS Restart facility, full support for emergency restarts and warm starts must be provided in your CICS/VS system. This includes specifying START=AUTO in the SIT.

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.

TST (Temporary Storage Table)

To use the MPS Restart facility, the temporary storage queue name, DLZTSQ00, must be specified as TYPE=RECOVERY in the TST. This ensures that the correct checkpoint ID will be available for verification when restarting MPS batch jobs in the event of a failure.

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 ensure that the program issuing the scheduling command 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).

How to Create an Online Nucleus Interactively . . .

The DL/I online nucleus creation and generation procedure described in this chapter can also be done interactively on a 3270-type terminal by using IMF (Interactive Macro Facility).

IMF provides the same options that are available through the conventional method of coding DLZACT macros. Through its easy-to-use interactive dialogue, however, the need for a detailed knowledge of these macros is reduced.

For overview information about IMF, see "Using the Interactive Facilities" in Chapter 1.

For more detailed IMF introductory and procedural information, see *DL/I DOS/VS Interactive Resource Definition and Utilities.*

Establishing the Control Section for the DL/I Application Control Table

The control section into which the ACT is assembled is established with 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 two-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 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 determined 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 DOS/VS Data Base Administration.* 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 *DL/I DOS/VS Data Base Administration.*

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 *DL/I DOS/VS Data Base Administration.*)

Describing the Application Program Relationship to DL/I Data Bases

The authorization for a CICS/VS application program to schedule a DL/I data base is granted through the DLZACT TYPE=PROGRAM macro instruction.

A maximum of 4095 DLZACT TYPE=PROGRAM statements 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.

A DLZACT TYPE=PROGRAM statement must be provided for every CICS/VS application program allowed to issue a PCB call or SCHEDULE command. This includes mirror programs such as the MPS mirror program (DLZBPC00) and the intersystem communication (ISC) mirror program (DFHMIR).

```
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 MPS batch 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 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 suffi-

cient 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 inter 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. Inter communication support is invoked when the DL/I program request handler detects a remote PSB during a scheduling command.

The system programmer, when generating a DL/I online nucleus, can optionally specify the location of PSBs through the DLZACT TYPE=RPSB macro instruction. (See *DL/I DOS/VS Data Base Administration* for details.)

The format of this statement is:

```
DLZACT TYPE=RPSB,
       PSB=psbname,
       [LNAME=localname,]
       RNAME=remotename,
       SYSID=systemid,
       [,LANG=pli]
```

where:

PSB
: specifies the name of a PSB specified on any previous DLZACT TYPE=PROGRAM statement. It must not be the same name specified by either LNAME or RNAME.

LNAME
: specifies the name of the PSB in the local system that is to be used with the PSB in the remote system to produce a remote PSB with a local component.

RNAME
: specifies the name given to the PSB in the remote system—remotename cannot be defined as an RPSB with a local component in the remote system.

SYSID
: specifies the four-character identifier of the CICS/VS system to which the remote PSB and associated data bases are assigned.

LANG
: specifies that the remote PSB is to be used by an MPS application program written in PL/I (pli may be coded as PL/I, PLI, PL/1, or PL1).

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 SHISAM 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 SHISAM 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
```

Job Control Statements for Creating the Online Nucleus

Online nucleus generation is run as a standard job and requires the following job control statements:

```
// JOB      NUCGEN
// OPTION   CATAL
// EXEC     ASSEMBLY
             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
/&
```

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

An Assembler language listing of the assembled online nucleus is provided.

Load Module

After the online nucleus generation is assembled, it must be link-edited and cataloged into a 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 because 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=488,   X
          CLASS=LONG,DTB=YES
 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
10. DFHPCT TYPE=ENTRY,PROGRAM=DLZMPUR0,TRANSID=CSDP,               X
          CLASS=LONG
                 •
                 •
    DFHPCT TYPE=FINAL
11. DFHPPT TYPE=INITIAL,SUFFIX=01
                 •
                 •
12. DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTR0,RES=PGOUT
13. DFHPPT TYPE=ENTRY,PROGRAM=DLZMPC00,RES=YES
14. DFHPPT TYPE=ENTRY,PROGRAM=DLZBPC00,RES=YES
15. DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTP0,RES=PGOUT
```

```
16. DFHPPT TYPE=ENTRY,PROGRAM=DLZMPUR0,RES=PGOUT
17. DFHPPT TYPE=ENTRY,PROGRAM=DLZHLPI
18. DFHPPT TYPE=ENTRY,PROGRAM=DLZSTP00
19. DFHPPT TYPE=ENTRY,PROGRAM=DL1PROG
20. DFHPPT TYPE=ENTRY,PROGRAM=DFHPLTSU
21. DFHPPT TYPE=ENTRY,PROGRAM=DFHPLTSD
22. DFHPPT TYPE=ENTRY,PROGRAM=DLZFSDP0,RES=PGOUT
23. DFHPPT TYPE=ENTRY,PROGRAM=DFHDBP01
                •
                •
    DFHPPT TYPE=FINAL
24. DFHJCT TYPE=INITIAL,SUFFIX=01
25. DFHJCT TYPE=ENTRY,JFILEID=SYSTEM,BUFSIZE=1024,...
                •
                •
    DFHJCT TYPE=FINAL
26. DFHPLT TYPE=INITIAL,SUFFIX=SU
27. DFHPLT TYPE=ENTRY,PROGRAM=DLZMSTR0
                •
                •
    DFHPLT TYPE=FINAL
28. DFHPLT TYPE=INITIAL,SUFFIX=SD
29. DFHPLT TYPE=ENTRY,PROGRAM=DLZMSTP00
                •
                •
30. DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
31. DFHPLT TYPE=ENTRY,PROGRAM=DLZSTP00
                •
                •
    DFHPLT TYPE=FINAL
32. DFHALT TYPE=INITIAL,SUFFIX=01
                •
                •
33. DFHALT TYPE=ENTRY,PROGRAM=DLZMPC00,ADRSPCE=LOW,PAGEOUT=NO,     X
          ALIGN=...
34. DFHALT TYPE=ENTRY,PROGRAM=DL1PROG,ADRSPCE=LOW,...
                •
                •
35. DFHALT TYPE=ENTRY,ALIGN=YES,ADRSPCE=HIGH,                      X
          PROGRAM=(DLZMSTR0,DLZMSTP0,DLZSTP00,DLZFSDP0),           X
          PAGEOUT=YES
                •
                •
    DFHALT TYPE=FINAL
36. DLZACT TYPE=INITIAL,SUFFIX=01
37. DLZACT TYPE=CONFIG,PI=YES,...
38. DLZACT TYPE=PROGRAM,PGMNAME=DL1PROG,PSBNAME=(...)
39. DLZACT TYPE=PROGRAM,PGMNAME=DLZBPC00,PSBNAME=(...)
                •
                •
40. 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 of CICS/VS. 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. Note that dynamic transaction backout support is specified. This is required for the MPS Restart facility.
8. DL/I MPS Batch Partition Controller transaction. Note that dynamic transaction backout support is specified for the MPS Batch Partition Controller transaction. This causes the DL/I updates made by an MPS batch program to be backed out if the batch program abnormally terminates.
9. DL/I MPS Stop transaction.
10. DL/I MPS Purge Temporary Storage transaction (required by MPS Restart).
11. This defines the PPT with a suffix of "01" to match the SIT specification.
12. DL/I MPS Start program referred to by (6).
13. DL/I MPS Master Partition Controller program referred to by (7).
14. DL/I MPS Batch Partition Controller program referred to by (8). Note that this program is also defined in the DL/I ACT (39).
15. DL/I MPS Stop program referred to by (9).
16. DL/I MPS Purge Temporary Storage program referred to by (10).
17. This is required by EDF to process the HLPI commands.
18. DL/I System Termination Program.
19. An example of a DL/I application program PPT entry. Note that this program is also defined in the CICS/VS ALT (34) and the DL/I ACT (38).
20. Start-up PLT entry referenced in (1) and defined in (26).
21. Shut-down PLT entry referenced in (1) and defined in (28).
22. DL/I's formatted system dump program used during CICS/VS ABEND processing.
23. This defines the dynamic transaction backout program specified in (1).
24. This defines the JCT with a suffix of "01" to match the SIT specification.
25. Although a buffer size of 1024 bytes (default value) is defined here, the maximum size handled by DL/I logging is 32,767 bytes.
26. This defines the start-up PLT with a suffix of "SU" to match the SIT specification.
27. The MPS Start program is listed in the start-up PLT so that MPS operation will be initiated automatically during CICS/VS initialization.
28. This defines the shut-down PLT with a suffix of "SD" to match the SIT specification.
29. 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.

30. 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.
31. This is the required DL/I System Termination Program entry.
32. This defines the ALT with a suffix of "01" to match the SIT specification.
33. 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.
34. 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.
35. 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 (PGOUT=YES). They are aligned to allow a clean page-out, that is, the page-out will not carry code for other programs with it.
36. This defines the DL/I ACT with a suffix of "01" to match the SIT specification.
37. This specifies the DL/I environmental factors. Note that PI=YES is specified. This is also the default action.
38. This is an example of a DL/I online application program entry. Note that this program is also defined in the CICS/VS PPT (19).
39. 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 (14). This is the only one of the four DL/I MPS programs that should be defined in the DL/I ACT.
40. This specifies the DL/I buffer pool characteristics.

CICS/VS Execution Diagnostic Facility (EDF)

You can use the CICS/VS Execution Diagnostic Facility (EDF) if your application program runs in the CICS/VS online environment. EDF is an interactive debug tool that allows you to see, in EXEC command format, what your program is doing at execution time. EDF should enhance your programming productivity for programs using DL/I HLPI and CICS/VS commands.

EDF:

- Displays each command or selected commands and options before the command is executed.
- Allows temporary modification of command arguments before execution.
- Displays each command or selected commands and options after the command is executed.
- Displays, and allows modification to, DIB fields and the programs' working storage (including I/O areas) on request.
- Allows redisplay of the last ten commands executed.

Note: If the host language of your program is COBOL, you should always specify SEGLENGTH for fixed length segments so that EDF will display the correct amount of data.

For more information on EDF, see the *CICS/VS Application Programmer's Reference Manual (Command Level).*

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
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 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 8 "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.

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/DOS/VS Installation and Operations Guide.*

Scheduling Command

Online application programs must schedule a PSB to notify DL/I that they want to use a data base. The format of the SCHEDULE command is:

```
EXECUTE DLI SCHEDULE PSB(name);
```

where "name" is a constant naming one of the PSBs available to your application program.

If a PSB was scheduled successfully after a SCHEDULE command is executed, a blank (b̄b̄) status code is placed in the DIBSTAT field of the DIB. If a status code other than a blank is issued, DL/I reports the code in a message and then terminates the application program. See "Status Codes" in Chapter 4 for a list of the status codes. (See Figure 5-1 and Figure 5-2 for examples of the SCHEDULE command.)

Note: DL/I data base services are available to CICS/VS application programs through the HLPI commands. The commands are the same for batch and online programs: EXECUTE DLI or EXEC DLI.

Terminating Command

When you no longer need a data base, while operating in an online environment, you should release (terminate) the PSB activated with the SCHEDULE command. This tells DL/I you have finished using the data base and the released PSB will be available for other applications that may be active. The format of the TERMINATE command is:

```
EXECUTE DLI TERMINATE;
```

If your program needs to use the data base again, you must reschedule the PSB with another SCHEDULE command at that time.

If the TERMINATE command was executed successfully, a blank (b̄b̄) status code is returned in the DIBSTAT field of the DIB. The status code TG is issued if a TERMINATE command was issued when a PSB had not been previously scheduled for the task. If a status code other than a blank (b̄b̄) or TG is issued, DL/I reports the code in a message and then terminate the application program. See "Status Codes" in Chapter 4 for a complete list of status codes. (See Figure 5-1 and Figure 5-2 for examples of the TERMINATE command.)

COBOL Online Program Structure

The example in Figure 5-1 illustrates the use of the HLPI SCHEDULE and TERMINATE commands, using the sample data base. A GET NEXT command is included to show the use of other HLPI commands that can be issued between the SCHEDULE and TERMINATE commands. The following explanations reference the numbers along the left side of the figure.

1. This is the control statement that signifies to the CICS/VS translator that HLPI and CICS/VS commands are present in your host language application program.
2. A 01 working-storage entry defines the program segment I/O area. Separate I/O areas may be allocated for each segment type.
3. A 01 working-storage entry defines the field variables used in the WHERE option clauses to qualify the segment selection. This area can also be further defined.

4. This is a typical SCHEDULE command to schedule the use of a PSB. This command notifies DL/I that it wants to use a data base.
5. This is a typical command to retrieve data from a data base to illustrate the use of an HLPI command between the execution of the SCHEDULE and TERMINATE commands. Immediately following the command, a test should be made of the status-code field of the DIB to determine if the command executed successfully.
6. This is a typical TERMINATE command that releases a PSB activated with the SCHEDULE command.
7. This is a CICS/VS command that returns control to DL/I.

```
1    CBL XOPTS(CICS,DLI)...
     ID DIVISION
     PROGRAM-ID. DLICOBLA.
          •
          •
     DATA DIVISION.
     WORKING-STORAGE DIVISION
     01  STSCCST.
         03......
2    01  STSCLOC.
         03......
     01  STPCORD.
         03......
     01  STLCITM.
         03......
     01  SAVE-AREAS.
         02  LOCSAV        PIC X(6).
3        02  ORDSAV.
             03.......
         02  NUMSAV        PIC X(6).
         02  ITMSAV        PIC XX.
               •
               •
     PROCEDURE DIVISION.
               •
               •
     EXEC DLI
4         SCHEDULE PSB(STBCUSR)
     END-EXEC.
               •
               •
     EXEC DLI
5         GET NEXT USING PCB(1)
          SEGMENT(STSCCST) INTO(STSCCST) SEGLENGTH(31)
     END-EXEC.
     IF DIBSTAT = '    ' GO TO LSTCUS
     IF DIBSTAT NOT = 'GB' GO TO REALER
               •
               •
     EXEC DLI
6         TERMINATE
     END-EXEC.
               •
               •
     EXEC DLI
          SCHEDULE PSB(STBCUSU)
     END-EXEC.
               •
               •
     EXEC DLI
          TERMINATE
     END-EXEC.
               •
               •
7    EXEC CICS RETURN END-EXEC.
```

Figure 5-1. COBOL SCHEDULE and TERMINATE Example

PL/I Online Program Structure

The example in Figure 5-2 illustrates the use of the HLPI commands, using the sample data base. A GET UNIQUE is included to illustrate the use of other HLPI commands between the SCHEDULE and TERMINATE commands. The following explanations reference the numbers along the left side of the figure.

1. This is the control statement that signifies to the CICS/VS translator that HLPI and CICS/VS commands are present in your application program.
2. The segment I/O areas are defined.
3. The field variables are defined for use in the WHERE option clauses to qualify the segment selection. This area can also be further defined.
4. This is a typical SCHEDULE command to schedule the use of a PSB. This command notifies DL/I that it wants to use a data base.
5. This is a typical command to retrieve data from a data base to illustrate the use of an HLPI command between the execution of the SCHEDULE and TERMINATE commands. Immediately following the command, a test should be made of the status-code field of the DIB to determine if the command executed successfully.
6. This is a typical TERMINATE command that releases a PSB activated with the SCHEDULE command.
7. This is a CICS/VS command that returns control to DL/I.

```
1    *PROCESS XOPTS(CICS,DLI)....;
     DLIPL1JB: PROCEDURE OPTIONS(MAIN);
               •
               •
     DCL 1 ORDSAV....;
     DCL 1 STSCCST,
           3......;
2    DCL 1 STSCLOC,
           3......;
     DCL 1 STPCORD,
           3......;
     DCL 1 STLCITM,
           3......;
     DCL 1 SAVE_AREAS,
           3 LOCSAV       CHAR (6),
3          3 NUMSAV       CHAR (6),
           3 ITMSAV       CHAR (2),
     DCL 1 FILL_ORDSAV,
           3......;
               •
               •
     EXEC DLI
4         SCHEDULE PSB(STBCUSR);                  /*SCHEDULE COMMAND         */
               •
               •
     EXEC DLI
5         GET UNIQUE USING PCB(1)                 /*GET UNIQUE COMMAND       */
          SEGMENT(STSCCST) INTO(STSCCST);
     IF DIBSTAT ¬= '  ' THEN                      /*WAS THERE A DL/I ERROR? */
          CALL REALER:                            /*YES. CALL ERROR ROUTINE */
               •                                  /*NO. CONTINUE            */
               •
               •
6    EXEC DLI
          TERMINATE;                              /*TERMINATE COMMAND        */
               •
               •
     EXEC DLI
          SCHEDULE PSB(STBCUSU);                  /*SCHEDULE COMMAND         */
               •
               •
     EXEC DLI
           TERMINATE;                             /*TERMINATE COMMAND        */
               •
               •
7    EXEC CICS RETURN;
     END DLIPL1JB;
```

Figure 5-2. PL/I SCHEDULE and TERMINATE Example

Terminating an Online Program

In the online environment, control is returned to DL/I by coding a CICS/VS command as shown here:

```
COBOL:    EXEC CICS RETURN END-EXEC.
PL/I:     EXEC CICS RETURN;
```

Compilation and Link-editing

Job Control

The following examples illustrate job control statements needed in an online environment and host language. Further examples and use of translator cataloged procedures can be found in the *CICS/DOS/VS Installation and Operations Guide.*

COBOL

```
// JOB COBSAMPL
// DLBL IJSYSPH,'COBOL TRANSLATION',yy/ddd
// EXTENT SYSPCH,balance of extent information
ASSGN SYSPCH,DISK,VOL=volid,SHR
// EXEC DFHECP1$,SIZE=...
 CBL LIB,XOPTS(CICS,DLI)
        •
        •
SOURCE DECK
        •
        •
/*
CLOSE SYSPCH,PUNCH
// DLBL IJSYSIN,'COBOL TRANSLATION',yy/ddd
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=volid,SHR
// OPTION SYM,ERRS,NODECK,CATAL
   PHASE COBSAMPL,*
   INCLUDE DFHECI
// EXEC FCOBOL,SIZE=...
// EXEC LINKEDT
/&
// JOB RESET
CLOSE SYSIPT,00C
/&
```

PL/I

```
// JOB PLISAMPL
// DLBL IJSYSPH,'PL/I TRANSLATION',yy/ddd
// EXTENT SYSPCH,balance of extent information
ASSGN SYSPCH,DISK,VOL=volid,SHR
// EXEC DFHEPP1$,SIZE=...
*PROCESS INCLUDE,XOPTS(CICS,DLI);
        •
        •
SOURCE DECK
        •
        •
/*
CLOSE SYSPCH,PUNCH
// DLBL IJSYSIN,'PL/I TRANSLATION',yy/ddd
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=volid,SHR
// OPTION CATAL
   PHASE PLISAMPL,*
   INCLUDE DFHPL1I
// EXEC PLIOPT,SIZE=...
// EXEC LINKEDT
/&
// JOB RESET
CLOSE SYSIPT,00C
/&
```

MPS (Multiple Partition Support) Considerations

When a scheduling conflict occurs, the online program is normally suspended. When such a conflict is with an MPS batch job, DL/I issues a status code to the online program. If the status code is other than a bb, DL/I reports the code in message DLZ0371 and then terminates the program.

Once an online task is scheduled, and if it must still wait for a resource owned by a batch MPS task, the MPS task will be informed on the next and all subsequent commands until a DL/I checkpoint is issued. This condition is flagged by setting the DIB flag byte (DIBFLAG) to X'FF'. (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.

Storage Considerations

MPS programs using Program Isolation with update intent for segments should issue frequent checkpoints to avoid using more storage than is necessary.

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

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

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 partitions only. If this is so, then any DLBL statements necessary for these data bases have to be placed in the job control 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 control 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. This includes the temporary storage data set, where CICS/VS dynamic logs for dynamic transaction backout reside.

Executing MPS Batch Programs

To execute an MPS batch 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 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. It may not be turned off as in the case of non-MPS batch execution.

No DLBL 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 job control statements for the CICS/VS partition.

The DL/I parameter information for MPS batch operation need only specify the program name and the PSB name. Any other parameter information, such as buffer pool options, are ignored because this is controlled in the CICS/VS partition.

The phase name on the EXEC statement must be "DLZMPI00". A size parameter is required. In the batch partition, the size parameter and partition size can be made smaller than those used previously. There are less DL/I control blocks, no buffer pools, no action modules, and less initialization/nucleus code.

Minimizing Online Data Base Contention

Because DL/I resource contention degrades performance, an attempt should be made to minimize potential contention 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 DL/I resource contention. They are:

- Selecting the proper PSB PROCOPT
- Scheduling a PSB for a short duration of time
- Selecting the proper scheduling mechanism.

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 ensure that intent is not propagated unnecessarily.

When specifying PROCOPT, using the following guidelines:

- Specify G whenever possible because there is no update intent with this processing option.
- Specify R instead of I and D whenever possible because 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, that is, GIRD, and a P or E processing option is also required.
- The use of PROCOPT=E should be well justified before it is used because it also prevents read-only tasks from running concurrently.

Note: Details on PROCOPT can be found in *DL/I DOS/VS Data Base Administration.*

Scheduling a PSB for a Short Duration

When an online DL/I transaction has a considerable amount of processing to do using an update PSB, it may be desirable to occasionally release the PSB (via a TERMINATE command). This allows other update or exclusive tasks to get a chance at the data base. This can be thought of as "time-sharing" the data base among several PSBs.

Because this technique involves scheduling an update PSB more than once within a single task execution, you must ensure this does not affect the logical consistency of the data base if a failure and subsequent backout occurs.

You should be aware that if the PSB is released, and you try to reschedule the PSB but a scheduling conflict occurs with an MPS task, your online HLPI program will be abnormally terminated. If this happens, DL/I returns a status code (TL) in the DIBSTAT field of the DIB. DL/I also reports the code in message DLZ037I and abnormally terminates your program.

Selecting a DL/I Scheduling Mechanism

Two DL/I task scheduling mechanisms are available to ensure the integrity of data bases concurrently accessed by multiple tasks in an online or MPS environment. They are: program isolation and intent scheduling.

Combined with CICS/VS dynamic transaction backout, program isolation provides data base integrity in an update environment. Intent scheduling also provides data base integrity. However, a higher level of data base contention is likely except when intent scheduling is used in a predominately read-only environment.

Program isolation generally allows a greater degree of concurrent data base access, 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 in an update environment. Program isolation is provided by default and should normally be used. However, intent scheduling might be a better choice in a system that has severe real storage constraints and only read-only tasks. Each of these facilities is discussed in the following sections.

Note: Details on program isolation and intent scheduling can be found in *DL/I DOS/VS Data Base Administration.*

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 DL/I resource contention in a DL/I-CICS/VS environment (including MPS). Program isolation is the recommended choice for DL/I online and MPS applications. Program isolation is not required in a batch environment since data bases cannot be simultaneously updated by more than one application in this environment. (See *DL/I DOS/VS Data Base Administration* for information on data sharing.)

Program isolation attempts to reduce the amount of data base contention by restricting contention in most cases, to actual segment occurrences rather than intent to reference a segment type.

Program isolation provides two major functions, contention management and deadlock avoidance.

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

Exclusive Intent:

Intent Scheduling	Program Isolation
Restrictions on task:	
Is not scheduled until all other tasks with any specified intent for this segment type have issued a TERMINATE command.	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 TERMINATE command.	Identical to intent scheduling.

Comparison: No differences. 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 TERMINATE command.	a. Is not scheduled until any other task with exclusive intent specified for this segment type has issued a TERMINATE command. b. Waits to read any segment updated by another task with update intent for that segment type until that task has issued a TERMINATE command. c. Waits to update any segment LOCKED by another task until that task has issued a TERMINATE command.
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 TERMINATE command.	a. Any other task with exclusive intent for this segment type wants to be scheduled until this task issues a TERMINATE command. b. Any other task with update intent for this segment type waits to update a segment LOCKED by this task until this task has issued a TERMINATE command. 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 TERMINATE command.

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 tasks has specified exclusive intent for this segment until that task has issued a TERMINATE command.	a. Identical to intent scheduling. b. Waits to read any segment updated by another scheduled task until that task has issued a TERMINATE command.
Restrictions on other tasks:	
Any other task with exclusive intent for this segment type waits to be scheduled until this task has issued a TERMINATE command.	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 (that is, MPS batch vs. MPS batch or online vs. online).

All CICS/VS transactions that modify DL/I data bases (including the MPS mirror transaction CSDC) must be defined for dynamic transaction backout in the CICS/VS Program Control Table. Failure to do so can cause loss of data base integrity.

Selection of DL/I Scheduling Mechanism: Program isolation is automatically selected during DL/I nucleus 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. Intent scheduling is selected by specifying PI=NO.

Operational Considerations: Three messages are issued by program isolation:

Message DLZ106I, indicating task termination due to an unresolvable confilict in data usage, identifies resource contention that is dependent on the current mix of tasks. The task may execute successfully if rerun. If not, evaluation of the mix of tasks and the resources they reference is required.

Message DLZ108I, indicating insufficient space, identifies a problem in getting more space from CICS/VS for queueing control blocks. When using program isolation, this contention could occur when a task(s) (possibly an MPS batch program) runs a long time without issuing a checkpoint or a CICS/VS sync point, thereby using 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.

Message DLZ113I indicates the status of program isolation during online initialization.

Programming Considerations

1. MPS batch programs with update intent for segments should issue frequent CHECKPOINT commands. This minimizes scheduling conflicts between programs since the segments are LOCKED and remain unavailable to other programs until your program issues a DL/I CHECKPOINT command. MPS Restart facilitates the use of frequent checkpointing by allowing users to restart their MPS batch programs from the last successful checkpoint taken before a failure.

   Note: A code is passed back in DIBFLAG of the DIB whenever another task is required to wait on a resource owned by the calling task. This could be used to trigger checkpoints.

2. If data is read that will be used in a data base update, and the PCB used references a different data base record before the update occurs, the LOCKED option should be specified to protect the data from modification by another concurrently executing task.
3. CICS/VS transactions that are not restartable must use PROCOPT=E on all segments to bypass program isolation and thus avoid the possibility of termination due to deadlock conditions.
4. Program isolation may cause an increased level of transaction backouts. These can 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 LOCKED option on the read to accomplish the same purpose.
8. Specifying processing option 'GO' in the PCB will avoid all program isolation 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 to understand 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 with two chains: the active chain and the suspend chain. The active task chain is maintained in task priority se-

quence 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 (because 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 the 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 SCHEDULE command and DL/I CMAXTSK had already been reached.
- The task issued a DL/I SCHEDULE command and DL/I determined it had intent conflict with another DL/I task.
- The task issued a DL/I data base request 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 nondispatchable 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 (that is, 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 ensure 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, for example, 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, which is not supported by HLPI, is available to data base administration to change the CMAXTSK value, (see *DL/I DOS/VS Data Base Administration* for additional information on CMAXTSK). Note that CMAXTSK can never exceed MAXTASK.

If a DL/I task issues a SCHEDULE command 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 SCHEDULE commands, 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; that is, 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: Processing Data Bases with RPG II

About This Chapter

This chapter contains considerations that apply to RPG II for data base processing. The topics include:

- The program structure and interface to DL/I, including the PCB-mask and the functions that request DL/I data base services.
- Application programming considerations and formats of the various RPG II statements.
- File definition and command codes that specify functional variations applicable to either the function or segment qualification.
- Data base positioning after a DL/I call is issued and the fundamental parts in the structure of an RPG II batch program.
- Job control statements and the RPG II translator that turns the DL/I requests, coded with the RQDLI commands, into DL/I calls.
- Online programming considerations for RPG II applications.

The information in this chapter supplements the information presented in other parts of this manual. RPG II application programs issue the RQDLI (request DL/I) command to access DL/I data bases. The DL/I calls produced by the RPG II translator are presented later in this chapter.

Program Structure and Interface to DL/I

Translation and Compilation

An RPG II program, which uses RQDLI commands must be translated before it is compiled. The RPG II program is compiled by the RPG II compiler and placed in the core image library, after it is link edited with the DL/I language interface module.

Interface Components

A DL/I batch application program executes in a manner similar to any other 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, which, in turn, invokes the operating system 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 6-1 presents an overview of DL/I and the application program during execution.

Before you execute an application program, a *program specifications 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 a core image library. The coding and generation of PSBs are described in Chapter 3 of this manual.

During initialization, DL/I loads the application program and its associated PSB from the library. The DL/I modules interpret and execute data base call requests issued by the program.

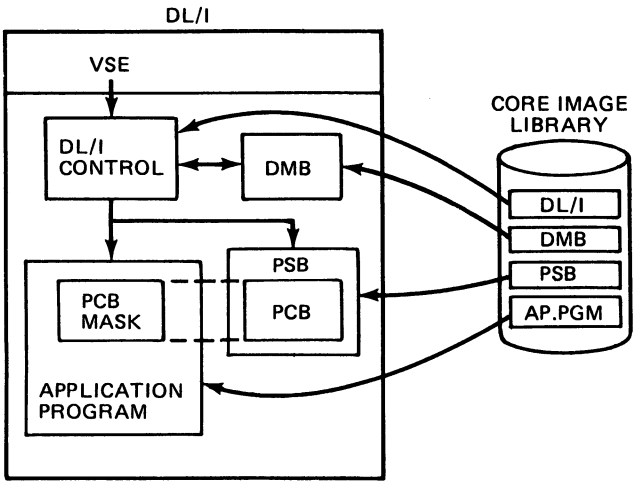


Figure 6-1. DL/I Interface with an Application Program

The application program interfaces with DL/I in the following ways:

- An entry statement specifying the PCBs used by the program.
- A PCB-mask that defines the information returned by DL/I to the application program.
- An I/O area for passing data segments to and from the data base.
- RQDLI commands specifying DL/I functions to be performed.
- A termination statement to return control to DL/I.

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 terminations 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 6-2 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 6-2 shows that when DOS/VSE 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 PSBs in the linkage section. Declaration of the PCB mask in the declaration portion of the application program need not be the same as the sequence in the entry statement.

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 automatically generates 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 de-

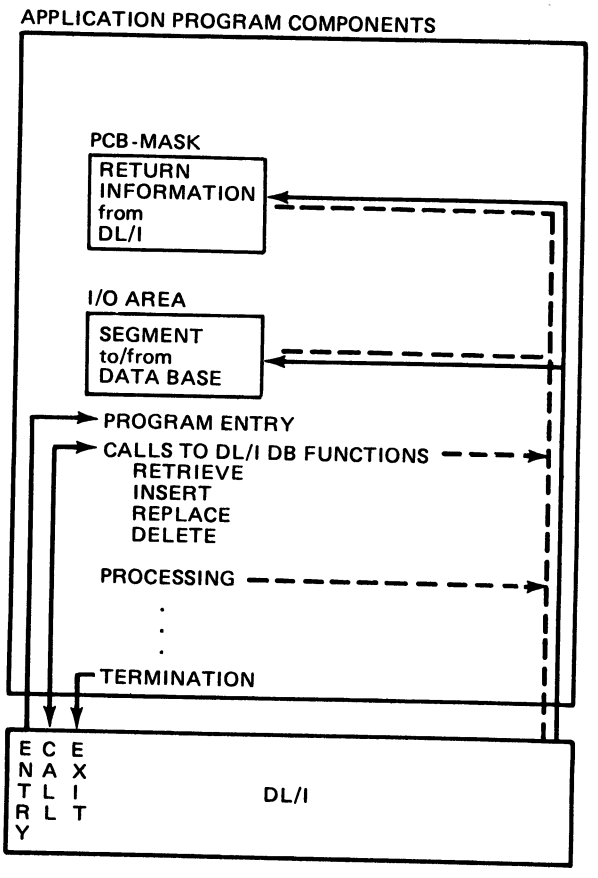


Figure 6-2. Structure of a Batch Application Program

scription of DB-files and PCB specification, see "Data Base File Definition" in this chapter.) The entry parameter list contains 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, the PCB names in the PARM statements must be in the same sequence as in the PSB generation for the program. The Translator does not check the contents of the list.

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 (see Figure 6-3, part 1).

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 generates this automatically. If you want to specify names of your choice, you may do so following the layout of the automatically generated PCB (see Figure 6-3, 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.

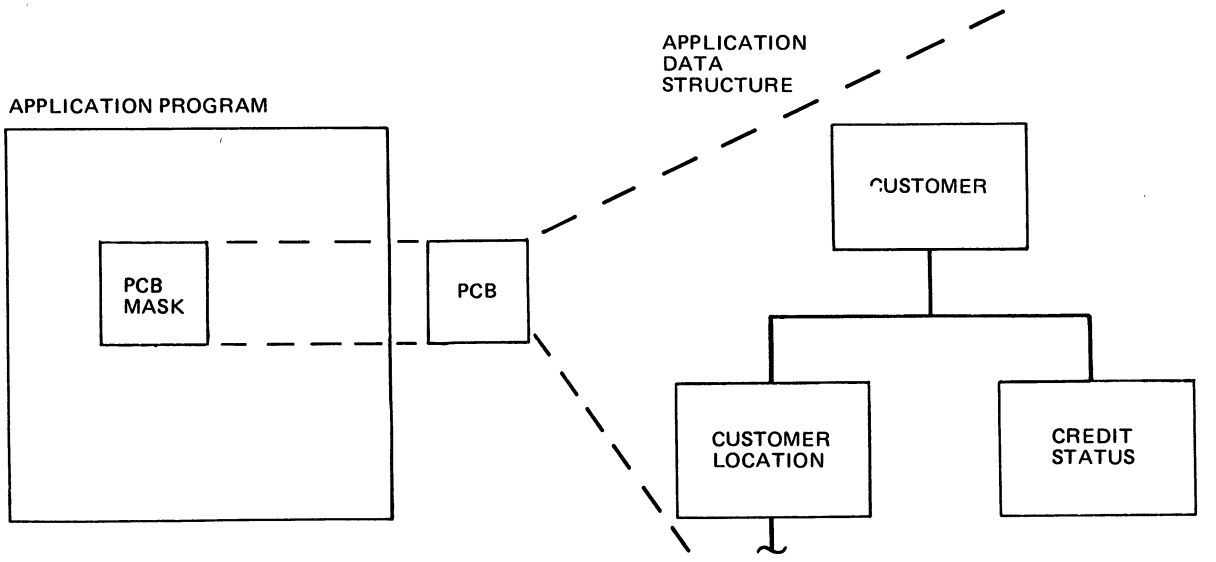


Figure 6-3. Application Program PCB Mask (Part 1 of 2)

RPG INPUT SPECIFICATIONS

GX21-9094-3 U/M050*
Printed in U.S.A.

IBM International Business Machines Corporation

Program		Punching Instruction	Graphic		Card Electro Number
Programmer	Date		Punch		

Page 1 2 of __ Program Identification 75 76 77 78 79 80

I

Line	Form Type	Filename	Sequence (O A, R N, D)	Number (1/N)	Option (O), U	Record Identifying Indicator, ** or DS	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Minus	Zero or Blank	
3 4 5	6	7 8 9 10 11 12 13	14 15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	P C B i j				D S																									1
0 2	I																				1	8		D B D N i j							2
0 3	I																				9	1 0		S E G L i j							3
0 4	I																				1 1	1 2		S T C D i j							4
0 5	I																				1 3	1 6		P R O C i j							5
0 6	I																			B	1 7	2 0		R E S R i j							6
0 7	I						RESBij overlays RESRij														1 7	2 0		R E S B i j							See Notes
0 8	I																				2 1	2 8		S E G N i j							7
0 9	I																			B	2 9	3 2 0		K E Y L i j							8
1 0	I						KEYBij overlays KEYLij														2 9	3 2		K E Y B i j							
1 1	I																			B	3 3	3 6 0		S S G N i j							9
1 2	I						SSGBij overlays SSGNij														3 3	3 6		S S G B i j							
1 3	I																				3 7	n n n		K E Y A i j							10
1 4	I																														
1 5	I																														
1 6	I																														
1 7	I																														
1 8	I																														
1 9	I																														
2 0	I																														
	I																														
	I																														
	I																														
	I																														
	I																														

nnn is the pcb-length + 36 (pcb-length is taken from pcb-keylength of the K-line of F-specs for DB-files).

*Number of sheets per pad may vary slightly.

Figure 6-3. Application Program PCB Mask (Part 2 of 2)

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. The name is not a field in the PCB. 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 number returned is that of the last segment that satisfied the search criteria along the path from the root 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 indication. (See *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces* for status code information.)
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 does 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 (four 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 GET NEXT or GET NEXT WITHIN PARENT 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.

Calls to DL/I

DL/I Application Programming 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 translates the RQDLI commands into RPG II call statements and parameter lists, and the DB-file specifications into File Description Specifications for SPECIAL files.

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.

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 in *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces.*

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 form 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 is issued even if the "record not found" condition is encountered. That means that in any case, the contents of the fields within the record are 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 uses indicator 13. The indicator should not be tested because, 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 filename specified with the RQDLI statement.

Segment Search Argument (SSA)

One segment search argument (SSA) can be provided for each segment accessed in a hierarchical path. The purpose of the SSA is to identify the segments 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 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	RO	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 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-vlaue.

Operator	Meaning
ƀ= or =ƀ	equal to
=>	greater than or equal to
=<	less than or equal to
ƀ> or >ƀ	greater than
ƀ< or <ƀ	less than
¬=	not equal to

Note: ƀ 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.

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 described above.)

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.

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 factor 1 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.

Note: When deleting segments within a packed key range, you must build your own SSA with the comparative field packed in the SSA.

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 expands 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.

Termination

At the end of processing of the application program, control must be returned to the DL/I control program. In RPG II, control is returned to DL/I by setting on the Last Record (SETON LR) indicator, specified in the calculation specifications.

DB (Data Base) File Definition

Each data base an application progrm wants to access may be defined in the File Description Specifications. 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.

Restriction: Test indicators (positions 65-70) for numeric fields can not be defined in the input specifications for a DL/I data base.

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 Specifications, with the names of the fields qualified by ij. The general format and the naming conventions can be seen in Figure 6-3 in "PCB Mask", in this chapter. If the K-lines for several DB-files define the sam 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.

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 eight-byte segment name. 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 be subsequently 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.

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 (GN) call. A get unique (GU) 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 insert (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/I 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/VS compatibility, the 'Q' command code must be followed by the character 'A'.

Note: By definition, the 'Q' command code 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/I 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

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 get next 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 an unsuccessful (partial) 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, and the level and key feedback length are set to 0.

In considering 'current position in the data base', remember that DL/I must first establish a starting position to be used in satisfying the call. This starting position is the current position in the data base for get next calls, and is a unique position normally established by the root SSA for get unique calls.

The following are clarifications of 'current position in the data base' for special situations:

- If no current position exists in the data base, then the assumed current position is the start of the data base.
- If the end of the data base is encountered, then the assumed current position to be used by the next call is the start of the data base.
- If a get unique call is unsatisfied at the root level, then the current position is such that the next segment retrieved is the first root segment with a key value higher than the one specified for the unsuccessful call.

You can always reestablish your data base positioning with a get unique 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.

RPG II Batch Program Structure

Figure 6-4 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 explanations reference the numbers along the right side of the figure.

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 select the names.
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 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 (DLZL1000 must be link-edited to the application program to provide a common interface to DL/I. When the application program is link edited, the operating system's automatic library look-up (AUTOLINK) feature retrieves the language interface module from a 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.

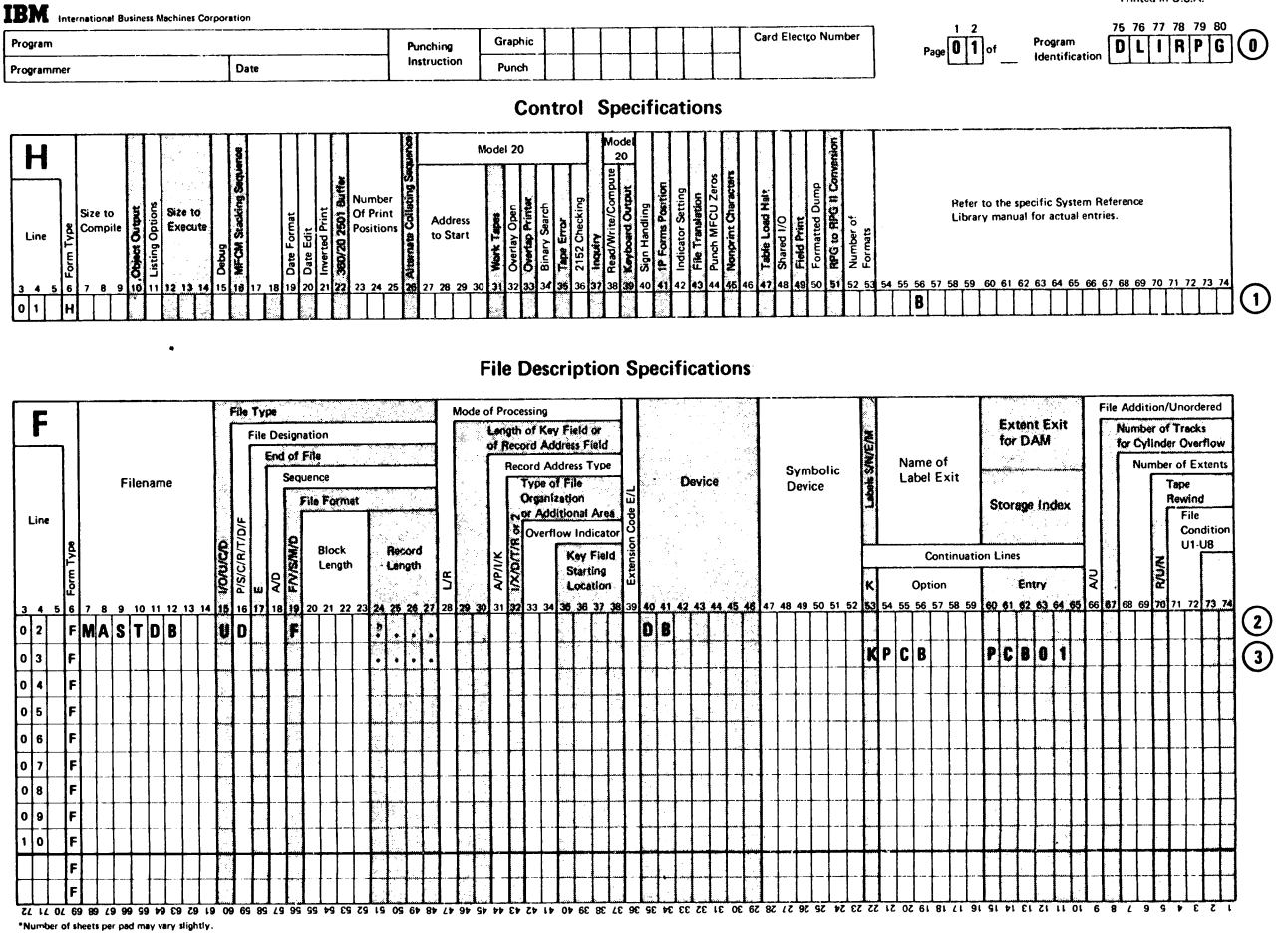


Figure 6-4. General RPG II Batch Program Structure (Part 1 of 6)

RPG INPUT SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9094-3 U/M050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number

Page 02 of __ | Program Identification DLIRPG

I

Line	Form Type	Filename	Sequence (O A, R N, D)	Number (1/N)	Option (O), U	Record Identifying Indicator, *, or DS	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank	
3 4 5	6	7 8 9 10 11 12 13	14 15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	MASTDB																													
0 2	I	* RECORD				LAYOT																									
0 3	I																														
0 4	I																														
0 5	I																														
0 6	I	DETAR				DS																									
0 7	I	*IOAREA FOR				DETAIL	DB																								
0 8	I																														
0 9	I																														
1 0	I	PCBDET				DS																									
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	20		FILLER							
1 6	I																														

(4) (5) (6)

Figure 6-4. General RPG II Batch Program Structure (Part 2 of 6)

RPG INPUT SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9094-3 U/M050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number

Page 03 of __ | Program Identification DLIRPG

I

Line	Form Type	Filename	Sequence (O A, R N, D)	Number (1/N)	Option (O), U	Record Identifying Indicator, *, or DS	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank	
3 4 5	6	7 8 9 10 11 12 13	14 15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I																				21	28		SEGNAM							
0 2	I																			B	29	32	0	KFBLEN							
0 3	I																			B	33	36	0	SENSSN							
0 4	I																				37	. . .		KEYFB							
0 5	I	QULSSA				DS																									
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																														

(7)

Figure 6-4. General RPG II Batch Program Structure (Part 3 of 6)

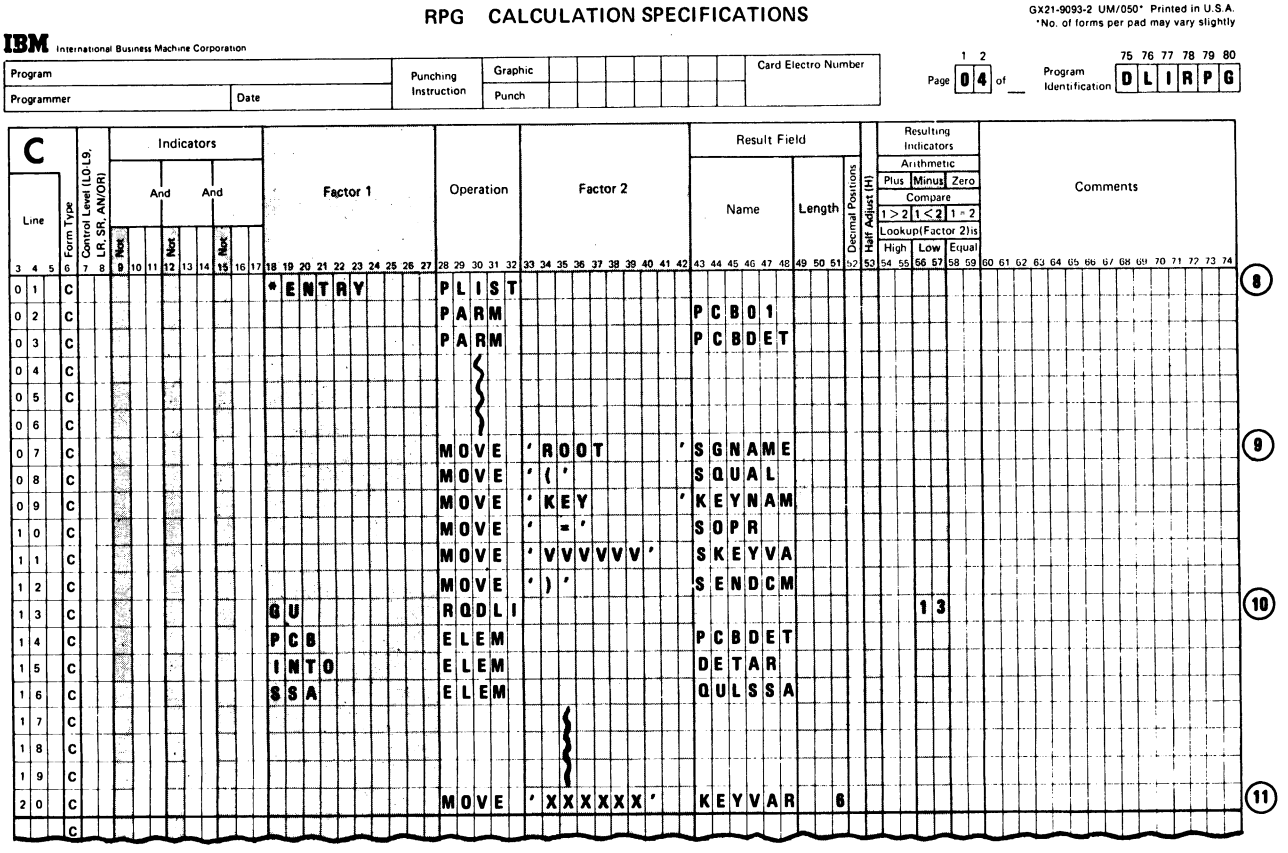


Figure 6-4. General RPG II Batch Program Structure (Part 4 of 6)

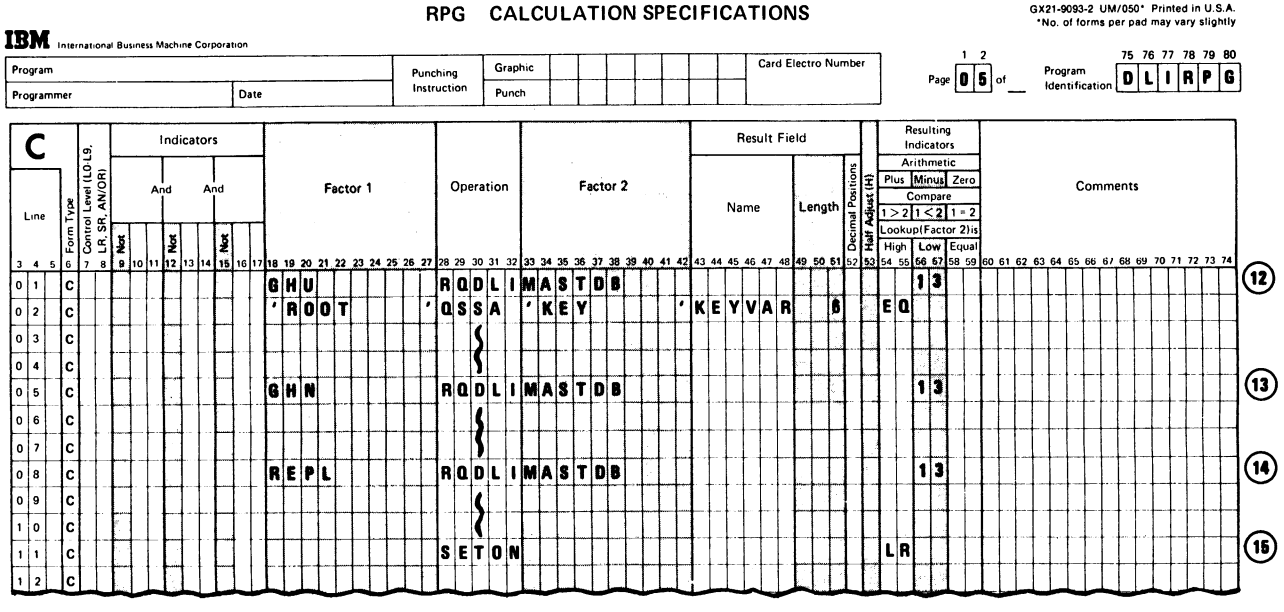


Figure 6-4. General RPG II Batch Program Structure (Part 5 of 6)

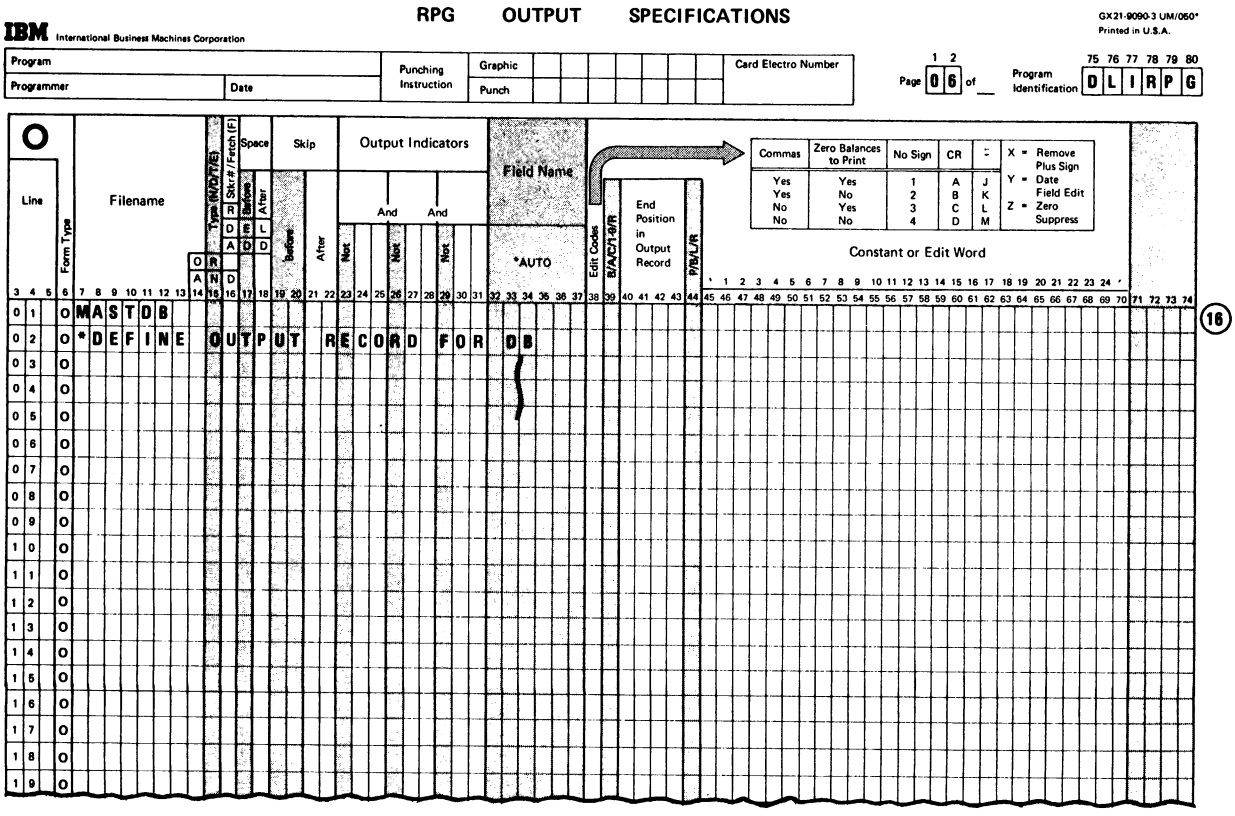


Figure 6-4. General RPG II Batch Program Structure (Part 6 of 6)

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 core image library and then executed as a separate job because they run as a subprogram of the DL/I initialization program.

Compile and Link Edit

The compilation of an RPG II program, which is going to run under DL/I, is a batch operation with the following steps:

RPG II program
(with B specified in Column 56 of Header Specifications)
↓
Translator
↓
RPG II Compiler
(Auto report optional)
↓
Application program

The RPG II program may optionally use Auto Report, to include RPG II program pieces from libraries, etc.

The function of the Translator is to accept as input a source program, written in PRG II, in which DL/I requests have been coded with RQDLI commands. The Translator produces as output an equivalent source program in which the DL/I requests have been translated 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. Its phase name is RPGIXLTR.

The Translator reads its input from SYSIPT, produces 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 Description 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 structures and fields together with MOVE or Z-ADD statements to build the proper SSAs from the USSA and QSSA statements, which are then used in the PARM statements 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 appears 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 environment, respectively.

If the leftmost two bits of the UPSI byte are 00 (either the standard setting, or set by // UPSI 00), the Translator directs its output to SYSPCH.

Example of job control statements for output on SYSPCH:

```
// JOB T
// EXEC RPGIXLTR (Translator)
    •
    •
/*
/&
```

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 then reads its input from SYS002 instead of SYSIPT.

Example of job control statements 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 then reads its input from SYS003 instead of SYSIPT.

Example of job control statements for output on SYS003:

```
// JOB TAR
// UPSI 10
// EXEC RPGIXLTR
    •
    •  Source to be translated
    •
/*
// EXEC RPGAUTO
/&
```

Online Programming Considerations

Before attempting to write a CICS/VS-DL/I program, you should be familiar with DL/I 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 application programs, the user has additional responsibilities when writing a CICS/VS-DL/I program.

The CICS/VS application programmer requests DL/I services by issuing the DL/I call just as would be done in a batch environment. 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 re-entrant 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. 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 the addresses of PCBs for use by the transaction by issuing a PCB RQDLI command. This is described in the following discussion, "Obtaining the Address of the PCB: The Scheduling Call."
2. Set the function code as in the batch environment.
3. Furnish the PCB address provided by the 'PCB' call previously issued.
4. 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 call in *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces.*

The format of the scheduling call is:

```
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 var-name (a variable psbname 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. The user must specify such a field in DFHDUM to establish the addressability of PCBs after a scheduling call.

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 UAPCBL contains the address of a PCB list. The PCB list consists of a series of four-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: 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 RQDLI commands requesting DL/I to access a data base, the application program must again schedule the PSB using a scheduling call. (This is not necessary if Program Isolation 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 RPG II program issues a call of the following format:

```
TERM RQDLI
```

Note: A TERM call causes a CICS/VS synchronization point. Also, a CICS/VS synchronization point causes a TERM call if a PSB is still scheduled by the transaction. Refer to the section on Recovery Services in the *CICS/VS Recovery and Restart Guide.*

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 the request, using the way 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 (transaction control area) or the UIB (user interface block) must always be checked first to ensure that the call was accepted.

Include code immediately following the call to examine the field UAPCBL 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
NORESP (Normal Response)	00000000
INVREQ (Invalid Request)	00001000
NOTOPEN (Not Open)	00001100

NORESP
: indicates that the requested function was completed normally and that the field UAPCBL contains the address of the PCB list.

INVREQ
: indicates that the field UDLTR contains one of the following error codes:

X'01'
: PSB name, as provided in the scheduling call, is not in the PSB dictionary.

X'03'
: The calling program has already successfully issued a scheduling (PCB) call that has not been followed by a termination (TERM) call.

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'07'
: A TERM call was issued when the task has already been terminated.

X'08'
: A data base call was issued when the task was not scheduled.

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 UDTLR 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
NORESP (Normal Response)	00000000
INVREQ (Invalid Request)	00001000

NORESP
indicates that the DL/I resources have been released.

INVREQ
indicates that the field UDLTR 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 *DL/I DOS/VS Messages and Codes.* The code normally begins with D but it begins with E if the message 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, UFCTR contains a X'0C' and UDLTR contains a X'02'. When the data base is not open, the fields contain X'0C' and X'01', respectively.

If any online tasks must wait for a resource owned by an MPS batch 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. It is labeled "RESRij."

If the MPS Restart facility is to be used with RPG programs, users will have to write their own VSE checkpoint interface in another programming language (for example, assembler). RPG II does not have its own VSE checkpoint interface.

MPS batch jobs not using Program Isolation or MPS Restart 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. 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.
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 is considered to be an online task from a scheduling viewpoint.

Notes:

- If PCB and TERM calls are issued by MPS batch jobs, the jobs must not be run in a normal DL/I batch (not MPS) environment or in an MPS batch environment using Program Isolation or MPS Restart.
- DL/I application programs can access a data base that is resident on another CICS/VS system via the CICS/VS Intersystem Communication support. If an MPS batch application is to access a remote data base using Intersystem Communication or MPS Restart, the program must not issue PCB or TERM calls. PCB calls would receive an abnormal return code of X'08' in TCAFCTR (TCAFCRC in ANS COBOL), or UFCTR in RPG II, and X'09' in TCSDLTR, or UDLTR in RPG II. If issued in the MPS Restart environment, an error message will be issued and the batch partition will be cancelled.

RQDLI Commands Under CICS/VS

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 earlier in this chapter.

   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 ADDRESS LIST OF THE PCB ADDRESSES
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 PCB ADDRESS
C* LIST IS PUT INTO THE BASE OF THE DS DEFINING THE ADDRESS LIST
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 DATA BASE 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 use to include data structures, program 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 Translator and, therefore, the inserted text is untranslated 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.

Chapter 7: Data Base Reorganization/Load Processing

About This Chapter

This chapter introduces the function of data base reorganization in a DL/I environment. It introduces:

- data base reorganization, and
- load processing

The theme of this chapter is to provide a general overview; specific details on how to code the utility programs are contained in *DL/I DOS/VS Resource Definition and Utilities.*

DL/I provides nine utility programs of two types: 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, all pointer relationships cannot be resolved during initial loading. These utility programs are provided to resolve these relationships.

What is Reorganization

Reorganization changes the physical storage and/or structure of a data base to achieve the application's performance requirements. The two types of reorganization are:

- Physical reorganization, to optimize the physical storage of the data base, and
- Logical reorganization, to optimize the data base structure.

When to Reorganize

Most reorganizations recover space once occupied by deleted segments and/or to resequence segments physically in their logical order. The DL/I reorganization programs provide statistical data that can help you determine if a data base should be reorganized. 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 HD than for HISAM data bases, because HD reuses space freed by deleted segments, and because HD attempts 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:

- Reorganize DL/I data bases
- Connect logical relationships when initially loading logically related data bases
- Create secondary index data base(s) when loading or reorganizing data bases

The nine 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)
- Partial Reorganization (DLZPRCT1 and DLZPRCT2)
- Data Base Prereorganization (DLZURPR0)
- Data Base Prefix Resolution (DLZURG10)
- Data Base Prefix Update (DLZURGP0)
- Data Base Scan (DLZURGS0)

Reorganization of HD Data Bases

Two methods can be used to accomplish reorganization of HD data bases. The first method is for a full reorganization of a data base and uses 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.

The second method is for a partial reorganization of a data base, which performs similar functions to a full reorganization. The partial reorganization utility performs these functions only on a specified portion of the HD data base. This utility is comprised of two parts: the first part performs the pre-reorganization functions, and the second part performs the actual reorganization of the selected portion of the data base.

Logical Relationship Resolution

Four utility programs are used when initially loading and/or fully reorganizing data bases that are involved in logical relationships.

These logical relationship resolution utility programs are:

- The data base prereorganization utility

  This program controls the execution of the other three logical relationship resolution utilities.

- 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

  This program combines and sorts all input work files. These input files may be 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 the file generated by the prefix resolution utility as input.

Reorganization/Load Flowchart

Figure 7-1 shows the programs required for physical reorganization and/or logical relationship resolution during initial load or reorganization of an HD data base.

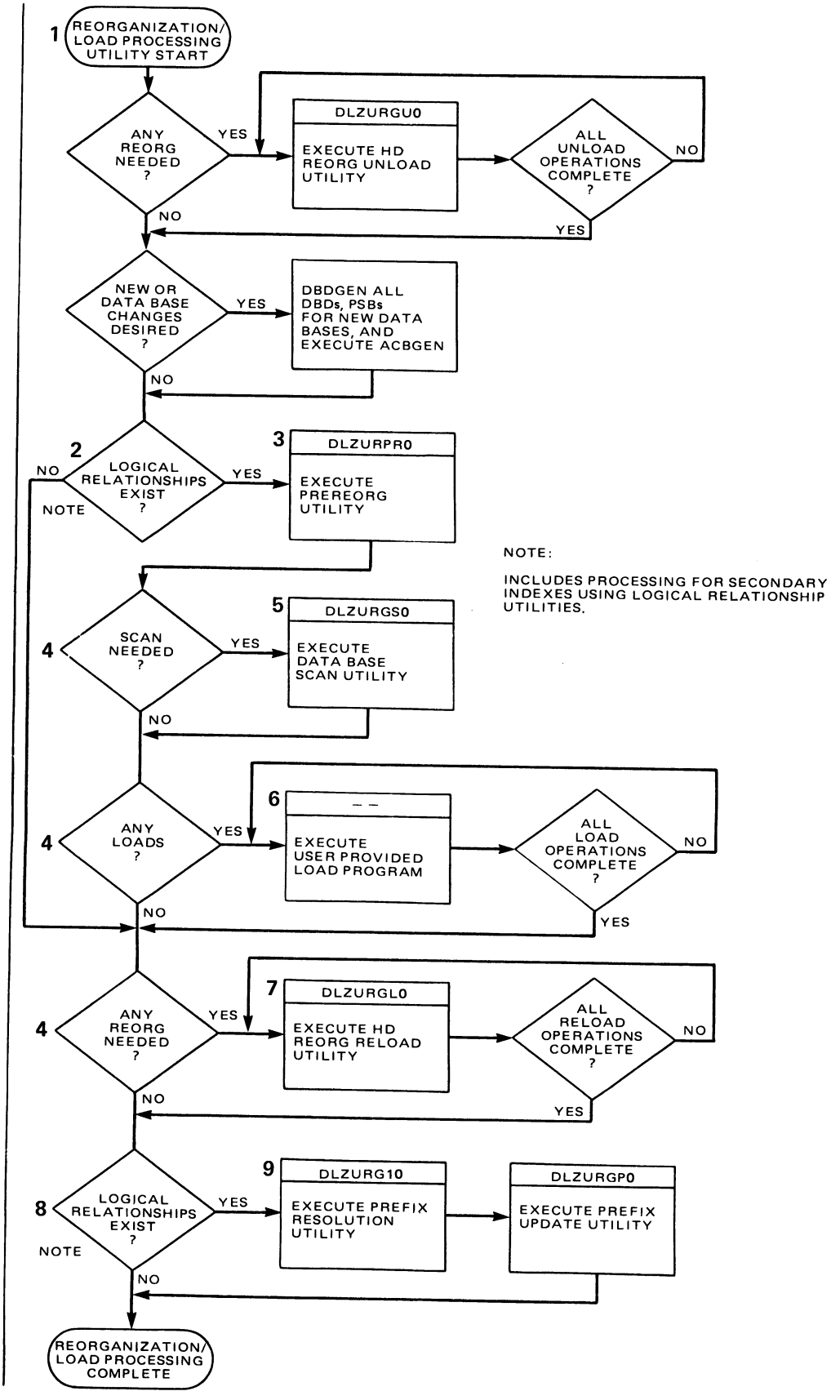


Figure 7-1. Reorganization/Load Flowchart

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 restrictions of data base logical relationship resolution utilities in *DL/I DOS/VS Resource Definition and Utilities.*

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. It can be executed once for all data bases to be scanned. A work file may be generated for each data base (or all data bases) 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. 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.

8. If any work files were generated during any of the data base operations that were executed, the YES branch must be taken. 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 to use.

When an indexed 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.

9. 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.

Partial Data Base Reorganization

The partial data base reorganization utility reorganizes a user-selected portion of data base records into a designated target area within the data base. For an indexed direct data base, the range is defined by a set of low-high key values in the primary index data set. For an HD randomized data base, the range is defined by a set of low-high relative block numbers in the root-addressable area.

Note: The DL/I DOS/VS Space Management Utilities, program number 5796-PKF, an Installed User Program, may be used to produce a report to assist you in identifying the ranges that need reorganization and the selection of a target area.

The format of the job control statements and the utility control statements required to run the partial data base reorganization utility are described in *DL/I DOS/VS Resource Definition and Utilities.*

Note: Data base structural changes cannot be made as they can, for example, in the full reorganization of a data base.

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 HD indexed 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 LOAD command 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 7-1.

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 LOAD command. 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 should be supplied as input to one execution of the prefix resolution utility. If there are no logical relationships, the work file WORKFIL will still be automatically opened by the HD Reload utility to prevent failure 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 DLZ007I 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

There are two ways of creating secondary indexes. One way is to create them automatically during initial loading or reloading. This is accomplished if all DLBL statements for the secondary index data bases are included in the job stream for the initial load or reload. However, because the index records are not normally in ascending key sequence, this usually leads to a significant performance degradation.

The other way of creating secondary indexes is to delay their creation until later using logical relationship utilities. One way of delaying their creation is to omit the DLBL statement(s) for the secondary index data base(s). This prevents the indexes from being created automatically. When using the HD Reload utility, there is another way to delay their creation. Leave the DLBL statement(s) as is, and include a BLDINDEX=NO control statement following the DLI parameter statement. If the BLDINDEX=NO statement is not provided, your labels will be used as submitted. In both cases, the logical relationship resolution utilities must be used to build the secondary index data base(s) and ASSGN and TLBL (or DLBL and EXTENT) statements must be provided for the work file (WORKFIL).

Note: When using the method of omitting the DLBL statements, message DLZ020I with a VSAM return code of X'80' occurs for every secondary index data base (because the index maintenance routine tries to open them), but loading does continue.

Resolution Utilities Overview

Figure 7-2 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 CONTROL DATA SET [CONTROL] CONTROL CARDS, optional
DATA BASE(S) CONTROL CARDS or CONTROL DATA SET [CONTROL]	DATA BASE SCAN UTILITY DLZURGS0	WORK FILE 1 [WORKFIL]
CONTROL DATA SET [CONTROL]	EACH USER LOAD PROGRAM physical DBD indicates presence of logical relationships or secondary indexes	LOADED DATA BASE(S) physical pointers established WORK FILE 1 [WORKFIL] SECONDARY INDEXES, optional
CONTROL DATA SET [CONTROL]	HD RELOAD UTILITY DLZURGL0 physical DBD indicates presence of logical relationships or secondary indexes	RE-LOAD DATA BASE physical pointers established WORK FILE 1 [WORKFIL] SECONDARY INDEXES, optional
CONTROL DATA SET [CONTROL] 1 TO n WORK FILE 1's [WRKIN01-WRKIN0n]	PREFIX RESOLUTION UTILITY DLZURG10	MESSAGES SORT WORK FILES [SORT WK n] WORK FILE 2 [INTRMED] WORK FILE 3 [WORKFIL] control information and sequenced pointer values SECONDARY INDEX WORK FILE, optional [INDXWRK]
WORK FILE 3 [WORKFIL] SECONDARY INDEX WORK FILE, optional [INDXWRK]	PREFIX UPDATE UTILITY DLZURGP0	MESSAGES USER DATA BASE(S) logical pointers established SECONDARY INDEXES, optional

Figure 7-2. Loading Data Bases with Logical Relationships and/or Secondary Indexes

Figure 7-3 is an example of loading two data bases with logical relationships and secondary indexes

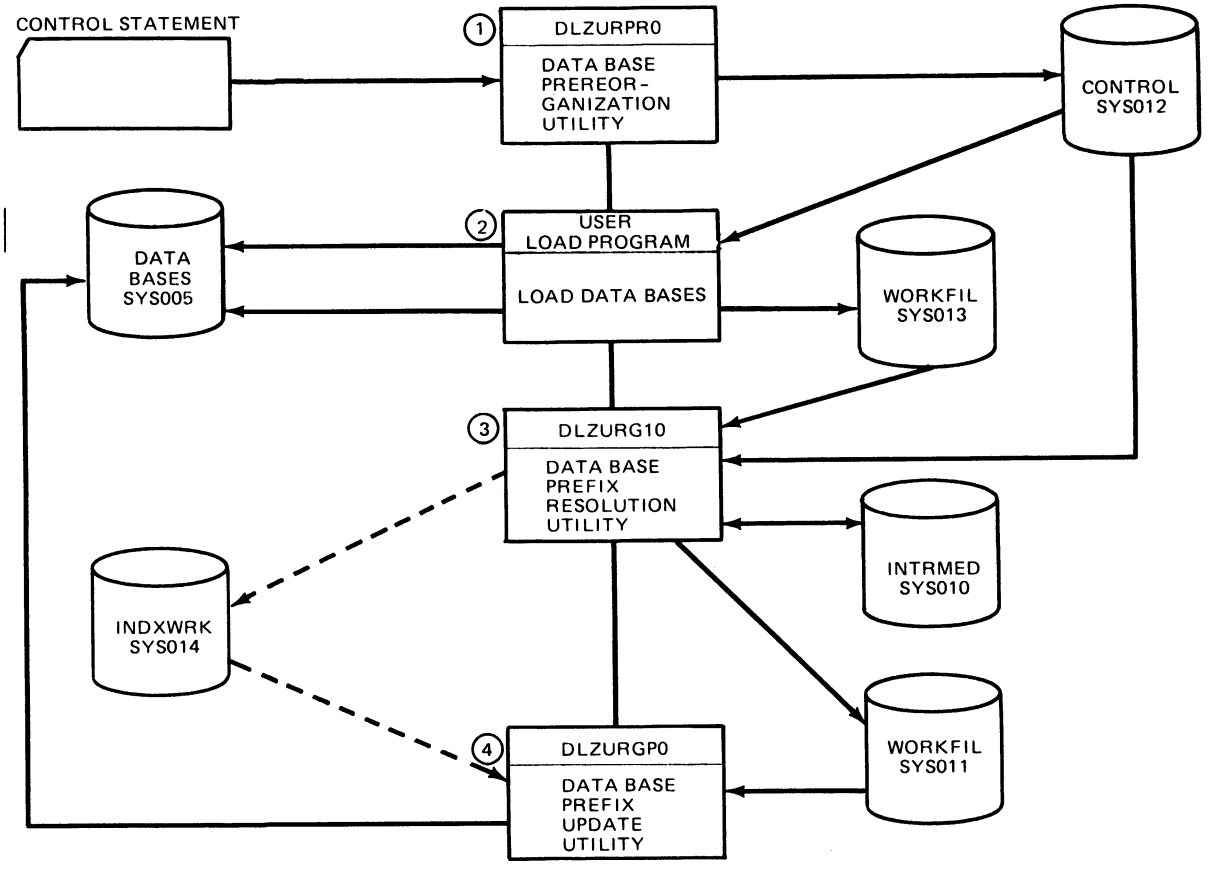


Figure 7-3. Loading Two Data Bases with Logical Relationships and Secondary Indexes

The following job control examples relate to Figure 7-3 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 SYS012,230
// EXEC DLZRRC00,SIZE=300K
ULU,DLZURPR0
DBIL=STDIDBP ,STDCDBP
OPTIONS=(NOPUNCH,STAT,SUMM)
/*
  /&
```

2
```
// JOB STJLDCST LOAD INVENTORY AND CUSTOMER DATA BASES
// OPTION PARTDUMP
// DLBL STDIDBC,'SAMPLE.INVEN',,VSAM
// DLBL STDCDBC,'SAMPLE.CUST',,VSAM
*  NOTE: NO DLBLS FOR SECONDARY INDEXES (BUILT BY PREFIX UPDATE)  - Note 1
// DLBL CONTROL,'CONTROL FILE',0,SD
// EXTENT SYS012,111111
// ASSGN SYS012,230
// DLBL WORKFIL,'WORKFILE J',0                                    - Note 2
// EXTENT SYS013,111111,1,0,1690,5
// ASSGN SYS013,230
// UPSI 00000010      NO LOG                                      - Note 3
// EXEC DLZRRC00,SIZE=300K
DLI,DLZSAM40,STBICLD,1,HDBFR=(6)
     •
     •  data statements
     •
/*
/&
```

Notes:

1. No DLBL 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 Chapter 4.

```
3 // JOB STJPRRES PREFIX RESOLUTION UTILITY
  // OPTION PARTDUMP
  // ASSGN SYS001,230
  // DLBL SORTWK1,'SORTWK FILE NR1',0,SD                          - Note 1
  // EXTENT SYS001,111111,1,0,3600,100
  // ASSGN SYS002,230
  // DLBL SORTWK2,'SORTWK FILE NR2',0,SD
  // EXTENT SYS002,111111,1,0,3700,100
  // ASSGN SYS003,230
  // 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 SYS013,230
  // DLBL CONTROL,'CONTROL FILE',0,SD
  // EXTENT SYS012,111111
  // ASSGN SYS012,230
  // ASSGN SYS010,230
  // DLBL INTRMED,'INTERMEDIATE WORK FILE',0,SD
  // EXTENT SYS010,111111,1,0,1620,20
  // ASSGN SYS011,230
  // DLBL WORKFIL,'PREFIX WORK FILE LR',0,SD
  // EXTENT SYS011,111111,1,0,1640,10
  // ASSGN SYS014,230
  // 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
  // DLBL STDIDBC,'SAMPLE.INVEN',,VSAM
  // DLBL STDCDBC,'SAMPLE.CUST',,VSAM
  // ASSGN SYS011,230
  // DLBL WORKFIL,'PREFIX WORK FILE LR',0,SD
  // EXTENT SYS011,111111
  // ASSGN SYS014,230
  // DLBL INDXWRK,'PREFIX WORK FILE SI',0,SD                      - Note 1
  // EXTENT SYS014,111111
  // DLBL STDCX2C,'SAMPLE.CUSTDX2',,VSAM
  // DLBL STDCX1C,'SAMPLE.CUSTDX1',,VSAM                          - Note 2
  // DLBL STDIX1C,'SAMPLE.INVDX',,VSAM                            - Note 2
  // UPSI 00000010     NO LOG                                     - Note 3
  // EXEC DLZRRC00,SIZE=300K
  ULU,DLZURGP0
  U
  /*
  /&
```

Notes:

1. These DLBL statements are required because the secondary indexes are to be created by this job.
2. These DLBL statements are always required for secondary indexes.
3. The UPSI byte setting is described in Chapter 4.

Chapter 8: DL/I Data Base Recovery/Restart

About This Chapter

This chapter contains information relative to the recovery and restart of your DL/I system following a failure. The parts include:

- Logging to capture all changes made to the data bases for use during the recovery process.
- Abnormal termination routine to minimize data base damage on a program check or other condition that prevents a normal end of job.
- Utility programs to correct errors or reconstruct data bases after damage.
- Checkpointing to minimize the time required to restart processing.
- VSAM and DL/I interaction for the recovery process.
- For more information about recovery and restart concepts and procedures, see *DL/I DOS/VS Recovery/Restart Guide.*

Introduction

Successful recovery and restart from 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, not much thought or effort is normally 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 8-1.

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.

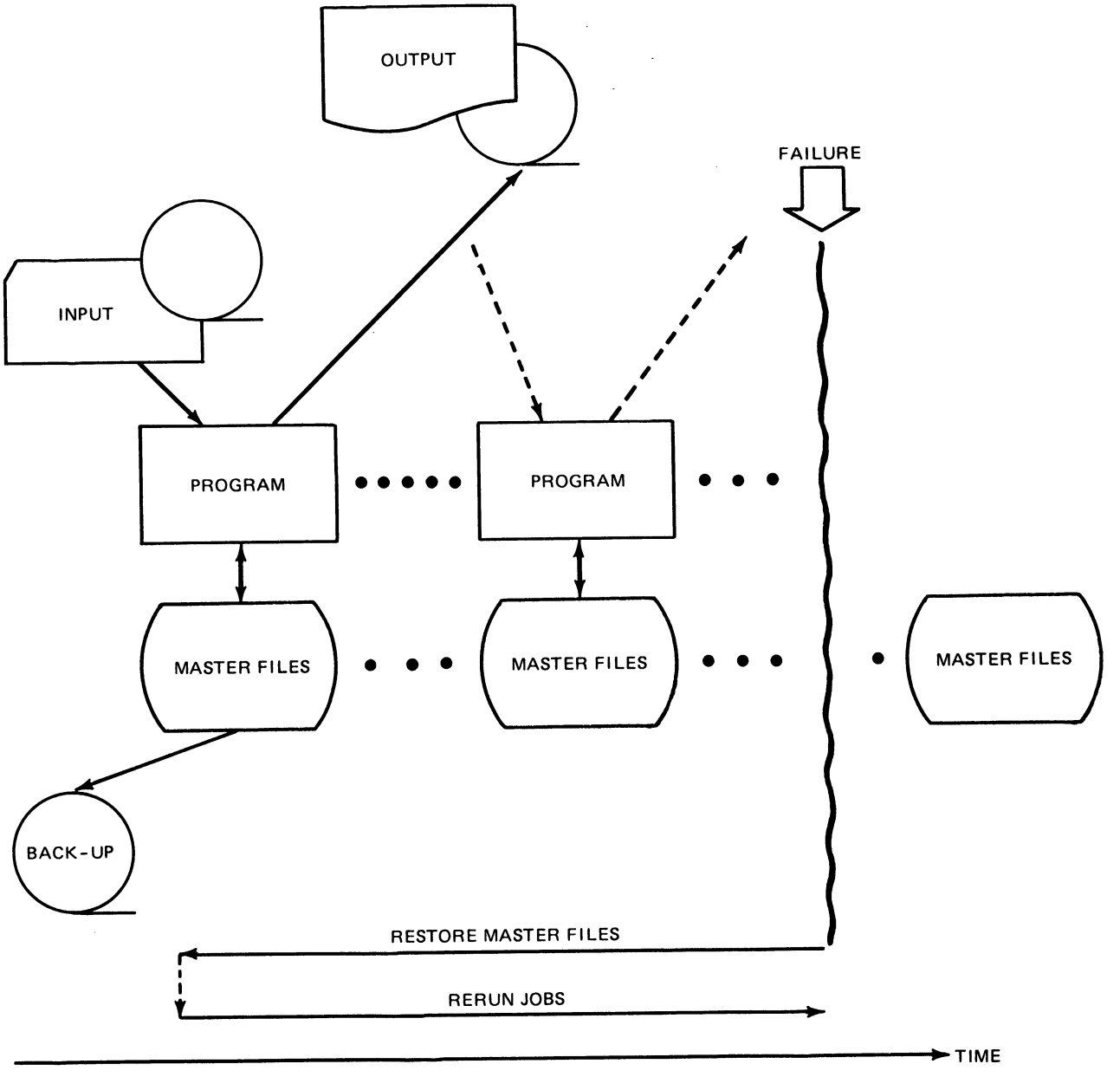


Figure 8-1. Traditional Recovery Approach

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 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 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 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/VSE, 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 can help you plan for recovery in your applications.

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. 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/VSE 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, and so on
- 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, and so on.

By reviewing the various DL/I messages found in *DL/I DOS/VS Messages and Codes,* you can identify other failure conditions and their symptoms.

When planning the recovery procedures for each of the failure conditions you define, consider the application needs and environment. For example, is the application online, batch, or 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 a 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 five hour image copy once a day may not. Similar times should also be developed for such operational steps as IPL, CICS/VS restart, and so on.

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 8-2.

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 flowchart 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 job control, operator responses to messages, and so on.

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:

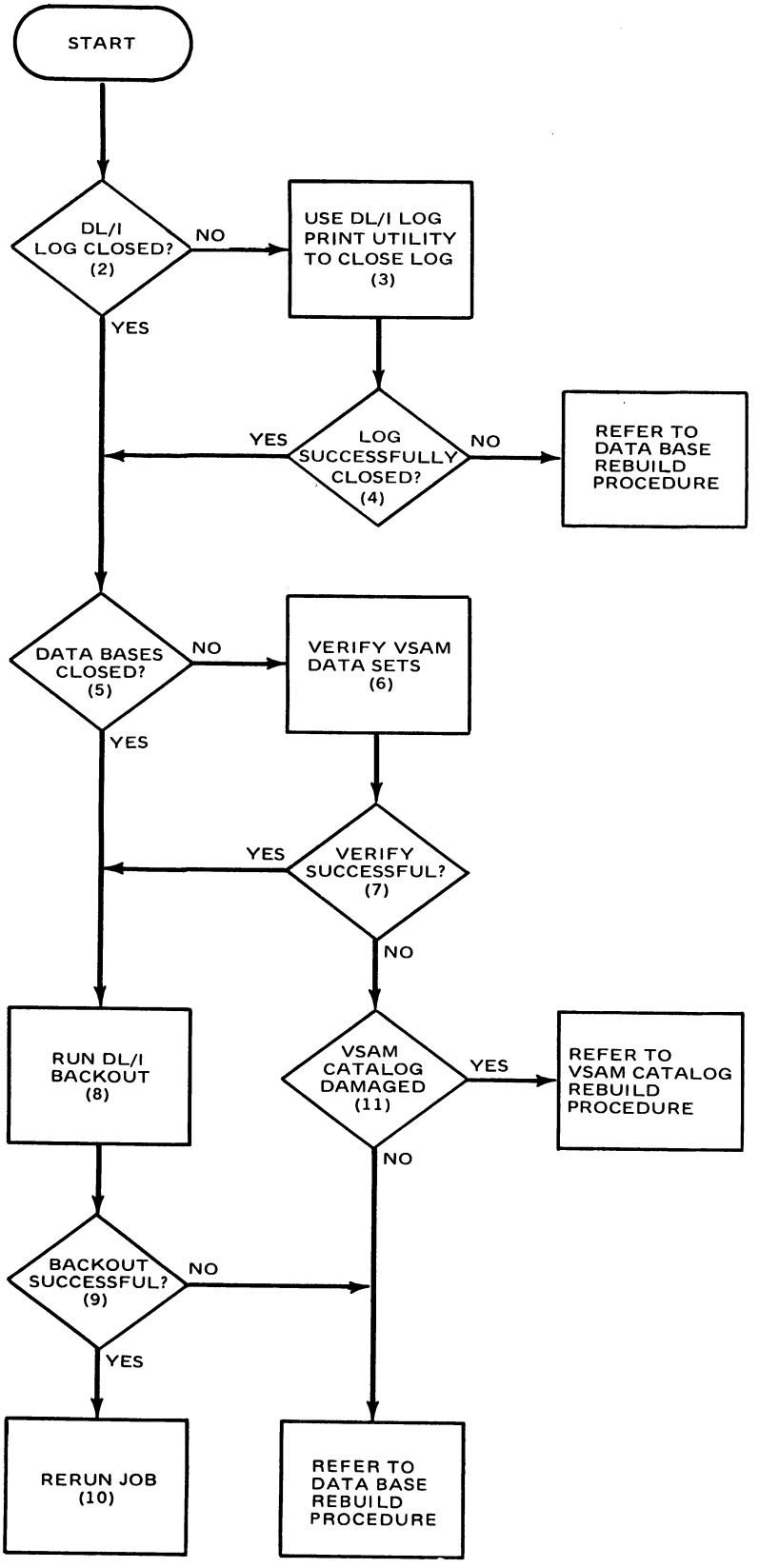


Figure 8-2. Sample Recovery Procedure Flowchart

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, but the data base does not reflect the change. This is the case when a failure occurs between the physical write of the log record and the physical write of the data base record.
3. Both the log and the data base reflect the latest change request made by the application program. This is the case when a failure occurs after the physical write of the data base record.

The various DL/I recovery utilities include logic to correct the data base for any of the above three conditions.

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 changes 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.

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 results in a physical write to the data base. This causes DL/I to force write the log more often than would normally occur if the HD access method were chosen. With the HD access method, 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 method tends to be deferred longer.
- Index data bases, either HD 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 non-MPS batch environment, you may use either of the first two choices.

In an MPS batch environment, the logging is performed from the CICS/VS partition, and no log device is assigned in the MPS batch 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. 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 gives you the best performance for your log. Figure 8-3 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 8-3, 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 8-3. 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 the batch, MPS, and CICS/VS environments. 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/VSE to invoke a STXIT AB exit. Possible causes of a STXIT AB wait can be found in *VSE/Advanced Functions Application Programming: User's Guide*, 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 are not executed.

DL/I uses the DOS/VSE 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, with the UPSI byte settings, to bypass 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, because 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 an MPS batch program cannot be bypassed. In an MPS batch environment, the DL/I abnormal termination routine performs the following functions:

- Writes message "DLZ096I" on the system console if the abnormal termination was caused by a program check or other error that causes DOS/VSE to invoke a STXIT AB exit. Possible causes of a STXIT AB can be found in *VSE/Advanced Functions Application Programming: User's Guide.* If DL/I's abnormal termination routine was entered directly from DL/I, rather than through a STXIT, then the "DLZ096I" message is not produced. Instead, DL/I produces messages indicating the explicit reason for the abend.
- Notifies DL/I in the CICS/VS partition that the MPS batch job is abending. If the condition that caused the abend originated in the MPS batch partition, the batch partition controller task supporting this batch partition issues a CICS/VS abend request.
- Deletes the XECB defined by this partition.
- Depending on the setting of the UPSI byte, it optionally produces a storage dump if the abnormal termination routine was entered via a STXIT AB or STXIT PC. If DL/I's abnormal termination routine was entered directly from DL/I rather than through a STXIT, a dump is always produced, regardless of the UPSI setting.
- Cancels the job. Any remaining job steps are not executed.

Because DL/I always uses STXIT AB and STXIT PC in an MPS batch environment, your application programs should not use any facilities that require STXIT AB or STXIT PC. 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 for 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 an 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 an MPS batch partition controller task, the XECBs defined by that task are deleted.
- Issues a TERMINATE command 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 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 sync-point record on the CICS/VS system journal.
- Releases any DL/I resources that belong to this task, such as, PST, PPST, PSB, and so on

- 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 command 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 command 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 does not consider these tasks to be "in-flight" and thus does not attempt to back them out. However, the DL/I backout utility, if executed in batch, backs out the data base updates made by these tasks because 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 shut-down is later attempted, DL/I issues message "DLZ068I", and does not attempt to close its data bases at that time. However, the DL/I log, if not assigned to the CICS/VS system journal, is closed at CICS/VS shut-down.

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/VSE 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.

No attempt is made to close the DL/I data bases on this error condition. However, the VSAM automatic close facility is invoked by DOS/VSE 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 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. 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 commands, 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 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 illustrated in Figure 8-4 consists of:

1. One PSB and one or more DMBs loaded from a core image library during DL/I initialization.
2. The input data base(s) 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 illustrated in Figure 8-4 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 Recovery and Restart Guide* for more information.

Data Base Recovery

Data base recovery is accomplished using the information on the DL/I system log and periodically created copies of the data base. The DL/I system log contains records that describe the modifications made to the data base. The number of 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.

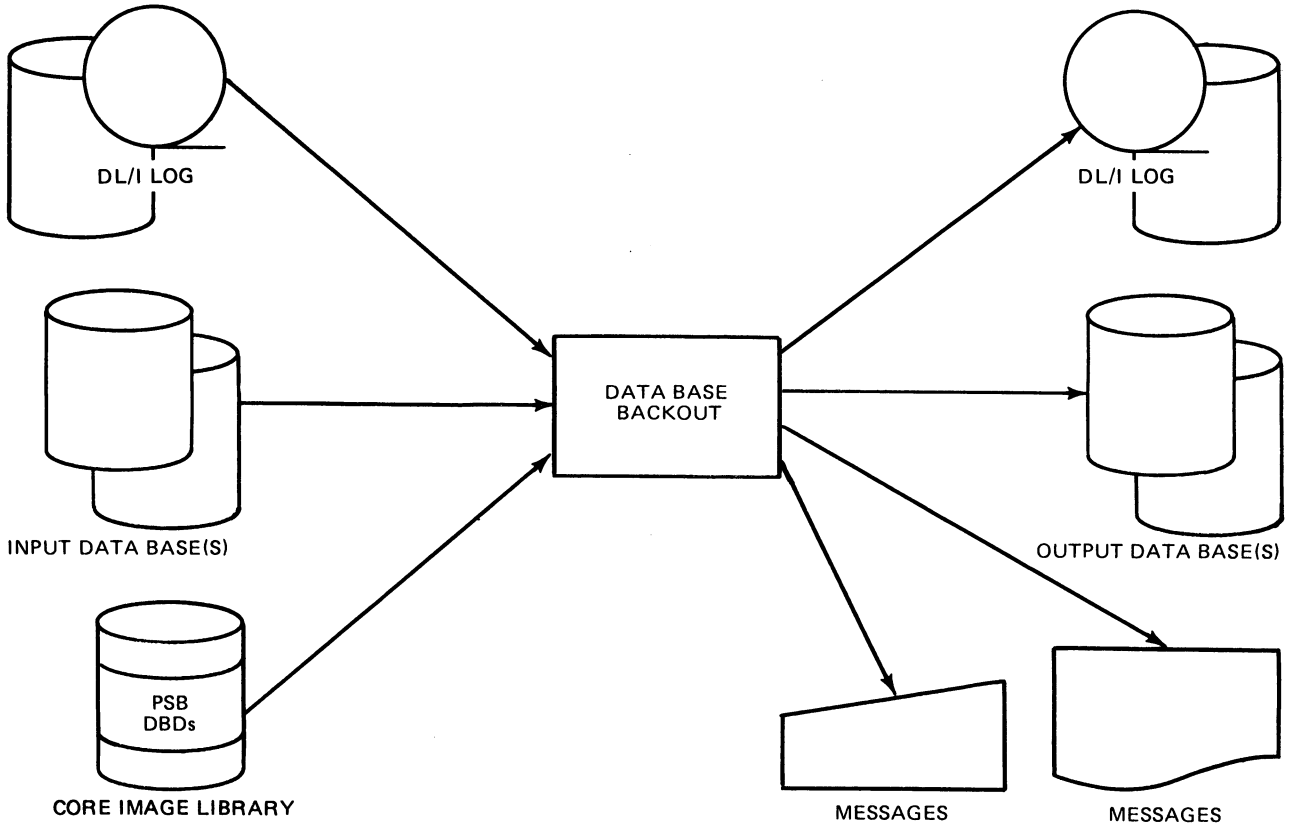


Figure 8-4. Data Base Backout Utility

- 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 HD 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, illustrated in Figure 8-5, 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.

DL/I Checkpoint Facility

DL/I provides a checkpoint facility to reduce the time required to back out data base changes after a failure.

Do not confuse this facility with the VSE CHKPT macro or program checkpointing facilities provided by COBOL and PL/I. The DL/I checkpoint facility does not use the VSE facility except when MPS Restart is active in the DL/I MPS batch environment.

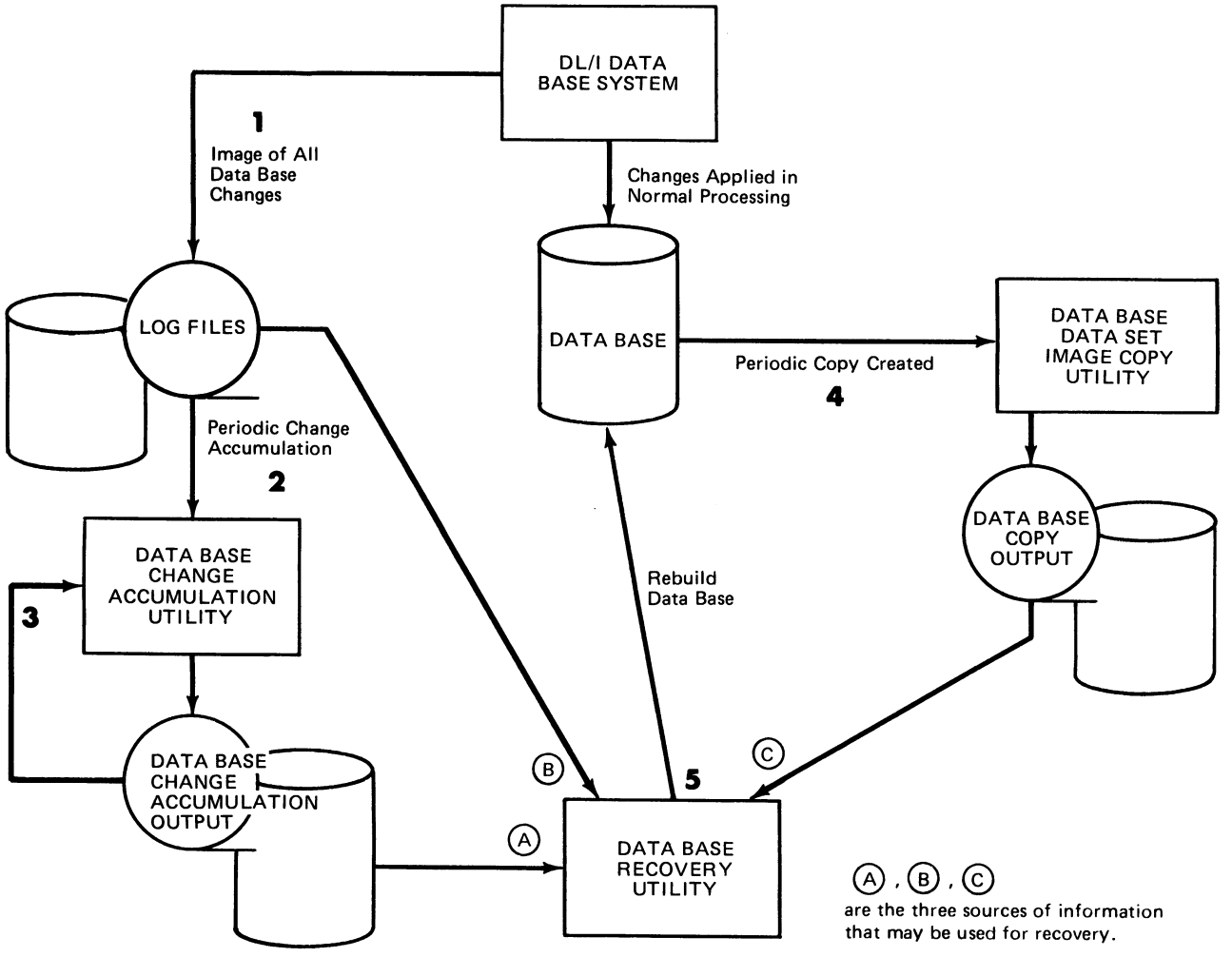


Figure 8-5. Data Base Recovery

CHECKPOINT Command

The CHECKPOINT command causes a checkpoint record to be written on the DL/I log as an aid to restart processing. For batch, the data base buffers are written to secondary storage and a checkpoint log record, with a user-supplied unique checkpoint identification, is written to the DL/I log.

For MPS and online tasks running with CICS/VS journaling active, a CHECKPOINT command is in effect a CICS/VS sync point command with the exception that the task's PSB scheduling status is not changed. Therefore, if the task has a scheduled PSB in effect at the time the CHECKPOINT is issued, a PSB scheduling command is not required after the CHECKPOINT command. In fact, a scheduling command issued under these circumstances causes a scheduling error. (See *DL/I DOS/VS Application Programming: High Level Programming Interface* for details on the scheduling and termination commands.)

You should issue periodic CHECKPOINT commands in long-running MPS and online tasks running with program isolation to keep control block size to a minimum.

You should also not use DL/I logging for MPS and online tasks. With DL/I logging active, a CHECKPOINT command 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 to be active. See *DL/I DOS/VS Application Programming: High Level Programming Interface* for additional details about the CHECKPOINT command.

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 command at intervals during your program's execution, you can shorten the time required for this recovery step. When a non-MPS batch program issues a CHECKPOINT command, 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 command has been sucessfully executed.
4. Returns control to the application program.

Thus, after a CHECKPOINT command, all data base changes are reflected on the data bases. However, the data bases are not closed on a CHECKPOINT command. After a CHECKPOINT command, your program must reestablish position in the data bases before continuing.

If your program were to fail immediately after the CHECKPOINT command, 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 because 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 command 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 8-6 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 command 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 command was issued. This may require logic in your program to reposition other files, restore internal counters and total fields, and so on. To assist in this restart, each checkpoint message written on the system console contains an eight-byte identification field that your program can specify in the CHECKPOINT command. This information can be used as input to your program

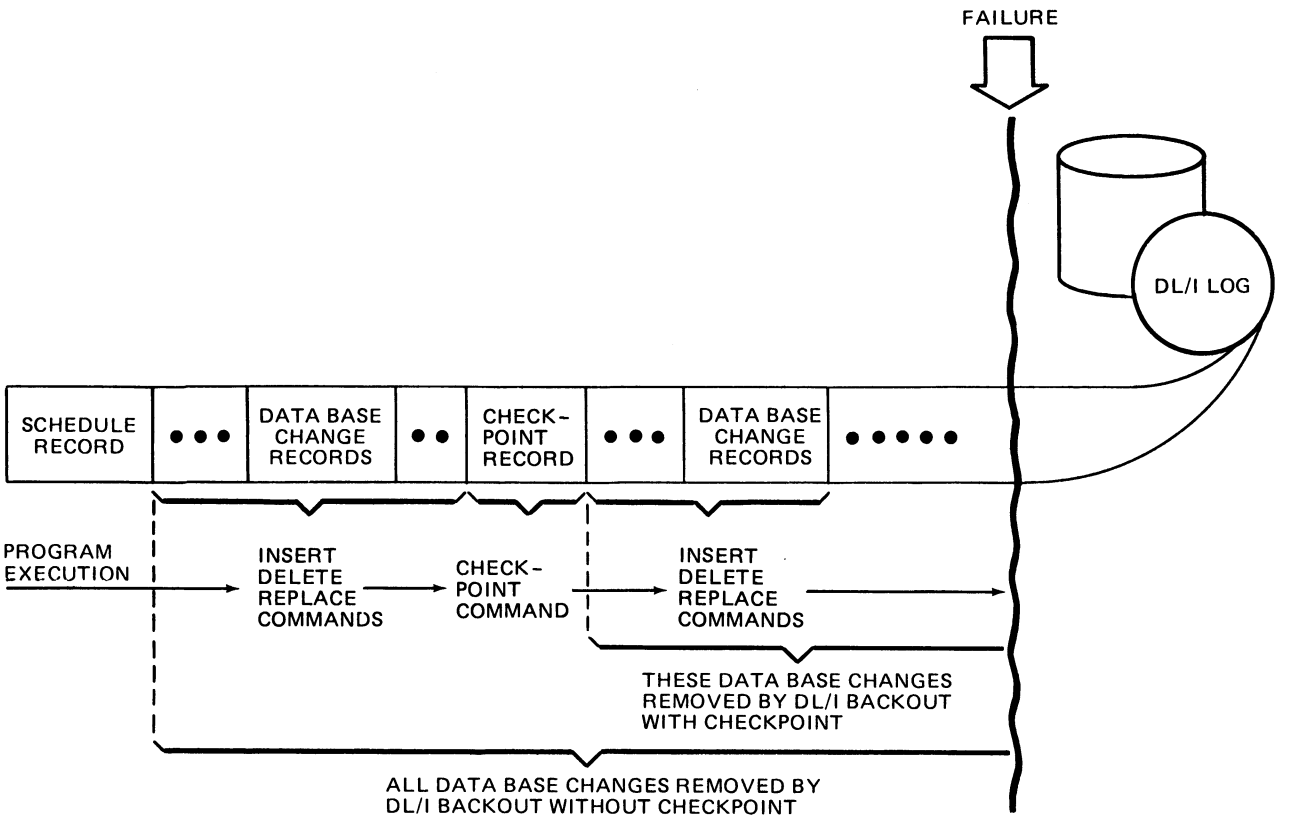


Figure 8-6. Checkpoint Records and DL/I Backout

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/VSE CHKPT macro and similar facilities in COBOL and PL/I cannot be used in your batch DL/I programs. The DOS/VSE 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 command, thus preventing the use of the DOS/VSE CHKPT facility.

DL/I Checkpoint in MPS Batch Programs

DL/I CHECKPOINT should be used in an MPS batch program operating in the following environment:

- The DL/I log is assigned to the CICS/VS system journal.
- Dynamic Transaction Backout support has been generated into the CICS/VS system.
- The DL/I Batch Partition Controller transaction, CSDC, has "DTB=YES" specified in its CICS/VS PPT entry.

When using the MPS Restart facility, additional environment requirements are:

- The DL/I Master Partition Controller transaction, CSDB, must have DTB=YES specified in its CICS/VS PCT entry.
- The MPS Restart purge temporary storage program, DLZMPUR0, must be specified in the PCT and PPT.

- START=AUTO must be specified in the System Initialization Table (SIT). Other specifications which support warm starts and emergency restarts should also be included.
- Temporary storage queue, DLZTSQ00, must have TYPE=RECOVERY specified in its CICS/VS TST entry.

When a DL/I CHECKPOINT command is issued in an MPS batch program under these conditions, DL/I performs the following functions:

1. Writes any altered data base records currently in main storage back to the data bases. This action will also force write the log records for these changed data base records, if necessary. At this point, any position in the data bases is lost.
2. Writes a CICS/VS sync-point record on the CICS/VS system journal.
3. Dequeues any records or segments enqueued for this program if Program Isolation is being used.
4. Writes message "DLZ105I" to the system console indicating that a DL/I CHECKPOINT command 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 command was issued.

To assure successful recovery and restart after a failure in an MPS batch program, the MPS Restart facility should be used. If you do not use the MPS Restart facility, your MPS batch programs should have their own restart capability. This includes procedures to reestablish the program environment at the point of the last checkpoint request before failure.

Using the MPS Restart Facility

To provide a restart capability for MPS batch users, DL/I supports the use of the VSE checkpoint/restart facility in conjunction with DL/I CHECKPOINT commands. A VSE checkpoint writes a copy of the partition in which the checkpoint was issued to disk or tape. The VSE RSTRT job control statement reloads this copy into the partition and passes control to the restart address that was specified when the VSE checkpoint was issued. For COBOL and PL/I programs using their respective VSE checkpoint interfaces, this corresponds to the next program statement after the one which invokes the VSE checkpoint.

To use MPS Restart, DLR must be specified on the DL/I parameter statement instead of DLI when the program is first executed. A VSE checkpoint must also be coded before each DL/I checkpoint in the application program. No other DL/I commands may be issued between the VSE checkpoint and the DL/I CHKP command.

MPS Restart provides the following functions:

- Combined Checkpoint Verification

  MPS Restart verifies that a VSE checkpoint is issued immediately before each DL/I CHKP command. This is called a combined checkpoint. A VSE checkpoint may be issued in PL/I and COBOL by using the checkpoint interfaces provided by those languages. It is recommended that the checkpoint ID returned by a VSE checkpoint be used as the checkpoint ID on the following DL/I CHKP call in PL/I and assembler language programs. This is not possible in COBOL because the returned checkpoint ID is not available to the application program. Using the VSE checkpoint ID on the DL/I CHKP call provides a

cross reference between the VSE and DL/I checkpoint messages issued to SYSLOG.

- MPS Batch Reinitialization

  The first DL/I command executed following a VSE restart must be a DL/I CHKP command. This will be the normal sequence when VSE checkpoints are placed immediately before each DL/I checkpoint. The CHKP command will automatically determine that a VSE restart has occurred and will reinitialize the MPS batch environment. Following successful reinitialization, the CHKP command will return control to the application program as if from a normal checkpoint and the program may continue processing.

- Checkpoint ID Notification and verification

  Besides the normal SYSLOG messages issued by VSE and DL/I when checkpoints are taken, a message containing the checkpoint ID (identifier) of the last successful combined checkpoint will be issued when a failure occurs. For individual job failures, the message is issued at the time of the failure. For system-wide failures, it is issued when MPS is started again in the online partition. The checkpoint ID contained in this message must be specified as a parameter on the VSE RSTRT job control statement when restarting an MPS batch job. MPS Restart will verify whether the checkpoint ID used for restart is the correct checkpoint ID. If it is not, DL/I will indicate the correct checkpoint ID which must be used and will cancel the job, allowing you to restart the job using the correct checkpoint ID.

Figure 8-7 shows a PL/I program using combined checkpoints for MPS Restart. A COBOL example is shown in Figure 8-8. Job control statements for execution and restart are provided in Appendix A.

Restrictions on Using VSE Checkpoint/Restart

When using the MPS Restart facility, you should be aware of certain restrictions that exist on the use of VSE checkpoint/restart:

```
*PROCESS XOPTS(CICS,DLI)...;
DLZPLIMP: PROCEDURE OPTIONS(MAIN);
          •
          •
DECLARE CHKPTID CHAR(8);
DECLARE RETCODE FIXED BIN(31);
          •
          •
CALL PLICKPT('',CHKPTID,'SYS100,2400',RETCODE); /* ISSUE A VSE CHKPT     */
IF RETCODE > 4 THEN                             /* VSE CHKPT ERROR       */
  DO;
    PUT EDIT('ERROR DURING CHECKPOINT. RETCODE=',RETCODE)(A,F(2));
    STOP;                                       /* ABNORMAL END          */
  END;
IF RETCODE = 4 THEN                             /* VSE RESTART OCCURRED */
    PUT EDIT('RESTARTED AT CHECKPOINT #',CHKPTID)(A);
EXEC DLI
     CHECKPOINT ID(CHKPTID);     /* ISSUE A DL/I CHKP WITH VSE CHKPT ID */
          •
          •
END DLZPLIMP;
```

Note: In the above program example, the VSE checkpoint ID (CHKPTID) is used as the checkpoint ID for the DL/I CHKP. This provides a cross reference between the normal checkpoint messages issued to SYSLOG as the result of taking VSE and DL/I checkpoints. While this is the recommended procedure for PL/I programs, it is not mandatory when using the MPS Restart facility.

Figure 8-7. PL/I Example of Combined Checkpoint in an MPS Batch Program Using MPS Restart

```
CBL XOPTS(CICS,DLI)...
ID DIVISION.
PROGRAM-ID. DLZCBLMP
          •
          •
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CHKPT-MSGS ASSIGN TO SYS101-UR-3203-S.
I-O-CONTROL.
* TAKE A VSE CHKPT ON EVERY WRITE TO CHKPT-MSGS
*
    RERUN ON SYS100-UT-2400-S
        EVERY 1 RECORDS OF CHKPT-MSGS.
DATA DIVISION.
FILE SECTION.
FD CHKPT-MSGS
    LABEL RECORD OMITTED
    RECORDING IS F
    RECORD CONTAINS 72 CHARACTERS
    DATA RECORD IS MSG-LINE.
01  MSG-LINE.
    02  FILLER    PIC X.
    02  ALL-71    PIC X(71).
          •
          •
WORKING-STORAGE SECTION.
01  INITMSG       PIC X(72) VALUE ' READY FOR VSE CHECKPOINTS '.
01  CHKPTMSG      PIC X(72) VALUE ' TAKING A VSE CHECKPOINT '.
01  CHKPTID       PIC X(8)  VALUE ' DL/ICHKP '.
          •
          •
PROCEDURE DIVISION.
     OPEN OUTPUT CHKPT-MSGS.
* INITIALIZE CHECKPOINT MESSAGE FILE
*
     WRITE MSG-LINE FROM INITMSG AFTER POSITIONING 2.
          •
          •
* TAKE AN IMPLICIT VSE CHECKPOINT
*
     WRITE MSG-LINE FROM CHKPTMSG AFTER POSITIONING 2.
* TAKE A DL/I CHECKPOINT
*
     EXEC DLI
          CHECKPOINT ID(CHKPTID)
     END-EXEC.
          •
          •
```

Note: In this example, a print file is defined so that a VSE checkpoint is issued each time a message is written to it. The initial message is necessary because COBOL does not start issuing VSE checkpoints until the write statement after the first write occurs. COBOL will also issue a VSE checkpoint when the print file is closed at the end of the program, but this additional checkpoint will not affect the function of MPS Restart. Because the invocation of a VSE checkpoint in COBOL programs is automatic, and the VSE checkpoint ID is not available to the application program, it cannot be used as the checkpoint ID on the DL/I CHKP command.

Figure 8-8. COBOL Example of a Combined Checkpoint in an MPS Batch Program Using MPS Restart

- VSAM files must be closed before a VSE checkpoint is issued (DL/I data bases used by MPS programs are in the online partition and are not affected by this restriction).
- RPG II does not support a VSE checkpoint interface. RPG II users will have to write their own interface in another programming language (Assembler Language, for example) if the MPS Restart facility is to be used with RPG programs.
- VSE restart cannot be used to restart programs that failed because of program logic errors. This is because a copy of the program, exactly as it was before it failed, is loaded back into the partition during a restart.

For additional information about such restrictions, see *VSE/Advanced Functions Application Programming: Reference* and *VSE/Advanced Functions Application Programming: User's Guide.*

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), and so on)
- 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, and so on).

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 changes 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; that is, 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 for all VSAM data sets. This will prevent more than one partition from updating the data set at a time. Use share option 2 for read-only data sharing. (See *DL/I DOS/VS Data Base Administration* for information on data sharing.) Share option 3 should never be used because 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 because 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 also should not be used with DL/I's VSAM data sets because this can 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 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/VSE writes a "4n26I" or "4n27I" message on the system console. Automatic close is invoked on a CICS/VS abend because the DL/I data base data sets are not closed by DL/I under these conditions.

If a VSAM data set was not closed, for example, because of 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 ensure that the high RBA value in the catalog is correct.

Chapter 9: DL/I Sample Application

About This Chapter

This chapter presents the DL/I sample application and includes descriptions for the:

- Sample application job stream
- Definition of the VSAM Master Catalog
- Sample program compression/expansion routine
- DL/I online sample load program
- DL/I online sample print program
- DL/I online sample application program

The DL/I sample application uses the Phase 3 customer and inventory data bases, and demonstrates an online order processing application. The sample application along with this manual can be 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 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 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
DLZCBL10*	DLZPLI10*	DLZRPG40
DLZCBL20*	DLZPLI20*	DLZRPG50
DLZCBL30*	DLZPLI30*	DLZRPG60
DLZCBL40	DLZPLI40	DLZRGMAP
DLZCBL50	DLZPLI50	
DLZCBL60	DLZPLI60	
DLZCBMAP	DLZPLMAP	

*HLPI program names

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

The PID distribution tape contains the 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 uses 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.

Note: Two job streams provided are for online and ACCESS. The ACCESS jobstream illustrates the new ACCESS statement, but this jobstream does not support the online sample applications (DLZCBL30, DLZPLI30, DLZCBL60, DLZPLI60, and DLZRPG60).

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 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 job control for the utility programs to ensure that the relative track specification and number of tracks specified meet the requirements of your installation. The job control for the reorganization and logical relationship resolution utilities used for the online sample application is included in Chapter 7 of this manual.

The PID distribution tape also 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 covers only DLZSAM40, DLZSAM50, and DLZSAM60.

Defining the VSAM Master Catalog

These job control and input job statements may be 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 SYS000,336
// 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.         *
*   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 SOURCE
         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 statements. The segment data starts in column 8 following the segment name. If continuation statements are needed for segment data, use a non-blank continuation punch in column 72. The segment data on continuation statements 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 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       00250000200000000000010
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
STSCSTA000000250000000000001000
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 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 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.

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 job control for the data bases must be included in the statements used to start CICS/VS. The job control is the DLBL and extent statements used to define the customer, inventory, and secondary index data bases to the 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)
```

Job Control Language for the Data Bases

```
// ASSGN SYS005,230
// 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 responds 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, increases. 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, and so on. 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 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 can 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 four 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 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. For information on installing DL/I, refer to the *DL/I DOS/VS Program Installation Directory* distributed with your DL/I system. Information on Batch Initialization is contained in this appendix.

Initialization of the DL/I Batch System

A batch DL/I program is executed as a called program in a 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.
DLZTPRT0 -	Trace Print
DLZPRCT1 -	Partial Reorganization (Part 1)

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/VSE. 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 statement on SYSIPT or directly from SYSLOG, depending on the UPSI byte setting.

The required control blocks and executable modules are loaded from a 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

DL/I application progams and some utility programs are run under the control of DL/I. This is done by specifying the name of the DL/I initialization module (DLZRRC00) in the EXEC statement as the program to be executed and not the name of the utility or application program. The actual user-written application 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, and whether this is a DL/I batch or utility initialization is then passed to DL/I through a parameter statement.

The requirements for this parameter statement are shown next. Note that in the formats shown for use by utility programs, a shortened form is used to illustrate optional parameters HDBFR, HSBFR, TRACE, ASLOG, and LOG. For the complete format of these optional parameters, see the statement format used by "DL/I Application Programs."

A detailed description of each parameter is given following the formats.

- *HD Reorganization Unload Utility* requires this parameter statement format to unload an entire data base.

```
ULU,DLZURGU0,dbdname
          [,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
            {1  }
```

This format is used for a selective unload.

```
PLU,DLZURGU0,psbname
```

- *HD Reorganization Reload Utility* requires this parameter statement format to reload a data base.

```
ULU,DLZURGL0,dbdname
          [,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
            {1  }
```

This format is used for a reload restart.

```
ULR,DLZURGL0,dbdname
          [,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
            {1  }
```

- *Data Base Prereorganization Utility* uses this parameter statement format.

```
ULU,DLZURPR0
```

- *Data Base Scan Utility* uses this parameter statement format.

```
ULU,DLZURGS0
```

- *Data Base Prefix Update Utility* uses this parameter statement format.

```
ULU,DLZURGP0
```

- *Part 2 of Partial Data Base Reorganization Utility* uses this parameter statement format.

```
DLI,DLZPRCT2,psbname
          [,{buf}][,HDBFR=][,HSBFR=][,TRACE=][,ASLOG=][,LOG=]
            {1  }
```

- *Data Base Data Set Recovery Utility* uses this parameter statement format.

```
UDR,DLZURDB0,dbdname
          [,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
            {1  }
```

- *Data Base Backout Utility* uses this parameter statement format.

```
DLI,DLZBACK0,psbname
          [,{buf}][,HDBFR=][,HSBFR=][,TRACE=][,ASLOG=][,LOG=]
            {1  }
```

- *DL/I Application Programs use this parameter statement format.*

```
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}
```

Parameter information must be entered from either SYSIPT or SYSLOG. The information must begin in column 1

A description of the parameters can be found in Chapter 4.

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` `[// TLBL`	`SYS011,cuu]` `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 statements may then be omitted.

`// ASSGN`
`// TLBL (Tape)`
`or`
`// DLBL (Disk)`
`// EXTENT`
	`SYSnnn,cuu`
	`filename`
	
	`filename`
	`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:
		nnn - logical unit assignment of the file.
		cuu - physical unit assignment of the file.
		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 Data Base Administration* for storage requirements.
Note: The above job control statements may contain additional information. Refer to the publication *VSE/Advanced Functions System Control Statements* SC33-6095, for more detailed information concerning 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; that is, the data bases they access may not be shared across several partitions while these programs are executing. The programs in this category 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.

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
{DLR}
```

Note: If you are using the MPS Restart facility, DLR must be specified instead of DLI. In the normal DL/I batch (not MPS) environment, DLR is treated the same as DLI.

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 Data Base Administration* for storage requirements.

No DLBL or TLBL statements are required to describe the data bases or DL/I log in the MPS batch job stream. This information is contained in the job control for the CICS/VS partition.

Note: The above job control statements may contain additional information. Refer to the publication *VSE/Advanced Functions System Control Statements*, SC33-6095, for more detailed information concerning the control statements.

Executing MPS Batch Programs

CICS/VS Release 1.6 or a subsequent release is required for DL/I Release 1.7. 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 program isolation will be used. There are no parameters for CICS/VS system generation specifically for MPS. Program Isolation 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 (DFHDBP1$) and the transaction backout program (DFHTBP1$) 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 1.7 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)*, SC33-0070, for more information on this subject.

The phase name of the // EXEC statement must be DLZMPI00. A SIZE parameter is required. In the batch partition, the size parameter and partition size can be made smaller than those used previously. There are less DL/I control blocks, no buffer pools, no action modules, and less initialization/nucleus code.

Executing MPS Batch Programs Using MPS Restart

Shown below are the execution job control statements for an MPS batch program using MPS Restart. Included in this example are statements that assign a tape to contain checkpoint records written by VSE checkpoints.

```
// JOB       UPDATE
// MTC       REW,280
// ASSGN     SYS100,280
// EXEC      DLZMPI00,SIZE=256K
DLR,INVUPDT,INVMSTR
                •
                •
  DATA CARDS IF REQUIRED
                •
                •
/*
/&
```

The MPS Restart facility is invoked for an MPS batch job by using the DLR function code in the parameter input to DL/I. This function code (see above example) replaces the DLI function code used for normal batch and MPS batch jobs. The DLR function code must be used when the job is first started and not just when it is restarted.

Restarting an MPS Batch Program Using MPS Restart

The following steps are required to restart an MPS batch program after a failure:

1. Get the VSE checkpoint ID from the SYSLOG message.
   a. If the individual MPS batch job failed, a message containing the correct checkpoint ID for restart is issued by DL/I at the time of failure.
   b. If there was a system failure, the message is issued when MPS is started again in the online partition.
2. Use the VSE checkpoint ID on the VSE RSTRT job control statement. The RSTRT statement is used instead of the EXEC statement when the job is resubmitted for execution.

The job control statements in the following example will restart the program in the previous job control example from checkpoint 0010. Note that the jobname must be the same on the restart job as it was on the job that failed.

```
// JOB       UPDATE
// MTC       REW,280
// ASSGN     SYS100,280
// RSTRT     SYS100,0010
DLR,INVUPDT,INVMSTR
                •
                •
  DATA CARDS IF REQUIRED
                •
                •
/*
/&
```

For additional information on the function, use, and restrictions of the VSE checkpoint/restart facility, see *VSE/Advanced Functions Application Programming: User's Guide.*

Restart Considerations

- If an MPS batch program using MPS Restart does not issue a combined checkpoint before a failure, it must be started over from the beginning using the EXEC job control statement rather than the RSTRT statement. For an individual job failure, this is indicated in the message issued at the time of failure. For a system failure, no message is issued for such jobs when MPS is started again in the online partition.
- The VSE restart facility requires the jobname to be the same on a restart job as it was on the job that failed. It also requires that the VSE partition start and end at the same addresses as when the job failed.
- During a VSE restart, a data check (tape checkpoint files) or end-of-file (disk checkpoint files) may occur if the checkpoint ID specified is greater than the actual number of checkpoints taken before the failure.
- On a restart, parameter input is ignored by DL/I because the parameters were already read and saved when the job first started. However, if the parameter input statement was included on SYSIPT (instead of having been entered from SYSLOG) when the job first started, it is important that one also be included when the job is restarted. This is because DL/I will attempt to position SYSIPT past the parameter input statement when the job is restarted.

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 job control statements 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 is in operation 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". This technique will not work if logging is required because there is no way to dynamically assign the DL/I log device if non-MPS operation is required.

The job control to execute a batch DL/I program in this environment uses an EXEC statement that specifies the XECB testing program's name, not DLZMPI00 or DLZRRC00. No other changes in the job control statements is required.

An example program to do this for read-only (no logging) DL/I programs is given below.

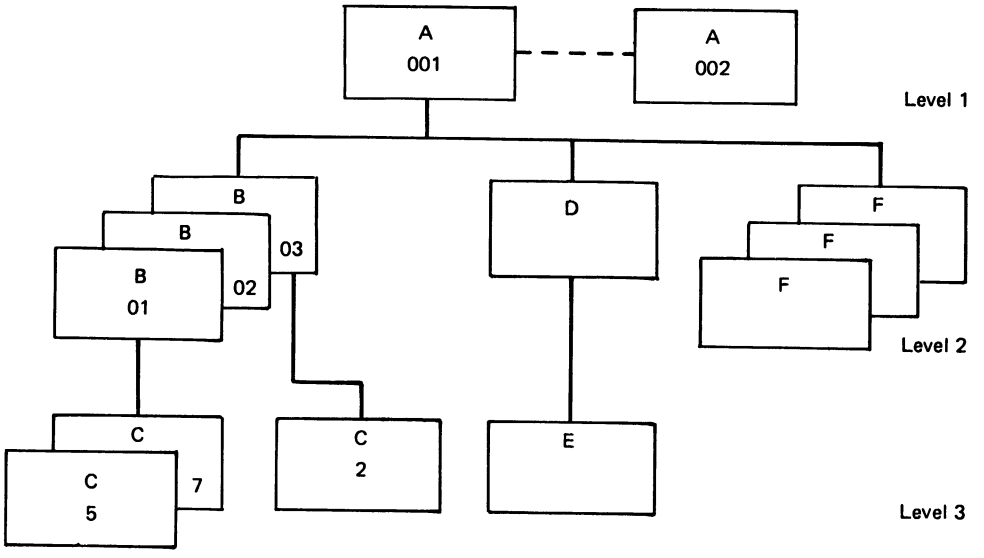
```
DLZCTRL   CSECT
          BALR    R12,0           ESTABLISH BASE REG (R12)
          USING   *,R12           ...AND TELL ASSEMBLER
          OPEN    PRINTER,CONSOLE
          XECBTAB TYPE=CHECK,XECB=DLZXCB00
          LTR     R15,R15         IS MPS ACTIVE?
          BNZ     NOMPS           ...NO
          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 MPS batch job, the operating system job accounting information for the batch partition does 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 is charged to the DL/I batch partition controller task, CSDC.

Glossary

A number of terms and phrases used in describing DL/I DOS/VS are either new to most readers, or have new meanings. To improve the readability and your understanding of this and other DL/I DOS/VS publications, the significant and important terms are defined in this Glossary. Some of the definitions refer to the representative DL/I hierarchical structure shown in Figure G-1.



- Each block represents a segment.
- The segment names are A through F.
- The numbers represent the sequence field (keys). If no number is present, the segment has no key.
- The lines connecting the segment blocks show the hierarchical paths.

Figure G-1. Representative DL/I Hierarchical Structure

ACB. (1) Application control blocks (DL/I). (2) Access method control block (VSAM).

ACBGEN. Application control block generation.

access method control block (ACB). A control block that links a program to a VSAM data set.

access method services. A multifunction utility program that defines VSAM data sets (or files) and allocates space for them, and lists data set records and catalog entries.

ACT. Application control table.

addressed direct access. In systems with VSAM, the retrieval or storage of a data record indentified by its relative byte address, independent of the record's location relative to the previously retrieved or stored record. (See also *keyed direct access, addressed sequential access, keyed sequential access,* and *relative byte address.)*

addressed sequential access. The retrieval or storage of a VSAM data record relative to the previously retrieved or stored record. (See also *keyed sequential access, addressed direct access,* and *keyed direct access.)*

aggregate. See *data aggregate.*

anchor point (AP). See *root anchor point.*

application control blocks. The control blocks created from the output of DBDGEN and PSBGEN, e.g., a DMB of an internal PSB created by the ACB utility program.

application control block generation (ACBGEN). The process by which application control blocks are created.

application control table (ACT). A DL/I online table describing those CICS application programs that utilize DL/I.

argument. (1) (ISO)[1] An independent variable. (2) (ISO)[1] Any value of an independent variable . (3) Information, such as names, constants, or variable values included within the parentheses in a DL/I command.

attribute. A property of an entity expressing a value. Synonymous with *field.*

backout. The process of removing all the data base updates performed by an application program that has terminated abnormally. See also *dynamic backout.*

---

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1.

batch checkpoint/restart. The facility that enables batch processing programs to synchronize checkpoints and to be restarted at a user-specified checkpoint.

batch processing. A processing environment in which data base transactions requested by applications are accumulated and then processed periodically against a data base.

Boolean operator. (1) (ISO)[1] An operator, each of the operands of which and the result of which, take one of two values. (2) An operator that represents symbolically relationships, such as AND, OR, and NOT, between entities.

business process. A defined function of a business enterprise usually interrelated through information requirements with other business processes. For example, personnel management is the business process responsible for employee welfare from pre-hire through retirement. It is related to the accounting business process through payroll.

CA. Control area.

call. (1) (ISO)[1] The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (2) (ISO)[1] In computer programming, to execute a call. (3) The instruction in the COBOL, PL/I, or Assembler program that requests DL/I services. For RPG II, see *RQDLI command.* See also *command.*

control area (CA). A collection of control intervals. Used by VSAM to distribute free space.

checkpoint. A time at which significant system information is written on the system log, and optionally, the system shut down.

child. Synonymous with *child segment.*

child segment. A segment one level below the segment which is its parent, with a direct path back up to the parent. Depending on the structure of the data base, a parent may have many children; however, a child has only one parent segment. Referring to Figure G-1:

- All the B, D, and F segments are children of A-001.
- C-5 and C-7 are children of B-01 (and A-001) but not children of the other B segments.
- B-02 has no children.

See also *logical child* and *physical child.*

CI. Control interval.

combined checkpoint. In MPS batch programs, a VSE checkpoint followed immediately by a DL/I CHKP call. The two checkpoints together form one logical checkpoint and allow the use of the VSE Restart facility to restart MPS batch programs.

command. The statement in DL/I High Level Programming Interface (HLPI) that requests services for application programs written in COBOL or PL/I. See also *call.*

command code. An optional addition to the SSA that provides specification of a function variation applicable to the call function.

concatenated key. The key constructed to access a particular segment. It consists of the key fields, including that of the root segment and successive children down to the accessed segment.

control interval (CI). (1) A fixed length amount of auxilary storage space in which VSAM stores records and distributes free space. (2) The unit of information transmitted to or from auxillary storage by VSAM.

data aggregate. A group of data elements that describe a particular entity. Synonymous with *segment.* See also *data element.*

data base (DB). (1) (ISO)[1] A set of data, part of the whole of another set of data, and consisting of at least one file, that is sufficient for a given purpose or for a given data processing system. (2) A collection of data records comprised of one or more data sets. (3) A collection of interrelated or independent data items stored together without unnecessary redundancy to serve one or more applications. See *physical data base* and *logical data base.*

data base administration (DBA). The tasks associated with defining the rules by which data is accessed and stored. The typical tasks of data base administration are outlined in *DL/I DOS/VS Data Base Administration.*

data base administrator (DBA). The person in an installation who has the responsibility (full or part time) for technically supporting the use of DL/I.

data base description (DBD). A description of the physical characteristics of a DL/I data base. One DBD is generated and cataloged in a core image library for each data base that is used in the installation. It defines the structure, segment keys, physical organization, names, access method, devices, etc., of the data base.

data base integrity. The protection of data items in a data base while they are available to any application program. This includes the isolation of the effects of concurrent updates to a data base by two or more application programs.

data base organization. The physical arrangment of related data on a storage device. DL/I data base organizations are hierarchical direct (HD) and hierarchical sequential (HS). See *hierarchical direct organization* and *hierarchical sequential organization.*

data base record. A collection of DL/I data elements called segments hierarchically related to single root segments.

Referring to Figure G-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.

data base reorganization. The process of unloading and reloading a data base to optimize physical segment adjacency, or to modify the DBD.

data communication (DC). A program that provides terminal communications and automatic scheduling of application programs based on terminal input. For example, CICS/DOS/VS.

data dictionary. (1) A centralized repository of information about data, such as its meaning, relationship to other data, usage, and format. (2) A program to assist in effectively planning, controlling, and evaluating the collection, storage, and use of data. For example, DOS/VS DB/DC Data Dictionary.

data element. The smallest unit of data that can be referred to. Synonymous with *field.* See also *data aggregate.*

data field. Synonymous with *field.*

data independence. (1) The concept of separating the definitions of logical and physical data such that application programs do not depend on where or how physical units of data are stored. (2) The

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1.

reduction of application program modification in data storage structure and access strategy.

data management block (DMB). The data management block is created from a DBD by the application control blocks creation and maintenance utility, link edited, and cataloged in a core image library. The DMB describes all physical characteristics of a data base. Before an application program using DL/I facilities can be run, one DMB for each data base accessed, plus a PSB for the program itself, must be cataloged in a core image library. The DMBs and the associated PSB are automatically loaded into main storage from the core image library at the beginning of the application program execution (their loading is controlled by the parameter information supplied to DL/I at the beginning of program execution).

data set. A named organized collection of logically related records. They may be organized sequentially, as in the case of DOS/VSE SAM, or in key entry sequence, as in the case of VSE/VSAM. Synonymous with *file.*

data set group (DSG). A control block linking together a data base with the data sets comprising this DL/I data base.

DB. Data base.

DBA. (1) Data base administration. (2) Data base administrator.

DBD. Data base description.

DBDGEN. Data base description generation -- the process by which a DBD is created.

DB/DC. Data base/data communication.

DC. Data communication.

dependent segment. A DL/I segment that relies on at least the root segment (or on another segment at a level immediately above its own) for its full hierarchical meaning. Synonymous with *child segment.*

destination parent. The physical or logical parent segment reached by the logical child path.

device independence. The concept of writing application programs such that they do not depend on the physical characteristics of the device on which data is stored.

DIB. DL/I interface block.

direct access. The retrieval or storage of a VSAM data record independent of the record's location relative to the previously retrieved or stored record. (See also *address direct access* and *keyed direct access*). Contrast with *sequential access.*

distributed data. The ability of DL/I application programs to access a data base that is resident on another processor.

distributed free space. See *free space.*

DL/I documentation aid. An extension to ACBGEN that collects DBD and PSB information and stores it in SQL/DS tables. This information can subsequently be accessed directly by ISQL.

DL/I interface block (DIB). Variables automatically defined in an application program using HLPI to receive information passed to the program by DL/I during execution. Contrast with *PCB mask.*

DMB. Data management block.

DSG. Data set group.

DTF. Define the file -- a control block that connect a program to a SAM data set.

dynamic backout. A process that automatically cancels all activities performed by an application program that terminates abnormally.

entity. A item about which information is stored. It has properties that can be recorded. Information about an entity is a record.

entry sequenced data set (ESDS). A VSAM data set whose records are physically in the same order as they were put in the data set. It is processed by addressed direct access or addressed sequential access and has no index. New records are added at the end of the data set.

ESDS. Entry sequenced data set.

exclusive intent. The scheduling intent type that prevents an application program from being scheduled concurrently with another application program. See *scheduling intent.*

FDB. field description block.

field. (1) (ISO)[1] In a record, a specified area used for a particular catagory of data, for example, in which a salary rate is recorded. (2) a unique or nonuique area (as defined during DBDGEN) within a segment that is the smallest unit of data that can be referred to. (3) any designated portion of a segment. (4) see also *key field.*

field level sensitivity. The ability of an application program to access data at the field level. See *sensitivity.*

file. (ISO)[1] A set of related records treated as a unit. See also *data set.*

forward. Movement in a direction from the beginning of the data base to the end of the data base, accessing each record in ascending root key sequence, and accessing the dependent segments of each root segment from top to bottom and from left to right. Referring to Figure G-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.

free space. Space available in a VSAM data set for inserting new records. The space is distributed throughout a key sequenced data set (KSDS) or left at the end of an entry sequenced data set (ESDS). Synonymous with *distributed free space.*

free space anchor point. A fullword at the beginning of a control interval pointing to the first free space element in this CI.

free space element. In HD data bases, the portions of direct access storage not occupied by DL/I segments are called and marked as free space elements.

---

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1.

FSA. free space anchor point.

FSE. free space element.

HD. Hierarchical direct.

HDAM. Hierarchical direct access method.

HIDAM. Hierarchical indexed direct access method

HIDAM index. A data base that consists of logical DL/I records, each containing an image of the key field of a HIDAM root segment. A HIDAM index data base consists of one VSAM KSDS (keyed sequenced data set).

hierarchic sequence. The sequence of segment occurrences in a data base record defined by traversing the hierarchy from top to bottom, front to back, and left to right.

hierarchical direct access method (HDAM). Provides for direct access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a randomizing routine. An HDAM data base consists of one VSAM entry sequence data set (ESDS).

hierarchical direct organization. An organization of DL/I segments of a data base that are related by direct addresses and may be accessed through an HD randomizing routine or an index.

hierarchical indexed direct access method (HIDAM). Provides for indexed access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a HIDAM index data base. A HIDAM data base consists of one VSAM Entry Sequenced Data Set (ESDS) and its associated index.

hierarchical indexed sequential access method (HISAM). Provides for indexed access to a DL/I data base. A HISAM data base consists of one VSAM key sequenced data set (KSDS) and one VSAM entry sequenced data set (ESDS).

hierarchical sequential access method (HSAM). The segments of a DL/I HSAM physical data base record are arranged in sequential order with the root segments followed by the dependent segments. HSAM data bases are accessed by the DOS/VSE sequential access method (SAM).

hierarchical sequential organization. An organization of DL/I segments of a data base that are related by physical adjacency.

hierarchy. (1) An arrangement of data segments beginning with the root segment and proceeding downward to dependent segments. (2) A "tree" structure.

high level programming interface (HLPI). A DL/I facility providing services to application programs written in either COBOL or PL/I Optimizer language through commands.

HISAM. Hierarchical indexed sequential access method.

HLPI. High level programming interface.

HS. Hierarchical sequential.

HSAM. Hierarchical sequential access method.

index data base. An ordered collection of DL/I index entries (segments) consisting of a key and a pointer used by VSAM to sequence and locate the records of a key sequenced data set (KSDS). Organized as a balanced tree of levels of index.

index data set. Synonymous with *index data base.*

index pointer segment. The segment that contains the data and pointers used to index the index target segments.

index record. A system-created collection of VSAM index entries that are in collating sequence by the key in each of the entries.

index segment. The segment in the index data base that contains a pointer to the segment containing data (the indexed segment). Synonymous with *index pointer segment.*

index set. The set of VSAM index levels above the sequence set. An entry in a record in one of these levels contains the highest key entered in an index record in the next lower level and a pointer that indicates the record's physical location.

index source segment. The segment containing the data from which the indexing segment is built.

index target segment. The segment pointed to by a secondary index entry, that is, by an index pointer segment.

indexed segment. A segment that is located by an index. Synonymous with *index target segment.*

intersection data. Any user data in a logical child segment that does not include the logical parent's concatenated key.

inverted file. In information retrieval, a method of organizing a cross-index file in which a key identifies a record. The items pertinent to that key are indicated.

ISPF. Interactive System Productivity Facility. A dialog manager for interactive applications. It provides control and services to support execution of the dialogs.

ISQL. The Interactive Structured Query Language is part of SQL/DS. It enables terminal users to work directly with the data without writing a program.

ISQL extract defines utility. This is a utility that creates ISQL EXTRACT DEFINE commands automatically to be used to define a DL/I data base to the extract facility of SQL/DS.

key. (1) (ISO)[1] One or more characters within a set of data that contains information about that set, including its identification. (2) The field in a segment used to store segment occurrences in sequential order. (3) A field used to search for a segment. See *primary key* and *secondary key.* (4) Synonymous with *key field* and *sequence field.*

Note: A segment may or may not have a key, that is, a sequence field. All root segments, except for HSAM and simple HSAM data bases, must have keys. DL/I ensures 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.

---

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1.

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.

key field. The field is a segment used to store segment occurrences in sequential ascending order. A key field is also a search field. Synonymous with *key* and *sequence field.*

key sequenced data set (KSDS). A VSAM file whose records are loaded in key sequence and controlled by an index. See also *keyed direct access* and *keyed sequential access.*

keyed direct access. The retrieval or storage of a data record by use of an index that relates the record's key to its physical location in the VSAM data set, independent of the record's location relative to the previously retrieved or stored record. See also *addressed direct access, keyed sequential access,* and *addressed sequential access.*

keyed sequential access. The retrieval or storage of a VSAM data record in its collating sequence relative to the previously retrieved or stored record, by the use of an index that specifies the collating sequence of the records by key. See also *addressed sequential access, keyed direct access,* and *keyed sequential access.*

KSDS. Key sequenced data set.

level. (1) (ISO)[1] The degree of subordination of an item in a hierarchic arrangement. (2) 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 DL/I data base is 15. For purposes of documentation and reference, the levels are numbered from 1 to 15, with the root segments being level number 1. Referring to Figure G-1:

- Three levels are shown.
- The A segments (roots) are at the highest level (Level 1).
- The C and E segments are at the lowest level (Level 3).

local system. (1) A specific system in a multisystem environment. Contrast with *remote system.* (2) The system in a multisystem environment on which the application program is executing. The local application may process data from data bases located on both the same (local) system and another (remote) system.

local view. A description of the data that an individual business process requires. See *system view.*

logical. When used in reference to DL/I components, logical means that the component is treated according to the rules of DL/I rather than physically as it may exist, or as it may be organized, on a physical storage device. For example, a logical DL/I record (a root segment and all of its dependent segments grouped) might be contained on several physical records or blocks on a storage device, and because of prior insertions and deletions, the segments might be in a different physical sequence than that by which they are retrieved logically for the application program by DL/I.

logical child. A pointer segment that establishes an access path between its physical parent and its logical parent. It is a physical child of its physical parent; it is a logical child of its logical parent. See also *logical parent* and *logical relationship.*

logical data base. A data base composed of one or more physical data bases representing a hierarchical structure derived from relationships between data segments that can be different from the physical structure.

logical data base record. (1) A set of hierarchically related segments of one or more segment types. As viewed by the application program, the logical data base record is always a hierarchic tree structure of segments. (2) All of the segments that exist hierarchically dependent on a given root segment, and that root segment.

logical data structure. A hierarchic structure of segments that is not based solely on the physical relationship of the segments. See also *logical relationships.*

logical parent. The segment a logical child points to. A logical parent segment can also be a physical parent. See also *logical child* and *logical relationship.*

logical relationship. A user defined path between two segments; that is, between logical parent and logical child, which is independent of any physical path. Logical relationships can be defined between segments in the same physcial data base hierarchy or in different hierarchies.

logical twins. All occurrences of one type of logical child with a common logical parent. Contrast with *physical twin.* See also *twin segment.*

MPS. Multiple partition support

MPS restart facility. The capability to restart an MPS batch job when a system or application program failure occurs, using VSE checkpoint/restart with the DL/I checkpoint command.

multiple partition support (MPS). Multiple partition support provides a centralized data base facility to permit multiple applications in different partitions to access DL/I data bases concurrently. MPS follows normal DL/I online conventions in that two programs cannot both update the same segment type in a data base concurrently. (With program isolation, two programs can concurrently update the same segment type; however, they cannot concurrently update the same segment. See *program isolation.*) However, two or more programs can retrieve from a data base while another program updates it. If one program has exclusive use of a data base, no other program can update it or retrieve from it.

multiple SSA. A series of segment search arguments (SSAs) included in a DL/I call to identify a specific segment or path. See also *segment search argument.*

object segment The segment at the lowest hierarchical level specified in a particular command. See also *path call.*

online. A operating environment in which DL/I is used with CICS/DOS/VS (or another data communication program) to permit end-users of application programs to access and store information in a data base through terminals.

option. A command keyword used to qualify the requested function.

parent. Synonymous with *parent segment.*

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1.

parent segment. (1) A segment that has one or more dependent segments. Contrast with *child.* (2) A parent is the opposite of a child, or dependent segment, in that dependent segments exist directly beneath it at lower levels. A parent may also itself be a child. Referring to Figure G-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 beginning point for the use of the get next in parent (GNP) or get hold next in parent (GHNP) functions. Parentage can only be established by issuing successful GU, GHU, GN, or GHN calls, or GET UNIQUE or GET NEXT commands.

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 following fields of the PCB:

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 G-1:

- The path to C-5 is A-001, B-01.
- The path to C-7 is the same as the path to C-5.
- There is no path to A-002 because it is a root segment.

path call. (1) The retrieval or insertion of multiple segments in a hierarchical path in a single call, by using the D command code and multiple SSAs. (2) The retrieval, replacement, or insertion of data for multiple segments in a hierarchical path in a single command, by using the FROM or INTO options specifying an I/O area for each parent segment desired. The object segment is always retrieved, replaced, or inserted.

PCB. Program communication block.

PCB mask. A skeleton data base PCB in the application program by which the program views a hierarchical structure and into which DL/I returns the results of the application's calls.

physical child. A segment type that is dependent on a segment type defined at the next higher level in the data base hierarchy. All segment types, except the root segment, are physical children because each is dependent on at least the root segment. See also *child segment.*

physical data base. An ordered set of physical data base records.

physical data base record. A physical set of hierarchically related segments of one or more segment types.

physical data structure. A hierarchy representing the arrangement of segment types in a physical data base.

physical parent. A segment that has a dependent segment type at the next lower level in the physical data base hierarchy. See also *parent.*

physical segment. The smallest unit of accessible data.

physical twins. All occurrences of a single physical child segment type that have the same (single occurrence) physical parent segment type. Contrast with *logical twins.* See also *twin segment.*

PI. Program isolation

pointer. A physical or symbolic identifier of a unique target.

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 the call function. Referring to Figure G-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.

primary key. The data element, or combination of data elements, within a segment that uniquely identifies an occurrence of that segment. See *key* and *secondary key.*

program communication block (PCB). Every data base accessed in an application program has a PCB associated with it. The PCB actually exists in DL/I and its fields are accessed by the application program by defining their names within the application program as follows:

COBOL - The PCB names are defined in the linkage section.
PL/I - The PCB names are defined under a pointer variable.
Assembler - The PCB names are defined in a DSECT.
RPG II - The PCB names are automatically generated by the translator, or may be defined by the user.

There are nine fields in a PCB:

1. Data base name
2. Segment hierarchy level indicator
3. DL/I results status code
4. DL/I processing options
5. Reserved for DL/I
6. Segment name feedback area
7. Length of key feedback area
8. Number of sensitive segments
9. Key feedback area.

Program Isolation (PI). A facility that isolates all data base activity of an application program from all other application programs active in the system until that application program commits, by reaching a synchronization point, that the data it has modified or created is valid.

This concept makes it possible to dynamically backout the data base activities of an application program that terminates abnormally without affecting the integrity of the data bases controlled by DL/I. It does not affect the activity performed by other application programs processing concurrently in the system.

program specification block (PSB). A PSB is generated for each application program that uses DL/I facilities. The PSB is associated with the application program for which it was generated and contains a PCB for each data base that is to be accessed by the program. Once it is generated, the PSB is cataloged in a core image library, and subsequently processed by a utility along with the associated DBDs to produce the updated PSB and DMBs; all of these are cataloged in a core image library for subsequent use by the application program during execution.

PSB. Program specification block

PSBGEN. PSB generation -- the process by which a program specification block is created.

qualified call. A DL/I call that contains at least one segment search argument (SSA). See also *segment search argument.*

qualified segment selection. The identification of a specific occurrence of a given segment type in a command, by using the WHERE option in the command for the desired segment. Contrast with *qualified SSA.*

qualified SSA. A qualified segment search argument contains both a segment name that identifies the specific segment type, and segment qualification that identifies the unique segment within the type for which the call function is to be performed. See also *segment search argument* and *multiple SSA.*

RAP. Root anchor point.

RBA. Relative byte address.

read-only intent. The scheduling intent type that allows a program to be scheduled with any number of other programs except those with exclusive intent. No updating occurs. See *scheduling intent.*

record. A data base record is made up of at least a unique root segment, and all of its dependent segments. See *data base record.*

relative byte address (RBA). The displacement of a stored record or control interval from the beginning of the storage space allocated to the VSAM data set to which it belongs.

remote system. In a multisystem environment, the system containing the data base that is being used by an application program resident on another (local) system Contrast with *local system.*

root anchor point (RAP). A DL/I pointer in an HDAM control interval that points to a root segment or a chain of root segments.

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/I services.

scheduling intent. An application program attribute defined in the PSB that specifies how the program should be scheduled if multiple programs are contending for scheduling. See *exclusive intent, read-only intent,* and *update intent.*

search field. In a given DL/I call, a field that is referred to by one or more segment search arguments (SSAs).

secondary index. Secondary indexes can be used to establish alternate entries to physical or logical data bases for application programs. They can also be processed as data bases themselves. See also *secondary index data base.*

secondary index data base. An index used to establish accessibility to a physical or logical data base by a path different from the one provided by the data base definition. It contains index pointer segments.

secondary key. A data element, or combination of data elements, within a segment that identifies -- and is used to locate -- those occurrences of the segment that have a property named by the key. See *key* and *primary key.*

segment. A segment is a group of similar or related data that can be accessed by the application program with one I/O function call. There may be a number of segments of the same type within a record.

segment name. A segment name is assigned to each segment type. Segment names for the different segment types must be unique within a data base. 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. Synonymous with *segment type.*

segment occurrence. One instance of a set of similar segments.

segment search argument (SSA). Describes the segment type, or specific segment within a segment type, that is to be operated on by a DL/I call. See also *multiple SSA, qualified SSA,* and *unqualified SSA.*

segment selection. The specifying of parent and object segments by name in a command. Selection may be either qualified or unqualified. Contrast with *segment search argument.*

segment type. A user-defined category of data. Referring to Figure G-1, there are six different types of segments; A through F.

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). Synonymous with *segment name.*

sensitivity. (1) A DL/I capability that ensures that only data segments or fields predefined as "sensitive" are available for use by a particular application program. The sensitivity concept also provides a degree of control over data security, inasmuch as users can be prevented from accessing particular segments or fields from a logical data base. (2) Sensitivity to the various segments and fields that constitute a data base is controlled, on a program-by-program basis, when the PSB for each program is generated. For example, a program is said to be sensitive to a segment type when it can access that segment type. When a program is not sensitive to a particular segment type, it appears to the program as if that segment type does not exist at all in the data base. Segment sensitivity applies to types of segments, not to specific segments within a type, and to all segment types in the path to the lowest level sensitive segment type.

sequence field. Synonymous with *Key field.*

sequence set. The lowest level of a VSAM index. It immediately controls the order of records in a key sequenced data set (KSDS). A sequence set entry contains the key of the highest keyed record stored in a control interval of the data set and a pointer to the control interval's physical location. A sequence set record also contains a pointer to the physical location of each free control interval in the fan-out of the record.

sequential processing. Processing or searching through the segments in a data base in a forward direction (see also *forward*).

simple HISAM. A hierarchical indexed sequential access method data base containing only one segment type.

source segment. A segment containing the data used to construct the secondary index pointer segment. See also *secondary index data base.*

SQL/DS. The Structured Query Language/Data System is a relational data base management system designed for end users. SQL/DS enables an end user to access data in online, interactive, and batch systems.

SSA. Segment Search Argument

status code. Each DL/I request for service returns a status code that reflects the exact results of the operation. The first operation that a program should perform immediately following a DL/I request is to test the status code to ensure that the function requested was successful. Following a command, the status code is returned in the DIB at the label DIBSTAT. Following a call, the status code is returned in field 3 of the PCB.

sync(h) point. Synonymous with *synchronization point.*

synchronization point. A logical point in time during the execution of an application program where the changes made to the data bases by the program are committed and will not be backed out. Synonymous with *sync point* or synch point.

A synchronization point is created by:

- a DL/I CHECKPOINT command or CHKP call
- a DL/I TERMINATE command or TERM call
- a CICS/VS synch point request
- an end of task (online) or an end of program (MPS-batch).

system view. A conceptual data structure that integrates the individual data structures associated with local views into an optimum arrangement for physical implimentation as a data base. See *local view.*

transaction. A specific set of input data that triggers the execution of a specific process or job.

twin segments. All child segments of the same segment type that have a particular instance of the same parent type. See also *physical twins* and *logical twins.*

twins. Synonymous with *twin segments.*

unqualified call. A DL/I call that does not contain a segment search argument.

unqualified segment selection. The identification of a given segment type in a command without specifying a particular occurrence of that segment type (without using the WHERE option). As a general rule, unqualified segment selection retrieves the first occurrence of the specified segment type. Contrast with *unqualified SSA.*

unqualified SSA. An unqualified SSA contains only a segment name that identifies the specific type of segment for which the I/O function is to be performed. As a general rule, the use of an unqualified SSA retrieves the first occurrence of the specified type of segment. See also *segment search argument.*

update intent. The scheduling intent type that permits application programs to be scheduled with any number of other programs except those with exclusive intent. See *scheduling intent.*

UPSI. User program switch indicator. A special 8 bit byte that allows each bit to be programmed by the user as "1" or "0". Bits may be read by a program to determine what the user wants to do.

Index

DL/I DOS/VS Guide for New Users (File No. S370/4300-50) Printed in U.S.A. SH24-5001-4

SH24-5001-4

DL/I DOS/VS
Guide for New Users
Order No. SH24-5001-4

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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:		
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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
Dept 812BP
1133 Westchester Avenue
White Plains, NY 10604, USA

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®

DL/I DOS/VS
Guide for New Users
Order No. SH24-5001-4

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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:		
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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

SH24-5001-4

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
Dept 812BP
1133 Westchester Avenue
White Plains, NY 10604, USA

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®

DL/I DOS/VS
Guide for New Users
Order No. SH24-5001-4

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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:		
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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

SH24-5001-4

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
Dept 812BP
1133 Westchester Avenue
White Plains, NY 10604, USA

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®

SH24-5001-04