***Title:*** *IBM DL/I DOS/VS Resource Definition and Utilities*
***Document Number:*** *SH24-5021-02*
***Build Date:*** *07/15/94 15:32:13* ***Build Version:*** *1.2*
***Book Path:*** *C:\IBMZLIB\BOOK\dlz21e02.boo*

# CONTENTS Table of Contents

---

# IBM Library Server Library

Find books in the catalog with titles, names, or doc numbers containing:

Find

Browse Bookcases

Browse Shelves

Administration

Help

# CONTENTS Table of Contents

---

## PREFACE.1.4 SQL/DS

*SQL/DS Terminal Users' Guide for VSE*, SH09-8034.

*SQL/DS Data Base Planning and Administration for VSE*, SH09-8023.

 *© Copyright IBM Corp. 1981, 1989*

# IBM Library Server Search

Book to be searched: *dlz21e02*
Search term:

Search Reset

Type of search:
- Exact
- Exact, any case
- Fuzzy

Where to search:
- ☑ Topic titles
- ☑ Topic text
- ☑ Indexed words

How to show results:
- List Topics by Importance
- List Topics in Sequence

---

# CONTENTS Table of Contents



***Title:*** *IBM DL/I DOS/VS Resource Definition and Utilities*
***Document Number:*** *SH24-5021-02*
***Build Date:*** *07/15/94 15:32:13* ***Build Version:*** *1.2*
***Book Path:*** *C:\IBMZLIB\BOOK\dlz21e02.boo*

# COVER Book Cover

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

**Resource Definition and Utilities**

Document Number SH24-5021-02

Program Number
5746-XX1

File Number S/370 9370-50

# EDITION Edition Notice

**Third Edition (May 1989)**

This edition is a major revision of SH24-5021-1. It applies to Version 1, Release 8 (Version 1.8) 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.



 *© Copyright IBM Corp. 1981, 1989*

# CHANGES Summary of Changes

Subtopics:



# CHANGES.1 Summary of Changes for SH24-5021-2 Versions 1.8 and 1.7.1

This edition has been revised to include enhancements introduced in DL/I DOS/VS 1.7.1 and 1.8. These include:

- Loading Data Bases
- Backup/Restore Improvements
- Directory of Programming Interfaces

The manual has been reformatted for easier use, and minor corrections have also been made.

 *© Copyright IBM Corp. 1981, 1989*

# CHANGES.2 Summary of Changes for SH24-5021-1 Version 1.7

This edition has been revised to include enhancements introduced in DL/I DOS/VS Release 1.7. These include:

- Variable Length Index Source Segments
- Various Utilities Operational Improvements
- MPS Restart
- DL/I Documentation Aid
- ISQL Extract Defines Utility

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

# CHANGES.3 Summary of Changes for SH24-5021-0 Version 1.6

This edition contains information previously appearing in *DL/I DOS/VS Utilities and Guide for the System Programmer,* SH12-5412, no longer published.

# PREFACE About this Manual

This manual describes procedures and utilities used in the physical implementation and maintenance of DL/I DOS/VS data bases. The book is directed primarily to those performing data base administration or system programming functions. It assumes the design of a data base is complete and that you are ready to implement it as a physical data base. Information about the tasks involved in planning, designing, using, and controlling DL/I data bases may be found in *DL/I DOS/VS Data Base Administration*. (See "Related Publications" in topic PREFACE.1 for a list of all related publications).

References to DL/I DOS/VS in this book are shortened to "DL/I".

The book is divided into eleven parts and an Appendix.

- Part 1, "Introduction" in topic 1.0
  explains coding and job control conventions and illustrates the control statement syntax used in this manual.

- Part 2, "Describing the Characteristics of DL/I Data Bases" in topic 2.0
  explains the data base description (DBD) generation procedure. The DBD describes to DL/I the contents of a data base. It names the segments, defines hierarchical relationships, and specifies the physical organization and characteristics of the data base data set.

- Part 3, "Describing an Application Program View of a Data Base" in topic 3.0
  explains the program specification block (PSB) generation procedure. The PSB defines the application data structure (view of a data base) required by a particular application program.

- Part 4, "Building Internal Control Blocks from DBDs and PSBs" in topic 4.0
  explains how to run the application control blocks creation and maintenance utility. This utility creates internal control blocks used by DL/I to provide data management services for application programs.

- Part 5, "Defining your Data Sets to VSAM" in topic 5.0
  discusses with examples how file attribute information to define VSAM data sets is obtained from the output listing of DBD generation.

- Part 6, "Describing DL/I Tables for an Online System" in topic 6.0
  explains how to define and generate application control tables (ACT) and storage layout control (SLC) tables. An ACT associates application programs in an online environment with one or more DL/I data bases. The SLC table is optional and is used to specify the sequence in which DL/I modules are loaded from core image library during DL/I initialization.

- Part 7, "Running DL/I Programs in Your System" in topic 7.0
  discusses how to execute DL/I application and utility programs in your system. Separate requirements are given for batch, online and multiple partition support (MPS) environment operation.

- Part 8, "Loading Data Bases" in topic 8.0
  describes initial and partial data base load procedures.

- Part 9, "Data Base Reorganization Utilities" in topic 9.0
  describes the DL/I utility programs used to reorganize data bases and to resolve prefix information if the data base uses logical relationships or secondary indexes.

- Part 10, "Data Base Recovery Utilities" in topic 10.0
  describes the DL/I utility programs used to repair or reconstruct DL/I data base data sets in the event of a failure.

- Part 11, "Extracting DL/I Data to SQL/DS" in topic 11.0
  describes a utility to create an ISQL routine for EXTRACT DEFINE commands.

- Part 12, "Appendix" in topic APPENDIX1
  identifies DL/I interfaces by which a customer-written program can function with DL/I.

Subtopics:

- PREFACE.1 Related Publications

---


---

# PREFACE.1 Related Publications

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

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

*DL/I DOS/VS Recovery/Restart Guide*, SH20-5030

Subtopics:



## PREFACE.1.1 Other DL/I Publications

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

*DL/I DOS/VS Library Guide and Master Index*, GH24-5008

*DL/I DOS/VS Release Guide*, SC33-6211

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

*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 Guide for New Users*, SH24-5001

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

*DL/I DOS/VS Diagnostic Guide*, SH24-5002

## PREFACE.1.2 CICS/DOS/VS Publications

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

*CICS/DOS/VS Customization Guide*, SC33-0131

*CICS/DOS/VS Resource Definition Guide*, SC33-0149

*CICS/DOS/VS Intercommunication Facilities Guide*, SC33-0133

*CICS/DOS/VS Recovery and Restart Guide*, SC33-0135

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

*CICS/DOS/VS Application Programmer*, SC33-0077

*CICS/DOS/VS Application Programmer*, SC33-0079

## PREFACE.1.3 VSE/SP and VSE/VSAM

*VSE/SP Messages and Codes*, SC33-6470

*VSE/Advanced Functions System Control Statements*, SC33-6354

*Guide to the DOS/VSE Assembler*, GC33-4024

*Using VSE/VSAM Commands and Macros*, SC33-6432

*Using the VSE/VSAM Space Management for SAM Feature*, SC24-5192

## PREFACE.1.4 SQL/DS

*SQL/DS Terminal Users' Guide for VSE*, SH09-8034.

*SQL/DS Data Base Planning and Administration for VSE*, SH09-8023.

 *© Copyright IBM Corp. 1981, 1989*

# Figures

---

# Tables

---



---

# 1.0 Part 1. Introduction

Subtopics:



## 1.1 Coding Conventions

Several conventions are followed in illustrating the format and coding of control statements used for DL/I resource definitions. These statements are written using DOS/VSE assembler language macro instructions and are, therefore, subject to the rules contained in *Guide to the DOS/VSE Assembler*. These assembler language syntax conventions apply:

- The control statements must follow these rules:
  - 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 must be separated from the previous operand by a comma with no intervening blanks.
  - Operands may be continued on the subsequent statements by placing a non-blank character in statement column 72, and by continuing the operand in statement column 16 on the continuation statement.
- Names for DBDs, PSBs, segment names, and field names may be composed only of alphameric characters: A-Z, 0-9, #, $, and @. The character @ must not be used for DBD or PSB names.
- Uppercase letters, stand-alone numbers, and punctuation marks must be coded exactly as shown. The only exceptions to this convention are brackets []; braces {}; ellipses ...; and subscripts. These are never coded.
- Lowercase letters and words and associated numbers represent variables for which specific information or specific values must be substituted.
- In the printed version of this manual, a blank symbol sometimes is used to indicate one blank position.
- Items or groups of items within [] are optional; they may be omitted if not required. Any item or group of items not within brackets must be coded.
- Stacked items, enclosed in braces ({}), represent alternative items. No more than one of the stacked items may be coded.
- If an alternative item is underlined, that item is implied; that is, DL/I automatically assumes that the underlined item is the choice if none of the items is coded.

- Ellipses, ..., indicate that the preceding item or group of items can be coded more than once in succession.

Two different formats are used in this book to illustrate these coding conventions:

Subtopics:

- 1.1.1 Format 1
- 1.1.2 Format 2

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 1.1.1 Format 1

The DBD control statement shown in this format example illustrates assembler language syntax conventions.

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS=HDAM<br>,RMNAME=(module-name,{anch}[,rbn[,bytes]])<br>{*1* }<br>[,IMSCOMP={YES}]<br>{*NO* } |

*In this example*

- DBD must be coded, beginning after column 1.
- NAME= must be coded, leaving at least one blank after DBD.
- In place of date-base-name, the actual name of the data base being described must be coded.
- The comma and ACCESS=HDAM must be coded.
- The comma and RMNAME=( must be coded. It must be followed by the name of the module to which RMNAME applies and, if other parameters are to follow, a comma.
- If a value other than 1 is required for the anch parameter, it must be coded next. If the parameter is omitted, the implied value of 1 is assumed.
- The rbn parameter is optional. If the rbn parameter is coded, the value must be preceded by a comma. This comma indicates, in the case of the anch parameter being omitted, that it has been omitted resulting in two adjacent commas being coded. If the anch parameter is not omitted, this comma separates the anch parameter from the rbn parameter.
- The bytes parameter is optional; it can only be coded if the rbn parameter has been coded. If the bytes parameter is coded, the value must be preceded by a comma to separate it from the rbn parameter.
- The closing parenthesis must always be coded.
- The IMSCOMP= operand is optional. If omitted, NO is assumed.

## 1.1.2 Format 2

The same DBD control statement shown in "Format 1" is now illustrated in another format used throughout this book. This format is more pictorial in that it graphically conveys a visual image of the control statement and its coding requirements.

Table 1. Sample Statement Syntax

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Data base name. 1-7 alphameric characters; the first must be alphabetic. |
| | ,ACCESS= | **H D A M** | HDAM = Hierarchical Direct Access Method |
| | ,RMNAME= | (° ° ° ° ° ° ° ° ,_ _ _ ,_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _) | Randomizing module name. Number of root anchor points in each block.(1) Control intervals in root-addressable area.(2) Record insert limit.(3) |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS. NO = Format compatibility not required. |

**Notes:**

1. This parameter is optional. If omitted, the default is 1 root anchor point.
2. This parameter is optional. If omitted, no upper limit check is performed on the rbn created by randomizing module.
3. This parameter is optional; it can only be coded if the previous parameter has been coded. If omitted, no limit is placed on the record size.

*Coding Example*

```
        DBD                                                  X
              NAME=STDCDBP,                                  X
              ACCESS=HDAM,                                   X
              RMNAME=(DLZHDC10,                              X
              3,                                             X
              100,                                           X
              600),                                          X
              IMSCOMP=NO
```

*In this example*

- DBD must be coded, beginning after column 1.

- NAME= must be coded, leaving at least one blank after DBD. The actual data base name must then be coded in place of the squares shown in the "Parameter" column.

  **Note:** Squares represent "required" parameters. The number of squares shown depicts the maximum length of your data.

- A brief explanation of each parameter is given in the "Description" column. Experienced DL/I users may find this information adequate when doing a resource definition. If more information is needed, however, it can be found in the description of the control statement and its operands.

- The comma and ACCESS=HDAM must be coded.

- The comma and RMNAME= must be coded. They must immediately be followed by an opening parenthesis and then by RMNAMEs first parameter, the actual name of the randomizing module.

  The next three parameters for RNAME are optional. "Notes" appearing below the illustration describe what happens if these parameters are omitted.

  **Note:** Underscores represent "optional" parameters. The number of underscores shown depicts the maximum length of your data.

  The closing parenthesis must be coded.

- The IMSCOMP= operand is optional. If omitted, the underlined choice shown in the Mini-Description is assumed. In this example, it is NO.

- A coding example follows the control statement illustration.

---

# 1.2 Job Control Conventions

This manual contains several job stream examples. For simplification of these examples, the following conventions apply:

**Job Control Statements**

Statements or parts of statements that are to be completed by the user are shown in lower-case, underscored, and enclosed in single quotes. For example:

```
// LIBDEF PHASE,CATALOG='private phase library'
// DLBL HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// ASSGN SYS011,DISK,VOL='volser',SHR
```

Note that in examples only the minimum required job control statements are given. All statements may contain additional parameters.

**Library Names**

DL/I is shipped in two parts: a production and a generation part. The production part contains the functions necessary to run your DL/I system. The generation part contains the functions necessary to generate your DL/I system. These parts usually reside in the following VSE/SP libraries:

```
PRD2.DBASE for the production library.
PRD2.GEDLI for the generation library.
```

In the examples in this manual, the following conventions have been established for names of the several libraries that might be accessed:

**System libraries:**

```
DL/I production library: 'DL/I prod lib'
DL/I generation library: 'DL/I gen lib'
CICS production library: 'CICS prod lib'
SQL production library: 'SQL prod lib'
```

**"Private" libraries:**

Usually the user-created DL/I blocks (for example DBDs, PSBs) are stored in a separate library, named 'DL/I user lib'.

For example:

```
.
.
.
// LIBDEF *,SEARCH='DL/I gen lib'
// LIBDEF PHASE,CATALOG='DL/I user lib'
// EXEC ASSEMBLY
```

**"Size" parameter for job execution:**
Within the examples, the size parameter in the EXEC statements shows only the minimum required execution size, and may vary depending on the individual installation.

Some utilities have a new headline layout with the date and time. The format for the date is either 'mmddyy' or 'ddmmyy', depending on the system default or the specification in the STDOPT or DATE command.

---

# 2.0 Part 2. Describing the Characteristics of DL/I Data Bases

This section assumes you have finished the design of your data bases and are ready to describe them to DL/I. The procedure you will use to do this is called *Data Base Description (DBD) Generation.* For brevity, it is simply called *DBDGEN.*

DBDGEN is a two-step procedure: the *definition* step and the *generation* step.

For the definition step, you prepare a set of statements that describe the structure of one data base in detail. Each data base you describe requires its own set of statements. Instructions on preparing these statements are given in this Part. Use the chapter that defines the type of access method and organization you want.

| **To define...** | **Use this chapter...** |
|---|---|
| HD data bases | 1 |
| HDAM data bases | 2 |
| HIDAM (and Primary INDEX) data bases | 3 |
| LOGICAL data bases and logical relationships in physical data bases | 4 |
| Secondary INDEX data bases and secondary indexing in physical data bases | 5 |
| SHISAM and HISAM data bases | 6 |
| SHSAM and HSAM data bases | 7 |

After you complete your data base definition, see Chapter 8, "Doing a DBD Generation" for information about doing the DBD generation step.

**Creating a DBD Interactively**

The DBD definition and generation procedures described here can also be done interactively on a 3270-type terminal by using the Interactive Macro Facility (IMF). See *DL/I DOS/VS Interactive Resource Definition* for more information.

Subtopics:

# 2.1 Chapter 1. Defining HD Data Bases

This chapter describes how to define HD data bases. HD data bases can be either randomized or indexed. DL/I decides which one is to be used according to the *primary access type* that you define in a DL/I control statement called ACCESS. The ACCESS statement is used only for HD data bases. It may be used to define primary randomized access, primary indexed access, and secondary indexed access.

Subtopics:



# 2.1.1 Input Structure and Rules

Figure 1 illustrates the rules for structuring a batch input stream to define an HD data base. The figure shows the types of control statements required, the number of each type that may be specified, and their specific order of sequence. A separate input stream is required for each data base you define.

***To Define an HD Organization Data Base . . .***

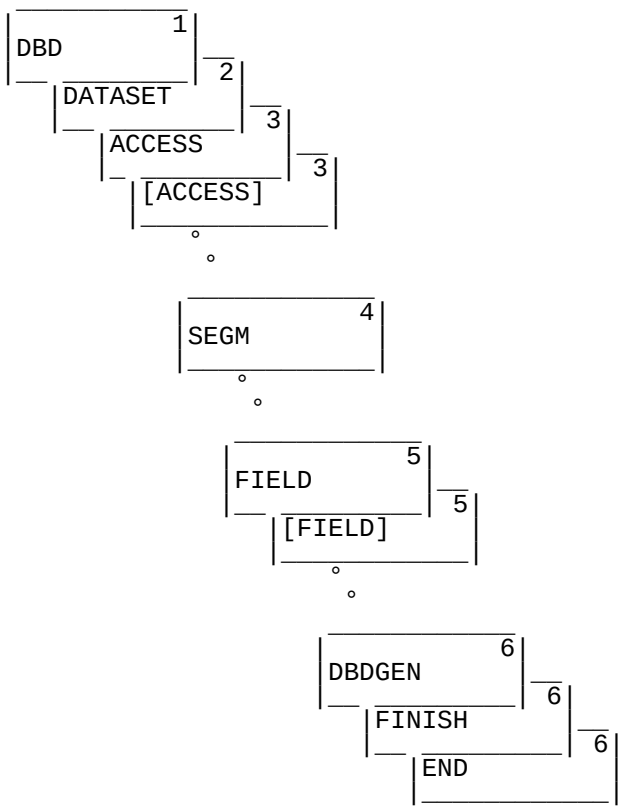



Figure 1. Batch Input Stream for HD Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.
2. Code one DATASET statement. It must follow the DBD statement.
3. Code one ACCESS statement to define the primary access type. It must be either primary randomized access or primary indexed access. Put this ACCESS statement after the DATASET statement. Optionally, you may code additional ACCESS statements to define secondary indexed access. Use one ACCESS statement for each secondary index you want defined for the data base. Place these additional ACCESS statements, if any, after the required one.
4. Code one SEGM statement for each segment type in your data base. The maximum allowed per data base is 255. Place the SEGM statements after

```
    the ACCESS statement(s).  Keep them in the same sequence as the
    segments appear in the hierarchic order of the data base being
    defined.
5.  Code at least one FIELD statement.  Use it to identify a "sequence
    field" within the root segment.  Optionally, you may define a maximum
    of 255 fields per segment type and 1020 fields per data base.  Use one
    FIELD statement for each field you define.  Place them after the SEGM
    statement defining the segment that contains those fields.
6.  Code one DBDGEN statement, one FINISH statement, and one Assembler END
    statement.  Place them in the sequence shown at the end of your input
    stream.
```

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 2.1.2 Control Statements Format

This section explains how to code DL/I control statements to define an HD data base. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introduction.

Subtopics:



## 2.1.2.1 DBD Statement

The format of the DBD statement when used to define an HD data base is:

Table 2. DBD Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Data base name; 1-7 alphameric characters; the first character must be alphabetic. |
| | ,ACCESS= | **H D** | HD for Hierarchical Direct. |
| | ,CIANPT= | _ _ _ | Number of root anchor points in each control interval.(1). |
| | ,PRIMCI= | _ _ _ _ _ _ _ _ | Number of control intervals in primary area.<br>1 through 16777215 or OVERLAP.(2). |
| | ,RILIM= | _ _ _ _ _ _ _ _ | Record insert limit.(3). |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

The following operands apply only to HD randomized data bases:

**Notes:**

1. This CIANPT operand is optional. If omitted, the default is 1.
2. This PRIMCI operand is optional. If omitted, the default is OVERLAP.
3. This RILIM operand is optional. If omitted, there is no record insert limit.

*Coding Example*

```
        DBD                                                    X
              NAME=STDCDBP,                                    X
              ACCESS=HD,                                       X
              CIANPT=3,                                        X
              PRIMCI=100,                                      X
              RILIM=600,                                       X
              IMSCOMP=NO
```

**NAME=**
Specifies the name of the HD data base. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Specifies the access method used for the data base. Here you must specify HD.

**CIANPT=**
Specifies the number of root anchor points per VSAM control interval. The number you specify must not exceed 255 or be less than 1. Typical values are from 1 to 5. This operand is optional. If you omit it, the default is 1.

**PRIMCI=**
Specifies the maximum number of control intervals to be located in the prime (or root-addressable) area of this HD data base. It must be in the range of 1 through 16777215, or the characters 'OVERLAP'.

The value specified for the parameter is compared to the relative block number (rbn) that is generated by your randomizing module each time it is called. If the generated rbn is greater than this value, this value is used in its place. If OVERLAP is specified, no upper limit check is performed on the rbn created by the randomizing module. OVERLAP is the default value.

**Note:** You must specify a numeric value in the operand if you plan to use one of the randomizing modules supplied by DL/I. If you omit it, or specify OVERLAP, unpredictable results will occur in the module during load execution.

**RILIM=**
Specifies the record insert limit. This is the maximum number of bytes that may be inserted into the primary area with one set of contiguous inserts for a single data base record (same root segment). The value specified for this parameter must not exceed 16777215 or be less than 1. The RILIM operand is optional. If omitted, there is no record insert limit.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

**Note:** Changing the IMSCOMP= entry for an existing data base without unloading and reloading the data base may cause unpredictable results, due to a different record size used by DL/I internally in the VSAM CI.

---



---

## 2.1.2.2 DATASET Statement

The format of the DATASET statement when used to define an HD data base is:

Table 3. DATASET Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Name of data set; 1-7 alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | _ _ _ _ _ | VSAM control interval size; (512-8192 bytes in steps of 512 bytes, 8192-30720 bytes in steps of 2048 bytes). |
| | ,SCAN= | _ _ _ _ _ | Number of cylinders (or blocks if FBA) to be scanned when searching for space during segment insertions.(2). |
| | ,FRSPC= | (_ _ _<br><br>,_ _) | Leave every nth block free (0-100 excluding 1). 0 is recommended for HD randomized access.(3).<br><br>Leave this percent free in every block (0-99).(3). |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate a value during DBD generation.
2. The SCAN operand is optional. See the operand description for details.
3. The FRSPC operand is optional. If omitted, the default is 0.

*Coding Example*

```
        DATASET                                  X
              DD1=STDCDBC,                       X
              DEVICE=3380,                       X
              BLOCK=2048,                        X
              SCAN=2
```

**DD1=**

Specifies the name you want used as the symbolic VSAM data set (ESDS) name for this HD data base. The name must be 1-7 alphameric characters long.

**DEVICE=**
Identifies the physical device type used for storage of this data base. You must use one of the following:

FBA, 3380, 3375, 3350, 3340, 3330, or 2314.

**BLOCK=**
Specifies the VSAM control interval size. The size must be from 512 to 8192 bytes in steps of 512 bytes, or from 8192 to 30720 bytes in steps of 2048 bytes. This operand is optional; if omitted, DL/I will determine the size during DBD generation.

**SCAN=**
Specifies the number of cylinders (for count-key-data devices) or blocks (for FBA devices) to be scanned in both directions when searching for available storage space during segment insertions. "Cylinders" may be specified as any integer from 0 to 255. If you specify "blocks," it can be any integer from 0 to 32767.

The SCAN operand is optional. If omitted for count-key-data devices, the default is 3 cylinders. If omitted for FBA devices, a default is calculated during DBD generation that is approximately equal to three cylinders.

**Note:** If SCAN=0 is specified, only the current cylinder for COUNT-KEY-DATA devices is scanned. For FBA devices the number of blocks that is equal to one logical cylinder is scanned. Scanning is performed in both directions from the current position. If space cannot be found for segment insertion within the bounds defined by the operand, space at the current 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.

The first parameter specifies that every *nth* block is to be left free. This is a number from 0 to 100, excluding 1. Zero is the default and is recommended for HD randomized data bases.

The second parameter specifies the percentage of each block to be left free. This is a number from 0 to 99. Zero is the default.

Either or both parameters may be specified to achieve any combination of free and/or partially free blocks.

***Example:*** A specification of FRSPC=(5, 40) would result in a data base load (or reload) in which every fifth block (5, 10, 15, etc.) would be left free and at least forty 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.

**Note:** 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.

---

---

# 2.1.2.3 ACCESS Statement

The ACCESS statement may be used to define:

- Primary randomized access,
- Primary Indexed access, and
- Secondary Indexed access.

Each of these is described separately.

Subtopics:



## 2.1.2.3.1 Primary Randomized Access

The format of the ACCESS control statement when used to define primary randomized access is:

Table 4. Primary Randomized ACCESS Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| ACCESS | SEGM= | ° ° ° ° ° ° ° ° | Name of root segment. |
| | ,RMRTN= | ° ° ° ° ° ° ° ° | Name of randomizing module. |
| | ,SEQFLD= | ° ° ° ° ° ° ° ° | Name of sequence field. |
| | ,SEQVAL= | _ _ _ _ _ _ _ _ | UNIQUE (or U) - means each value of the sequence field is unique. DUPLICATE (or D) - means duplicate values may occur. |

*Coding Example*

```
ACCESS                                          X
      SEGM=STSCCST,                             X
      RMRTN=DLZHDC10,                           X
      SEQFLD=STQCCNO,                           X
      SEQVAL=U
```

**SEGM=**
Identifies the name of the root segment for this HD organization data base.

**RMRTN=**
Identifies the phase name in the 'DL/I prod lib' or 'DL/I user lib' of the randomizing module. The phase name must be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. You can supply your own randomizing module or use one provided by DL/I.

**For More Information...**

about randomizing modules, see *DL/I DOS/VS Data Base Administration*.

**SEQFLD=**
Identifies the name of the sequence field within the root segment.

**SEQVAL=**
Indicates whether each value of the sequence field is unique or if duplicate values occur. This operand is optional. If you choose to omit it, the default is "unique."

---

---

## 2.1.2.3.2 Primary Indexed Access

The format of the ACCESS control statement when used to define primary indexed access is:

Table 5. Primary Indexed ACCESS Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| ACCESS | SEGM= | ° ° ° ° ° ° ° ° | Name of root segment. |
| | ,SEQFLD= | ° ° ° ° ° ° ° ° | Name of sequence field. |
| | ,REF= | ° ° ° ° ° ° ° | DBD name for primary index data base. |

*Coding Example*

```
ACCESS                                  X
      SEGM=STPIITM,                     X
      SEQFLD=STQIINO,                   X
      REF=STDIX1P
```

**SEGM=**
Identifies the name of the root segment for this HD organization data base.

**SEQFLD=**
Identifies the name of the sequence field within the root segment.

**REF=**
Specifies the name you want used as a symbolic VSAM data set (KSDS) name, as the DBD name for the primary INDEX data base, as the segment name in the primary index data base, and as the name of the sequence field in the primary index data base. The name must be unique; it must not duplicate any other DBD name. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

## 2.1.2.3.3 Secondary Indexed Access

The format of the ACCESS control statement when used to define secondary indexed access is:

Table 6. Secondary Indexed ACCESS Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| ACCESS | SEGM= | ° ° ° ° ° ° ° ° | Name of target segment. |
| | ,SEQFLD= | (° ° ° ° ° ° ° ° ,_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _) | Name of field(s) to be used for sequencing.(1). |
| | ,REF= | ° ° ° ° ° ° ° | DBD name for INDEX data base. |
| | ,SEQVAL= | _ _ _ _ _ _ _ _ _ | UNIQUE (or U) means each value of the sequence field is unique. DUPLICATE (or D) means duplicate values may occur. |
| | ,SEQSEG= | _ _ _ _ _ _ _ _ | Name of source segment.(2). |
| | ,SUPVAL= | _ _ _ _ _ _ _ _ _ _ _ _ | Value on which to suppress creation of index pointer segment.(3). |
| | ,SUPRTN= | _ _ _ _ _ _ _ _ | Phase name of index suppression routine; 1-8 alphameric characters.(4). |

**Notes:**

1. The SEQFLD operand requires at least one field to be identified for sequencing. Optionally, up to four additional fields may be specified. If multiple fields are specified, their names must be enclosed in parentheses.
2. The SEQSEG operand is used to identify the segment in which the sequence field(s) reside, if other than the target segment.
3. The SUPVAL operand is optional. Use it only if you want to suppress creation of indexes for particular values of the sequence field(s).
4. The SUPRTN operand is optional. It may be used to specify the name of a user-supplied index suppression routine. See the operand description for details.

*Coding Example*

```
|           ACCESS                                    X                 |
|                 SEGM=STSCCST,                       X                 |
|                 SEQFLD=STQCODN,                     X                 |
|                 REF=STXCRDN,                        X                 |
|                 SEQVAL=U,                           X                 |
|                 SEQSEG=STPCORD                                        |
|                                                                       |
```

**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 1-8 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 1-8 characters long. It is recommended that each name start with an alphabetic character; other DL/I features may require it.

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 VSAM data set (KSDS) name and as a DBD name for the secondary INDEX data base. It will also define the field name to be used in the segment search argument (SSA) 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 1-7 characters long. The first character must be alphabetic; the rest can 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. If you choose to omit it, the default is "unique."

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

**SEQSEG=**

Identifies the segment in which the sequence field(s) reside, if other than the target segment.

The sequence segment's name must be 1-8 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 may be the physical dependent of a logical child.

**SUPVAL=**

This operand is optional. It may be used to suppress 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.

The value must be a 1-byte self-defining term (X'10',C'Z',5 or B'00101101') or the words BLANK or ZERO. BLANK is equivalent to C'' or X'40'. ZERO is equivalent to X'00' or 0, but not C'0'. 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 NULLVAL=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 composes the search field is considered to be a separate packed value. For example, if the NULLVAL=X'9F' were specified in a case where the search field was composed of three 2-byte packed source fields, there would be no index entries with the search field value of X'999F999F999F' since these and only these would be suppressed.

Also, with the same NULLVAL=X'9F', if the search field were one 6-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=**

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

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 whether or not an index entry should be created.

**For More Information...**

about user-supplied exit routines, see *DL/I DOS/VS Data Base Administration*.

## 2.1.2.4 SEGM Statement

The format of the SEGM statement when used to define an HD data base is:

Table 7. SEGM Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Segment name; 1-8 characters. |
| | ,PARENT= | ((_ _ _ _ _ _ _ _<br>,_ _ _ _)) | Name of this segment's parent.(1).<br>SNGL - for a PCF pointer.<br>DBLE - for both PCF and PCL pointers. |
| | ,BYTES= | _ _ _ _<br>or<br>(_ _ _ _ , _ _ _ _) | Fixed-length segment size.(2).<br>or<br>Variable-length segment size (maximum, minimum lengths). |
| | ,POINTER= | _ _ _ _ _ _ _ | TWIN for a PTF pointer.<br>TWINBWD for both PTF and PTB pointers.<br>NOTWIN for no pointers, but to ensure only one segment occurrence per parent. |
| | ,RULES= | (,_ _ _ _ _) | FIRST, LAST, or HERE. |
| | ,COMPRTN= | (_ _ _ _ _ _ _ _<br>,_ _ _ _<br>,_ _ _ _) | Name of segment compression routine.(3).<br>DATA (or D) for upward source compatibility to IMS/VS.<br>INIT indicates that initialization and termination processing control is required. |

**Notes:**

1. For a root segment the PARENT operand may be omitted or else PARENT=0 may be specified. For a dependent segment the first parameter is required, but the second parameter is optional. If you omit it, SNGL is used by default. If both parameters of the PARENT operand are specified, note the double parentheses which must be coded. Do not code the double parentheses, however, if only the first parameter is specified.
2. The BYTES operand is optional. If omitted, DL/I assumes the segment is a fixed-length segment and calculates its length during the DBDGEN procedure.
3. The COMPRTN operand is optional. It is used to select the segment compression option for variable-length segments. See the operand description for details.

*Coding Example.*

```
SEGM                                          X
      NAME=STSCCST,                           X
      PARENT=0,                               X
      BYTES=106,                              X
      POINTER=TWIN
SEGM                                          X
      NAME=STSCHIS,                           X
      PARENT=STSCCST,                         X
      BYTES=(130,53),                         X
      POINTER=TWINBWD,                        X
      COMPRTN=DLZSAMCP
SEGM                                          X
      NAME=STSCSTA,                           X
      PARENT=((STSCCST,SNGL)),                X
      BYTES=24,                               X
      POINTER=TWIN,                           X
      RULES=(,FIRST)
```

**NAME=**

Specifies the name of the segment being defined. The name can be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. Duplicate segment names are not allowed within a DBD.

**PARENT=**

Specifies the name of the physical parent of this segment. For the root segment, either omit this operand or, specify PARENT=0.

The second parameter controls the physical child pointer(s) in the physical parent of this segment. SNGL specifies only a *physical child first pointer* and is used in this segment's parent. DBLE specifies both a *physical child first pointer* and *physical child last pointer* are used in this segment's parent. The second parameter is optional. If omitted, SNGL is the default. If both parameters are specified, note the double parentheses which must be coded.

**For More Information...**

about physical child first (PCF) and physical child last (PCL) pointers, see *DL/I DOS/VS Data Base Administration*.

**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. Segments residing in an HD data base may be either fixed-length or variable-length.

*For fixed-length segments,* only one parameter is used. It specifies the number of bytes in the data portion of the segment.

*For variable-length segments,* two parameters are used. The first parameter specifies the maximum length of the data portion of a segment, including the 2-byte length field. The second parameter specifies the minimum length of the data portion of a segment, including the 2-byte length field. Four is the minimum specification.

The BYTES operand is optional. If it is omitted, DL/I assumes the segment to be a fixed-length segment and will calculate its length during DBD generation.

**For More Information...**

about determining segment lengths and about using variable-length segments, see *DL/I DOS/VS Data Base Administration*.

**POINTER=**
(or its abbreviation PTR) specifies the pointer fields to be reserved in the segment's prefix area. These pointers are used to relate this segment to its physical twin segments.

**TWIN (or T)**
Reserves a 4-byte field in this segment's prefix for a physical twin forward (PTF) pointer.

**TWINBWD (or TB)**
Reserves two 4-byte fields in this segment's prefix for both a physical twin forward (PTF) pointer and physical twin backward (PTB) pointer. For performance reasons, always specify POINTER=TWINBWD for root segments of indexed data bases.

**NOTWIN (or NT)**
Ensures that no more than one occurrence of this segment type will exist under its parent. If coded for the root segment, only one root segment can exist in this data base.

The POINTER operand is optional. If omitted, the default is TWIN.

**For More Information...**

about physical twin forward (PTF), and physical twin backward (PTB) pointer, and the NOTWIN parameter, see *DL/I DOS/VS Data Base Administration*.

**RULES=**
This parameter of the RULES 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 (or F)**
States that a new occurrence is to be inserted before the first existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted before all existing occurrences of the same sequence field value.

**LAST (or L)**
States that a new occurrence is to be inserted after the last existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted after all existing occurrences of the same sequence field value. This is the default option.

**HERE (or H )**
Assumes the user has determined positioning by a previous DL/I call, and the new occurrence is inserted before the segment that satisfied the last call. If the segment has a non-unique 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.

**COMPRTN=**

This operand is used to select the segment compression option. It is supported only for variable-length segments.

The first parameter specifies the name of a user-supplied routine used to compress this segment. The name must be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. The phase name (type PHASE) should be unique.

The second parameter maintains upward source compatibility to IMS/VS. Either specify DATA (or D) or omit it; DATA (or D) is the default value.

The third parameter is optional. If INIT is specified, it indicates that initialization and termination processing control is required by the segment compression routine.

**For More Information...**

about segment compression routines, see *DL/I DOS/VS Data Base Administration*.

---

# 2.1.2.5 FIELD Statement

The format of the FIELD statement when used to define an HD data base is:

Table 8. FIELD Statement Syntax for HD Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (° ° ° ° ° ° ° °<br>,_ _ _<br>,_) | Field name; 1-8 characters.<br>SEQ - identifies this as a sequence field.(1).<br>U - only unique values of this sequence field allowed.<br>M - duplicate values of this sequence field can occur in multiple occurrences of the segment. |
| | ,BYTES= | _ _ _ | Field length; 1-256 bytes.(2). |
| | ,START= | _ _ _ _ _ _ _ _ | Field starting position.(3). |
| | ,TYPE= | _ | Field data type.(4). |

**Notes:**

1. The parameters SEQ and U or M are optional. Specify them only if you want to identify this field as a sequence field in a dependent segment type.
2. See "Rules and Restrictions" in the BYTES operand description.
3. The START operand is optional. If omitted, DL/I will determine the field's starting position during DBD generation.
4. The TYPE operand is optional. If omitted, TYPE=C is assumed.

*Coding Examples*

```
        FIELD                                   X
              NAME=STKCIIN,                     X
              BYTES=6,                          X
              TYPE=C
        FIELD                                   X
              NAME=(STQCILI,SEQ,U),             X
              BYTES=2,                          X
              START=7,                          X
              TYPE=C
```

**NAME=**

The first parameter specifies the name of the field being defined within a segment type. The name specified can be referred to by an application program in a DL/I call SSA. Duplicate field names must

not be defined for the same segment type. The field name can be 1-8 alphameric characters long. It is recommended that you start the field name with an alphabetic character.

The presence of the word SEQ as a second parameter of this operand identifies this field as a sequence field in a dependent segment type. As a general rule, a segment can have only one sequence field.

**Note:** 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 operand in the SEGM statement.

The last parameter of this operand is optionally used in conjunction with the SEQ parameter. 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 or default.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 256. If this field is the sequence field in the root segment of a primary indexed data base, however, the length cannot exceed 236 bytes.

***Rules and Restrictions . . .***

- The BYTES operand must be specified for field data types X, P, C, and Z (see TYPE operand for field data types).
- The BYTES operand 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 operand for field data types E, D, and 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. Overlapping fields are permitted. If an overlapping field starts in the same position as a previously defined field, you may specify the name of the previously defined field, instead of a numeric value, to indicate the starting position (START=*fieldname*). Each field must not extend beyond the defined segment length (start position plus byte value).

The START operand is optional. if omitted, 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 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.

**Notes:**

1. All DL/I calls perform field comparisons on a byte-by-byte binary basis. No check is made by DL/I to ensure that the data contained within a field is of the type specified by this operand, except when the defined field is indexed, or converted by the Field Level Sensitivity feature.
2. It is recommended that you explicitly specify the field 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 during PSB generation.

---

---

# 2.1.2.6 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. They must be included at the end of your DBD generation input stream.

Table 9. DBDGEN, FINISH, END Statements for HD Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

 *© Copyright IBM Corp. 1981, 1989*

# 2.1.3 Control Statements Summary

Table 10 summarizes the DL/I control statements used to define HD data bases. Only those control statements and operands pertinent to HD are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 10. Summary of Control Statements Used to Code HD DBDs

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS=HD<br>[,CIANPT=root-anch-points]<br>[,PRIMCI={prime-ci-value}]<br>{*OVERLAP* }<br>[,RILIM=record-insert-limit]<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DEVICE=device-type<br>,DD1=filename<br>[,BLOCK=(control-interval-size)]<br>[,SCAN={cylinder}]<br>{blocks }<br>{*3* }<br>[,FRSPC=({free-blocks,percentage})]<br>{*0* ,*0* } |
| | ACCESS(1). | SEGM=root-seq-name<br>,RMRTN=randomizing-rtn-name<br>,SEQFLD=sequencing-fld-name<br>[,SEQVAL={*UNIQUE* }]<br>{U }<br>{DUPLICATE}<br>{D } |
| | ACCESS(2). | SEGM=root-seg-name<br>,SEQFLD=sequencing-fld-name<br>,REF=index-dbd-name |
| | ACCESS(3). | SEGM=target-seg-name<br>,SEQFLD={fldname }<br>{(fldname1,fldname2[,...,fldname5])}<br>,REF=index-dbd-name<br>[,SEQVAL={*UNIQUE* }]<br>{U }<br>{DUPLICATE}<br>{D }<br>[,SEQSEG=source-seg-name]<br>[,SUPVAL=value]<br>[,SUPRTN=rtn-name] |
| | SEGM | NAME=segment-name<br>[,PARENT={*0* }]<br>{((parent-name[{,*SNGL*}]))}<br>{,DBLE}<br>[,BYTES={fixed-length }]<br>{(max-var-length,min-var-length)}<br>[,POINTER={*TWIN* }]<br>{TWINBWD}<br>{NOTWIN }<br>[,RULES=({,FIRST})] |

| | | |
|---|---|---|
| | | {,*LAST* }<br>{,HERE }<br>[,COMPRTN=(routine-name[,*D*,INIT])] |
| | FIELD | Name=(field-name[,SEQ[{,*U*}]])<br>{,M}<br>[BYTES=length]<br>[,START=starting-position]<br>[,TYPE=data-type] |
| | DBDGEN | |
| | FINISH | |
| | END | |

(1).Primary randomized access.
(2).Primary indexed access.
(3).Secondary indexed access.

---

# 2.1.4 Examples of HD DBD Generations

The rest of this chapter shows examples of HD data bases and the control statements required to define them. The examples shown are as follows:

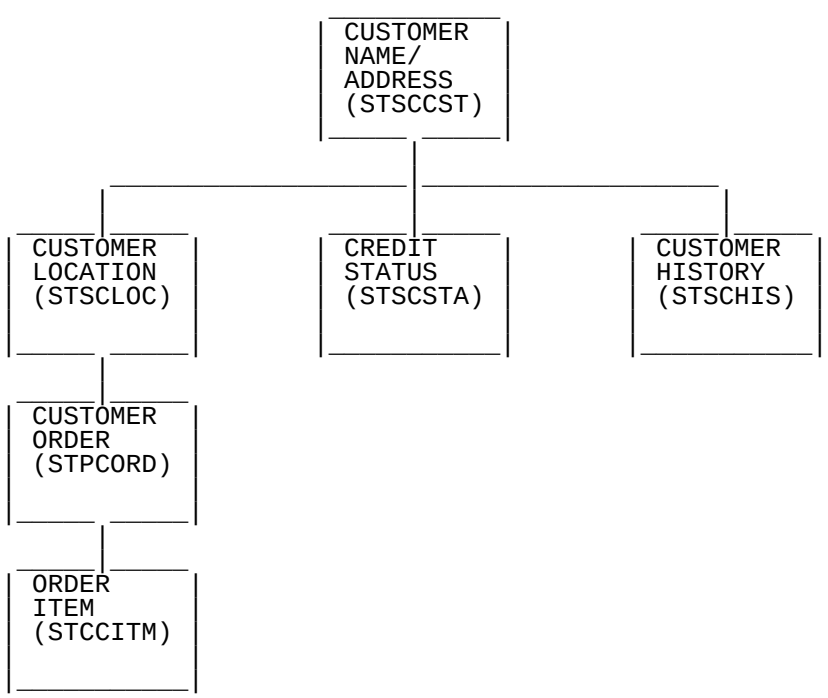
Figure 2 shows the physical DBD for the Customer data base. In this example, the ACCESS statement defines *primary randomized access*.

Figure 3 shows the physical DBD for the Inventory data base. Like Figure 2 this example also shows the ACCESS statement defining *primary randomized access.*

Figure 4 is another example of the physical DBD for the Inventory data base. The difference between this example and the one in Figure 3 is that this one shows an ACCESS statement defining *primary indexed access.*

Figure 5 shows how the physical DBD for the Customer data base is extended for secondary access. The figure shows an ACCESS statement defining *secondary indexed access.*

Customer Data Base
(DBD Name - STDCDBP)




```
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
```

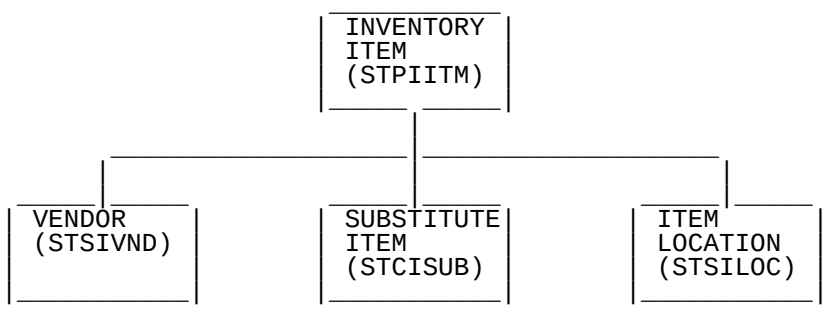
```
      DD1=STDCDBC,            DLBL FILE NAME                  X
      DEVICE=3380,            DISK DEVICE                     X
      BLOCK=2048,             VSAM CONTROL INTERVAL SIZE      X
      SCAN=2                  CYLINDERS SCAN FOR ISRT SPACE
ACCESS                                                        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,SEQ,U),   UNIQUE KEY FIELD (CUST #)       X
      BYTES=6,                FIELD LENGTH                    X
      START=1,                WHERE IT STARTS IN SEGMENT      X
      TYPE=C                  ALPHAMERIC DATA




SEGM                                                          X
      NAME=STSCLOC,           SEGMENT NAME CUSTOMER LOCATION  X
      PARENT=STSCCST,         PARENT IS CUST NAME/ADDR SEGM   X
      BYTES=106,              FIELD LENGTH                    X
      POINTER=TWINBWD         BOTH PHYS TWIN FWD AND BWD
FIELD                                                         X
      NAME=(STQCLNO,SEQ,U),   UNIQUE KEY FIELD (LOCATION #)   X
      BYTES=6,                FIELD LENGTH                    X
      START=1,                WHERE IT STARTS IN SEGMENT      X
      TYPE=C                  ALPHAMERIC DATA
SEGM                                                          X
      NAME=STPCORD,           SEGMENT NAME CUSTOMER ORDER     X
      PARENT=STSCLOC,         PARENT IS CUST 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
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
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
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 FIRST
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
```

Figure 2. Example of HD DBD (Defined with Primary Randomized Access) for Customer Data Base

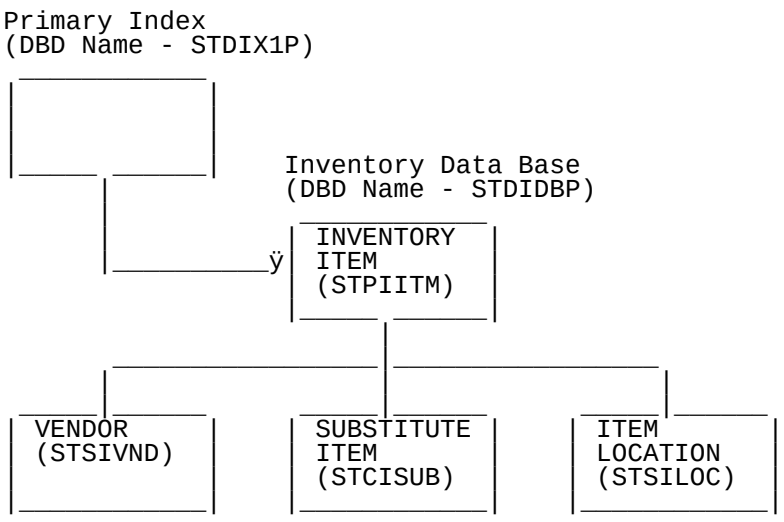
Inventory Data Base
(DBD Name - STDIDBP)




```
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=3380,             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,SEQ,U),    UNIQUE KEY FIELD (ITEM #)     X
      BYTES=6,                 FIELD LENGTH                  X
      START=1,                 WHERE IT STARTS IN SEGMENT    X
      TYPE=C                   ALPHAMERIC
SEGM                                                         X
      NAME=STSIVND,            AUTHORIZED VENDOR INFORMATION X
      PARENT=STPIITM,          PARENT IS INVENTORY ITEM SEGM X
      BYTES=106,               FIELD LENGTH                  X
      POINTER=TWIN             PHYSICAL TWIN FWD ONLY
FIELD                                                        X
      NAME=(STQVVNO,SEQ,U),    UNIQUE KEY FIELD (VENDOR #)   X
      BYTES=6,                 FIELD LENGTH                  X
      START=1,                 WHERE IT STARTS IN SEGMENT    X
      TYPE=C                   ALPHAMERIC DATA
SEGM                                                         X
      NAME=STCISUB,            SEG NAME FOR SUB-ITEM INFO    X
      PARENT=STPIITM,          PARENT IS INVENTORY ITEM SEGM X
      BYTES=56,                DATA LENGTH                   X
      POINTER=TWINBWD          BOTH PHYS TWIN FWD AND BWD
FIELD                                                        X
      NAME=(STQCCNO,SEQ,U),    UNIQUE KEY FIELD (SUB-ITEM #) X
      BYTES=6,                 FIELD LENGTH                  X
      START=1,                 WHERE IT STARTS IN SEGMENT    X
      TYPE=C                   ALPHAMERIC DATA


SEGM                                                         X
      NAME=STSILOC,            SEGMENT NAME INVENTORY LOCATION X
      PARENT=STPIITM,          PARENT IS INVENTORY ITEM SEGM X
      BYTES=12,                DATA LENGTH                   X
      POINTER=TWINBWD          BOTH PHYS TWIN FWD AND BWD
FIELD                                                        X
      NAME=(STQILNO,SEQ,U),    UNIQUE KEY FIELD INVENTORY LOC# X
      BYTES=6,                 FIELD LENGH                   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
```

Figure 3. Example of HD DBD (Defined with Primary Randomized Access) for Inventory Data Base

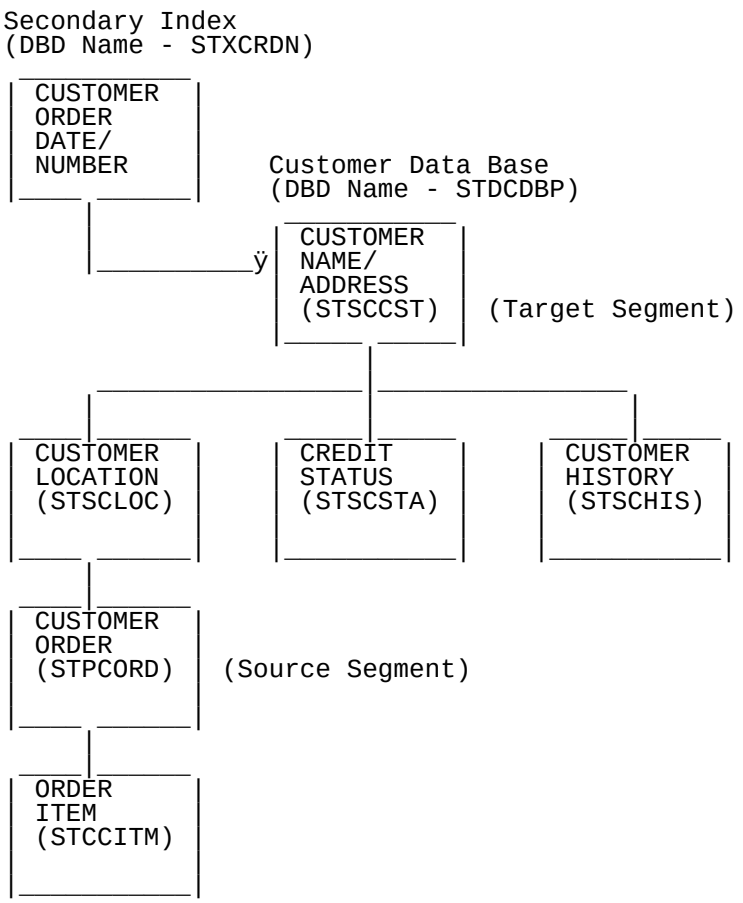



```
DBD                                                          X
      NAME=STDIDBP,            DATA BASE DESCRIPTION NAME    X
      ACCESS=HD                HIERARCHICAL DIRECT ACCESS
DATASET                                                      X
      DD1=STDIDBC,             DLBL FILE NAME                X
      DEVICE=3380,             DISK DEVICE                   X
      BLOCK=2048,              VSAM CONTROL INTERVAL SIZE    X
      SCAN=2                   # CYLINDERS SCAN FOR ISRT SPACE
ACCESS                                                       X
      SEGM=STPIITM,            ROOT SEGMENT NAME             X
      SEQFLD=STQIINO,          SEQUENCE FIELD                X
      REF=STDIX1P              PRIMARY INDEX NAME
SEGM                                                         X
      NAME=STPIITM,            SEGMENT NAME INVENTORY ITEM   X
      PARENT=0,                IT IS A ROOT SEGMENT          X
      BYTES=56,                DATA LENGTH                   X
      POINTER=TWIN             PHYSICAL TWIN FWD ONLY
FIELD                                                        X
      NAME=(STQIINO,SEQ,U),    UNIQUE KEY FIELD (ITEM #)     X
      BYTES=6,                 FIELD LENGTH                  X
      START=1,                 WHERE IT STARTS IN SEGMENT    X
      TYPE=C                   ALPHAMERIC
SEGM                                                         X
      NAME=STSIVND,            AUTHORIZED VENDOR INFORMATION X
      PARENT=STPIITM,          PARENT IS INVENTORY ITEM SEGM X
      BYTES=106,               FIELD LENGTH                  X
      POINTER=TWIN             PHYSICAL TWIN FWD ONLY
FIELD                                                        X
      NAME=(STQVVNO,SEQ,U),    UNIQUE KEY FIELD (VENDOR #)   X
      BYTES=6,                 FIELD LENGTH                  X
      START=1,                 WHERE IT STARTS IN SEGMENT    X
      TYPE=C                   ALPHAMERIC DATA
SEGM                                                         X
      NAME=STCISUB,            SEG NAME FOR SUB-ITEM INFO    X
      PARENT=STPIITM,          PARENT IS INVENTORY ITEM SEGM X
      BYTES=56,                DATA LENGTH                   X
      POINTER=TWINBWD          BOTH PHYS TWIN FWD AND BWD


FIELD                                                        X
      NAME=(STQCCNO,SEQ,U),    UNIQUE KEY FIELD (SUB-ITEM #) X
      BYTES=6,                 FIELD LENGTH                  X
      START=1,                 WHERE IT STARTS IN SEGMENT    X
      TYPE=C                   ALPHAMERIC DATA
SEGM                                                         X
      NAME=STSILOC,            SEGMENT NAME INVENTORY LOCATION X
      PARENT=STPIITM,          PARENT IS INVENTORY ITEM SEGM X
      BYTES=12,                DATA LENGTH                   X
      POINTER=TWINBWD          BOTH PHYS TWIN FWD AND BWD
FIELD                                                        X
```

```
            NAME=(STQILNO,SEQ,U),    UNIQUE KEY FIELD INVENTORY LOC# X
            BYTES=6,                 FIELD LENGH                     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
```

Figure 4. Example of HD DBD (Defined with Primary Indexed Access) for Inventory Data Base




```
      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=3380,             DISK DEVICE                 X
            BLOCK=2048,              VSAM CONTROL INTERVAL SIZE  X
            SCAN=2                   CYLINDERS SCAN FOR ISRT SPACE
      ACCESS                                                     X
            SEGM=STSCCST,            ROOT SEGMENT                X
            SEQFLD=STQCCNO,          SEQUENCE FIELD              X
            SEQVAL=U,                UNIQUE VALUE                X
            RMRTN=DLZHDC10           RANDOMIZING ROUTINE
      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                                                       X
            NAME=STSCCST,            SEGMENT NAME FOR CUST NAME/ADDR X
```

```
      PARENT=0,               IT IS A ROOT SEGMENT           X
      BYTES=106,              DATA LENGTH                    X
      POINTER=TWIN            PHYSICAL TWIN FWD ONLY


FIELD                                                        X
      NAME=(STQCCNO,SEQ,U),   UNIQUE KEY FIELD (CUST #)      X
      BYTES=6,                FIELD LENGTH                   X
      START=1,                WHERE IT STARTS IN SEGMENT     X
      TYPE=C                  ALPHAMERIC DATA
SEGM                                                         X
      NAME=STSCLOC,           SEGMENT NAME CUSTOMER LOCATION X
      PARENT=STSCCST,         PARENT IS CUST NAME/ADDR SEGM  X
      BYTES=106,              FIELD LENGTH                   X
      POINTER=TWINBWD         BOTH PHYS TWIN FWD AND BWD
FIELD                                                        X
      NAME=(STQCLNO,SEQ,U),   UNIQUE KEY FIELD (LOCATION #)  X
      BYTES=6,                FIELD LENGTH                   X
      START=1,                WHERE IT STARTS IN SEGMENT     X
      TYPE=C                  ALPHAMERIC DATA
SEGM                                                         X
      NAME=STPCORD,           SEGMENT NAME CUSTOMER ORDER    X
      PARENT=STSCLOC,         PARENT IS CUST 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
SEGM                                                         X
      NAME=STCCITM,           SEGMENT NAME LINE ITEM         X
      PARENT=STPCORD,         PHYSICAL PARENT IS CUSTOMER ORDX
      BYTES=38,               DATA LENGTH                    X
      POINTER=TWINBWD         BOTH PHYS TWIN FWD AND BWD
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
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 FIRST
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
```

Figure 5. Example of HD DBD (Defined with Secondary Indexed Access) for Customer Data Base

---

# 2.2 Chapter 2. Defining HDAM Data Bases

This chapter describes how to define HDAM data bases. Only those requirements essential for the definition of a "basic" HDAM DBD are given. If you also want DL/I's logical relationship function and/or DL/I's secondary indexing function, you must extend the basic DBD described here by defining the additional requirements described in Chapter 4, "Defining Logical Relationships in HD, HDAM, and HIDAM" and Chapter 5, "Defining Secondary Indexing in HDAM and HIDAM" respectively.

***To define a physical data base for HDAM,*** you must prepare an input stream of several DL/I control statements specifying:

- The name of the randomizing module used to manage data stored in the data base.
- The symbolic VSAM data set (ESDS) name for this HDAM data base.
- The segment types for the data base.
- The use of physical child and physical twin pointers in the data base.
- The fields for each segment type, one of which identifies a sequence field in the root segment.

Subtopics:



---

---

## 2.2.1 Input Structure and Rules

Figure 6 illustrates the rules for structuring a batch input stream to define HDAM data bases. The figure shows the types of control statements required, the number of each type that may be specified, and their specific order of sequence. A separate input stream is required for each data base you define.

***To Define an HDAM Data Base . . .***

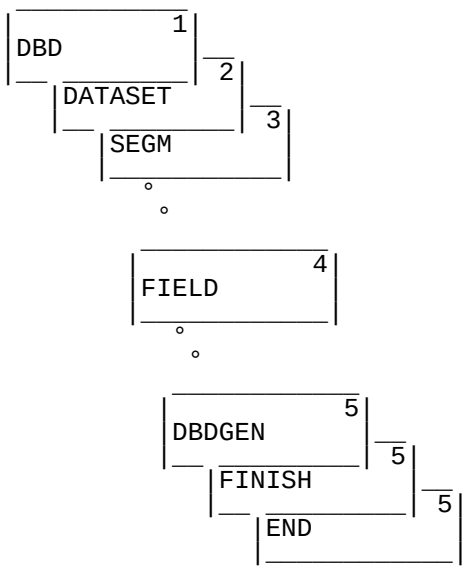



Figure 6. Batch Input Stream for HDAM Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.
2. Code one DATASET statement. It must follow the DBD statement.
3. Code one SEGM statement for each segment type in your data base. A minimum of one SEGM is required. The maximum allowed per data base is 255. Place the SEGM statements after the DATASET statement. Keep them in the same sequence as the segments appear in the hierarchic order of the data base being defined.
4. Code at least one FIELD statement. Use it to identify a "sequence field" within the root segment. Optionally, you may define a maximum of 255 fields per segment type and 1020 fields per data base. Use one FIELD statement for each field you define. Place them after the SEGM statement defining the segment that contains those fields.
5. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

# 2.2.2 Control Statements Format

This section explains how to code DL/I control statements used to define HDAM data bases. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introduction.

Subtopics:



## 2.2.2.1 DBD Statement

The format of the DBD statement when used to define an HDAM data base is:

Table 11. DBD Statement Syntax for HDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Data base name; 1-7 alphameric characters; the first character must be alphabetic. |
| | ,ACCESS= | **H D A M** | HDAM for Hierarchical Direct Access Method. |
| | ,RMNAME= | (° ° ° ° ° ° ° ° ,_ _ _ ,_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _ _) | Randomizing module name. Number of root anchor points in each block.(1) Control intervals in root-addressable area.(2) Record insert limit.(3) |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS. NO = Format compatibility not required. |

**Notes:**

1. This parameter is optional. If omitted, the default is 1 root anchor point.
2. This parameter is optional. If omitted, no upper limit check is performed on the rbn created by the randomizing module.
3. This parameter is optional; it can be coded only if the previous parameter has been coded. If omitted, no limit is placed on the record size.

*Coding Example*

```
       DBD                                                  X
             NAME=STDCDBP,                                  X
             ACCESS=HDAM,                                   X
             RMNAME=(DLZHDC10,                              X
             3,                                             X
             100,                                           X
             600),                                          X
             IMSCOMP=NO
```

**NAME=**
Specifies the name of the data base. The name must be 1-7 characters

long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Specifies the access method used for the data base. Here you must specify HDAM.

**RMNAME=**
Specifies information used to manage data stored in the associated HDAM data base.

*This first parameter* is required. It specifies the name of a randomizing module used for storing and accessing segments contained in this data base. The name must be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. DL/I provides several general randomizing modules (DLZHDC10, DLZHDC20, DLZHDC30, and DLZHDC40) that you can use, or you can provide your own randomizing routine.

**For more information...**

about randomizing routines, see *DL/I DOS/VS Data Base Administration*.

*The second parameter* specifies the number of root anchor points desired in each control interval or block in the root addressable area of a HDAM data base. The number you specify must not exceed 255 or be less than 1. Typical values are from 1 to 5. If you omit this parameter, the default is 1.

*The third parameter* specifies the maximum relative block number value that the user wishes to allow a randomizing module to produce for this data base. This value determines the number of control intervals or blocks in the root addressable area of an HDAM data base. This parameter must not exceed 16,777,215 or be less than 1. This parameter is optional. If omitted, no upper limit check is performed on the relative block number created by the randomizing module.

**Note:** You must specify this parameter if you use one of the randomizing modules supplied by DL/I. If you omit it, unpredictable results will occur in the module during load execution.

*The last parameter* specifies the maximum number of bytes of a data base record that can be stored into the root addressable area in a series of inserts unbroken by a call to another data base record. The value for this parameter must not exceed 16,777,215 or be less than 1. This parameter is optional. If 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.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

**Note:** Changing the IMSCOMP= entry for an existing data base without unloading and reloading the data base may cause unpredictable results, due to a different record size used by DL/I internally in the VSAM CI.



---

## 2.2.2.2 DATASET Statement

The format of the DATASET statement when used to define an HDAM data base is:

Table 12. DATASET Statement Syntax for HDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Name of data set; 1-7 alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | _ _ _ _ _ | VSAM control interval size; (512-8192 bytes in steps of 512 bytes; 8192-30720 bytes in steps of 2048 bytes). |
| | ,SCAN= | _ _ _ _ _ | Number of cylinders (or blocks if FBA) to be scanned when searching for space during segment insertions.(2) |
| | ,FRSPC= | (_ _ _<br><br>,_ _) | Leave every nth block free (0-100 excluding 1). 0 is recommended for HDAM data bases.<br><br>Leave this percent free in every block (0-99).(3) |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate a value during DBD generation.
2. The SCAN operand is optional. See the operand description for details.
3. The FRSPC operand is optional. If omitted, the default is 0.

*Coding Example*

```
DATASET                                          X
      DD1=STDCDBC,                               X
      DEVICE=3380,                               X
      BLOCK=2048,                                X
      SCAN=2
```

**DD1=**

Specifies the symbolic VSAM data set (ESDS) name for this HDAM data base. The name must be 1-7 alphameric characters long.

**DEVICE=**
Identifies the physical device type used for storage of this data base. You must use one of the following:

FBA, 3380, 3375, 3350, 3340, 3330, or 2314.

**BLOCK=**
Specifies the VSAM control interval size. The size must be from 512-8192 bytes in steps of 512 bytes or from 8192-30720 bytes in steps of 2048 bytes. This operand is optional. If omitted, DL/I will determine the size during DBD generation.

**SCAN=**
Specifies the number of cylinders (for count-key-data devices) or blocks (for FBA devices) to be scanned in both directions when searching for available storage space during segment insertions. "Cylinders" may be specified as any integer from 0 to 255. If you specify "blocks," it can be any integer from 0 to 32767.

The SCAN operand is optional. If omitted for count-key-data devices, the default is 3 cylinders. If omitted for FBA devices, a default is calculated during DBD generation that is approximately equal to three cylinders.

**Note:** If SCAN=0 is specified, only the current cylinder for COUNT-KEY-DATA devices is scanned. For FBA devices the number of blocks that is equal to one logical cylinder is scanned. Scanning is performed in both directions from the current position. If space cannot be found for segment insertion within the bounds defined by the operand, space at the current 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.

The first parameter specifies that every *nth* block is to be left free. This is a number from 0 to 100, excluding 1. Zero is the default and is recommended for HDAM data bases.

The second parameter specifies the percentage of each block to be left free. This is a number from 0 to 99. Zero is the default.

Either or both parameters may be specified to achieve any combination of free and/or partially free blocks.

***Example:*** A specification of FRSPC=(5, 40) would result in a data base load (or reload) in which every fifth block (5, 10, 15, etc.) would be left free and at least forty 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.

**Note:** 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.

---



---

## 2.2.2.3 SEGM Statement

The format of the SEGM statement when used to define an HDAM data base is:

Table 13. SEGM Statement Syntax for HDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Segment name; 1-8 characters. |
| | ,PARENT= | ((_ _ _ _ _ _ _ _<br>,_ _ _ _)) | Name of this segment's parent.(1)<br>SNGL - for a PCF pointer.<br>DBLE - for both PCF and PCL pointers. |
| | ,BYTES= | _ _ _ _<br>or<br>(_ _ _ _ , _ _ _ _) | Fixed-length segment size.(2)<br>or<br>Variable-length segment size (maximum, minimum lengths). |
| | ,POINTER= | _ _ _ _ _ _ _ | TWIN for a PTF pointer.<br>TWINBWD for both PTF and PTB pointers.<br>NOTWIN for no pointers, but to ensure only one segment occurrence per parent. |
| | ,RULES= | (,_ _ _ _ _) | FIRST, LAST, or HERE. |
| | ,COMPRTN= | (_ _ _ _ _ _ _ _<br>,_ _ _ _<br>,_ _ _ _) | Name of segment compression routine.(3)<br>DATA (or D) for upward source compatibility to IMS/VS.<br>INIT indicates that initialization and termination processing control is required. |

**Notes:**

1. For a root segment the PARENT operand may be omitted or else PARENT=0 may be specified. For a dependent segment the first parameter is required, but the second parameter is optional. If you omit it, SNGL is used by default. If both parameters of the PARENT operand are specified, note the double parentheses which must be coded. Do not code the double parentheses, however, if only the first parameter is specified.
2. The BYTES operand is optional. If omitted, DL/I assumes the segment is a fixed-length segment and calculates its length during the DBDGEN procedure.
3. The COMPRTN operand is optional. It is used to select the segment compression option for variable-length segments. See the operand description for details.

*Coding Examples*

```
SEGM                                          X
      NAME=STSCCST,                           X
      PARENT=0,                               X
      BYTES=106,                              X
      POINTER=TWIN
SEGM                                          X
      NAME=STSCHIS,                           X
      PARENT=STSCCST,                         X
      BYTES=(130,53),                         X
      POINTER=TWINBWD,                        X
      COMPRTN=DLZSAMCP
SEGM                                          X
      NAME=STSCSTA,                           X
      PARENT=((STSCCST,SNGL)),                X
      BYTES=24,                               X
      POINTER=TWIN,                           X
      RULES=(,FIRST)
```

**NAME=**

Specifies the name of the segment being defined. The name can be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. Duplicate segment names are not allowed within a DBD.

**PARENT=**

Specifies the name of the physical parent of this segment. For the root segment, either omit this operand or, specify PARENT=0.

The second parameter controls the physical child pointer(s) in the physical parent of this segment. SNGL specifies only a *physical child first pointer* and is used in this segment's parent. DBLE specifies both a *physical child first pointer* and *physical child last pointer* are used in this segment's parent. The second parameter is optional. If omitted, SNGL is the default. If both parameters are specified, note the double parentheses which must be coded.

**For More Information...**

about physical child first (PCF) and physical child last (PCL) pointers, see *DL/I DOS/VS Data Base Administration*.

**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. Segments residing in an HDAM data base may be either fixed-length or variable-length.

*For fixed-length segments,* only one parameter is used. It specifies the number of bytes in the data portion of the segment.

*For variable-length segments,* two parameters are used. The first parameter specifies the maximum length of the data portion of a segment, including the 2-byte length field. The second parameter specifies the minimum length of the data portion of a segment, including the 2-byte length field. Four is the minimum specification.

The BYTES operand is optional. If it is omitted, DL/I assumes the segment to be a fixed-length segment and will calculate its length during DBD generation.

**For More Information...**

about determining segment lengths and about using variable-length segments, see *DL/I DOS/VS Data Base Administration*.

**POINTER= (or its abbreviation PTR)**

Specifies the pointer fields to be reserved in the segment's prefix area. These pointers are used to relate this segment to its physical twin segments.

**TWIN (or T)**

Reserves a 4-byte field in this segment's prefix for a physical twin forward (PTF) pointer.

**TWINBWD (or TB)**

Reserves two 4-byte fields in this segment's prefix for both a physical twin forward (PTF) pointer and physical twin backward (PTB) pointer.

**NOTWIN (or NT)**

Ensures that no more than one occurrence of this segment type will exist under its parent. If coded for the root segment, only one root segment can exist in this data base.

The POINTER operand is optional. If omitted, the default is TWIN.

**For More Information...**

about physical twin forward (PTF), and physical twin backward (PTB) pointer, and the NOTWIN parameter, see *DL/I DOS/VS Data Base Administration*.

**RULES=**

This parameter of the RULES 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 (or F)**

States that a new occurrence is to be inserted before the first existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted before all existing occurrences of the same sequence field value.

**LAST (or L)**

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

**HERE (or H)**

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

**COMPRTN=**

This operand is used to select the segment compression option. It is supported only for variable-length segments.

The first parameter specifies the name of a user-supplied routine used to compress this segment. The name must be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. The phase name (type PHASE) should be unique.

The second parameter maintains upward source compatibility to IMS/VS. DATA (or D) is the default value if omitted.

The third parameter is optional. If INIT is specified, it indicates that initialization and termination processing control is required by the segment compression routine.

**For More Information...**

about segment compression routines, see *DL/I DOS/VS Data Base Administration*.

# 2.2.2.4 FIELD Statement

The format of the FIELD statement when used to define an HDAM data base is:

Table 14. FIELD Statement Syntax for HDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (° ° ° ° ° ° ° °<br>,_ _ _<br>,_) | Field name; 1-8 characters.<br>SEQ - identifies this as a sequence field.(1)<br>U - only unique values of this sequence field allowed.<br>M - duplicate values of this sequence field can occur in multiple occurrences of the segment. |
| | ,BYTES= | _ _ _ | Field length; 1-256 bytes.(2) |
| | ,START= | _ _ _ _ _ _ _ _ | Field starting position.(3) |
| | ,TYPE= | _ | Field data type.(4) |

**Notes:**

1. The parameters SEQ and U or M are optional. Specify them only if you want to identify this field as a sequence field in a dependent segment type.
2. See "Rules and Restrictions" in the BYTES operand description.
3. The START operand is optional. If omitted, DL/I will determine the field's starting position during DBD generation.
4. The TYPE operand is optional. If omitted, TYPE=C is assumed.

*Coding Examples*

```
FIELD                                          X
      NAME=STKCIIN,                            X
      BYTES=6,                                 X
      TYPE=C
FIELD                                          X
      NAME=(STQCILI,SEQ,U),                    X
      BYTES=2,                                 X
      START=7,                                 X
      TYPE=C
```

**NAME=**

The first parameter specifies the name of the field being defined within a segment type. The name specified can be referred to by an application program in a DL/I call SSA. Duplicate field names must not be defined for the same segment type. The field name can be 1-8 alphameric characters long. It is recommended that you start the field name with an alphabetic character.

The presence of the word SEQ as a second parameter of this operand identifies this field as a sequence field in a dependent segment type. As a general rule, a segment can have only one sequence field.

A sequence field must be provided for the root segment. It is optional for all dependent segment types.

**Note:** 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 operand in the SEGM statement.

The last parameter of this operand is optionally used in conjunction with the SEQ parameter. 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 or default.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 256.

***Rules and Restrictions***

- The BYTES operand must be specified for field data types X, P, C, and Z (see TYPE operand for field data types).
- The BYTES operand 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 operand for field data types E, D, and 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. Overlapping fields are permitted. If an overlapping field starts in the same position as a previously defined field, you may specify the name of the previously defined field, instead of a numeric value, to indicate the starting position (START=*fieldname*). Each field must not extend beyond the defined segment length (start position plus byte value).

The START operand is optional. if omitted, DL/I places this field adjacent to the end of the previous field, or if it is the first field in the segment, at the beginning of the segment (START=1). (Note that for concatenated segments, the beginning of the segment is the start of the destination parent concatenated key.)

**TYPE=**

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

```
    X - Hexadecimal
    H - Halfword binary
    F - Fullword binary
    P - Packed decimal
    Z - Zoned decimal
    C - Character
    E - Floating point (short)
    D - Floating point (long)
    L - Floating point (extended)
```

If this parameter is omitted, TYPE=C is assumed.

**Notes:**

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

2. It is recommended that you explicitly specify the field 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 during PSB generation.

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 2.2.2.5 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. They must be included at the end of your DBD generation input stream.

Table 15. DBDGEN, FINISH, END Statements for HDAM Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

 *© Copyright IBM Corp. 1981, 1989*

# 2.2.3 Control Statements Summary

Table 16 summarizes the DBD generation control statements used to define HDAM data bases. Only the control statements and operands pertinent to HDAM are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 16. Summary of Control Statements Used to Code HIDAM DBDs

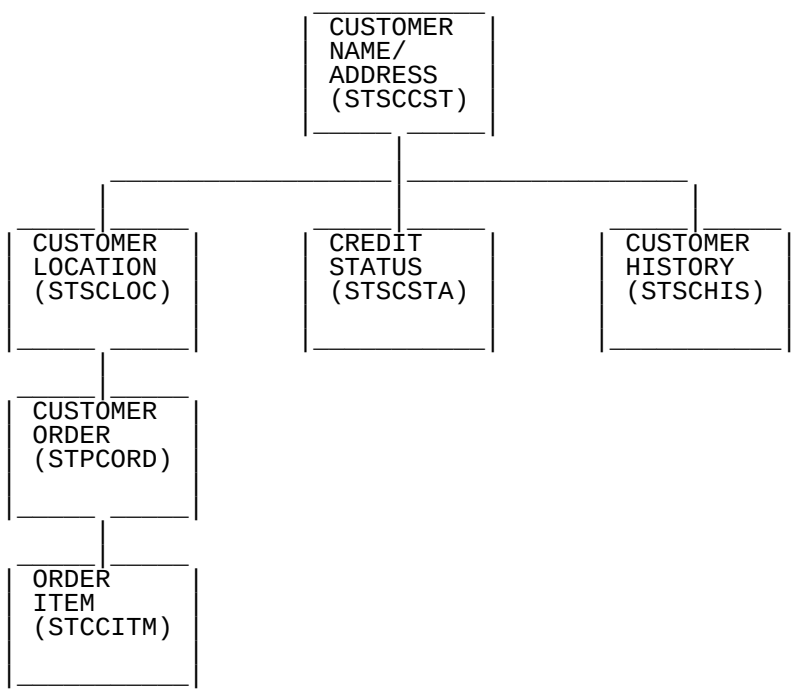
| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS=HDAM<br>,RMNAME=(module-name,{anch}[,rbn[,bytes]])<br>{*1* }<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DEVICE=device-type<br>,DD1=filename<br>[,BLOCK=(control-interval-size)]<br>[,SCAN={cylinder}]<br>{blocks }<br>{*3* }<br>[,FRSPC=({free-blocks,percentage})]<br>{*0* ,*0* } |
| | SEGM | NAME=segment-name<br>[,PARENT={*0* }]<br>{((parent-name[{,*SNGL*}]))}<br>{,DBLE}<br>[,BYTES={fixed-length }]<br>{(max-var-length,min-var-length)}<br>[,POINTER={*TWIN* }]<br>{TWINBWD}<br>{NOTWIN }<br>[,RULES=({,FIRST})]<br>{,*LAST* }<br>{,HERE }<br>[,COMPRTN=(routine-name[,*D*,INIT])] |
| | FIELD | NAME=(field-name[,SEQ[{,*U*}]])(1)<br>{,M}<br>[,BYTES=length]<br>[,START=starting-position]<br>[,TYPE=data-type] |
| | DBDGEN | |
| | FINISH | |
| | END | |

(1)SEQ must be specified for root segment.

# 2.2.4 Examples of HDAM DBD Generations

Figure 7 and Figure 8 illustrate sample HDAM data bases and the control statements used to describe them. Figure 7 is for the customer data base and Figure 8 is for the inventory data base.

Customer Data Base
(DBD Name - STDCDBP)



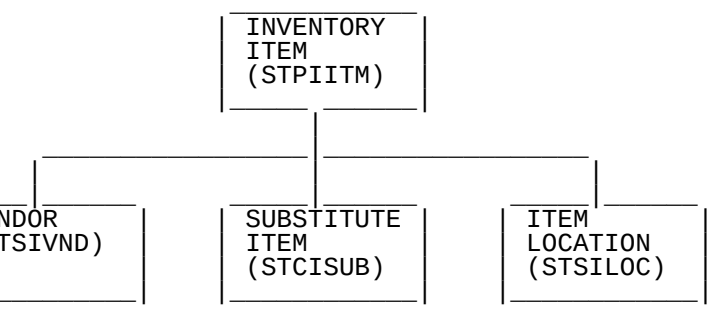

```
DBD                                                          X
      NAME=STDCDBP,           DATA BASE DESCRIPTION NAME     X
      ACCESS=HDAM,            HIERARCHICAL DIRECT            X
      RMNAME=(DLZHDC10,       RANDOMIZING ROUTINE PHASENAME  X
      3,                      ROOT ANCHOR POINTS PER BLOCK   X
      100,                    ROOT ADDR. AREA HI RELATIVE BLK X
      600)                    INSERT BYTES LIMIT FOR RAA
DATASET                                                      X
      DD1=STDCDBC,            DLBL FILE NAME                 X
      DEVICE=3380,            DISK DEVICE                    X
      BLOCK=2048,             VSAM CONTROL INTERVAL SIZE     X
      SCAN=2                  # CYLINDERS SCAN FOR ISRT SPACE
SEGM                                                         X
      NAME=STSCCST,           SEGMENT NAME FOR CUST NAME/ADDR X
      PARENT=0,               IT IS A ROOT SEGMENT           X
      BYTES=106,              DATA LENGTH                    X
      POINTER=TWIN            PHYSICAL TWIN FWD ONLY
FIELD                                                        X
      NAME=(STQCCNO,SEQ,U),   UNIQUE KEY FIELD (CUST #)      X
      BYTES=6,                FIELD LENGTH                   X
      START=1,                WHERE IT STARTS IN SEGMENT     X
      TYPE=C                  ALPHAMERIC DATA


SEGM                                                         X
```

```
      NAME=STSCLOC,           SEGMENT NAME CUSTOMER LOCATION  X
      PARENT=STSCCST,         PARENT IS CUST. NAME/ADDR SEGM  X
      BYTES=106,              FIELD LENGTH                    X
      POINTER=TWINBWD         BOTH PHYS. TWIN FWD AND BWD
FIELD                                                         X
      NAME=(STQCLNO,SEQ,U),   UNIQUE KEY FIELD (LOCATION #)   X
      BYTES=6,                FIELD LENGTH                    X
      START=1,                WHERE IT STARTS IN SEGMENT      X
      TYPE=C                  ALPHAMERIC DATA
SEGM                                                          X
      NAME=STPCORD,           SEGMENT NAME CUSTOMER ORDER     X
      PARENT=STSCLOC,         PARENT IS CUST. 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
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
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
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 FIRST
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
```

Figure 7. Example of HDAM DBD for Customer Data Base

Inventory Data Base
(DBD Name - STDIDBP)




```
      DBD                                                     X
      NAME=STDIDBP,           DATA BASE DESCRIPTION NAME      X
      ACCESS=HDAM,            HIERARCHICAL DIRECT             X
      RMNAME=(DLZHDC30,       RANDOMIZING ROUTINE PHASENAME   X
      3,                      ROOT ANCHOR POINTS PER BLOCK    X
```

```
      100,                  ROOT ADDR. AREA HI RELATIVE BLK X
      400)                  INSERT BYTES LIMIT FOR RAA
DATASET                                                     X
      DD1=STDIDBC,          DLBL FILE NAME                  X
      DEVICE=3380,          DISK DEVICE                     X
      BLOCK=2048,           VSAM CONTROL INTERVAL SIZE      X
      SCAN=2                # CYLINDERS SCAN FOR ISRT SPACE
SEGM                                                        X
      NAME=STPIITM,         SEGMENT NAME INVENTORY ITEM     X
      PARENT=0,             IT IS A ROOT SEGMENT            X
      BYTES=56,             DATA LENGTH                     X
      POINTER=TWIN          PHYSICAL TWIN FWD ONLY
FIELD                                                       X
      NAME=(STQIINO,SEQ,U), UNIQUE KEY FIELD (ITEM #)       X
      BYTES=6,              FIELD LENGTH                    X
      START=1,              WHERE IT STARTS IN SEGMENT      X
      TYPE=C                ALPHAMERIC
SEGM                                                        X
      NAME=STSIVND,         AUTHORIZED VENDOR INFORMATION   X
      PARENT=STPIITM,       PARENT IS INVENTORY ITEM SEGM.  X
      BYTES=106,            FIELD LENGTH                    X
      POINTER=TWIN          PHYSICAL TWIN FWD ONLY
FIELD                                                       X
      NAME=(STQVVNO,SEQ,U), UNIQUE KEY FIELD (VENDOR #)     X
      BYTES=6,              FIELD LENGTH                    X
      START=1,              WHERE IT STARTS IN SEGMENT      X
      TYPE=C                ALPHAMERIC DATA
SEGM                                                        X
      NAME=STCISUB,         SEG. NAME FOR SUB-ITEM INFO.    X
      PARENT=STPIITM,       PARENT IS INVENTORY ITEM SEGM   X
      BYTES=56,             DATA LENGTH                     X
      POINTER=TWINBWD       BOTH PHYS. TWIN FWD AND BWD
FIELD                                                       X
      NAME=(STQCCNO,SEQ,U), UNIQUE KEY FIELD (SUB-ITEM #)   X
      BYTES=6,              FIELD LENGTH                    X
      START=1,              WHERE IT STARTS IN SEGMENT      X
      TYPE=C                ALPHAMERIC DATA


SEGM                                                        X
      NAME=STSILOC,         SEGMENT NAME INVENTORY LOCATION X
      PARENT=STPIITM,       PARENT IS INVENTORY ITEM SEGM.  X
      BYTES=12,             DATA LENGTH                     X
      POINTER=TWINBWD       BOTH PHYS. TWIN FWD AND BWD
FIELD                                                       X
      NAME=(STQILNO,SEQ,U), UNIQUE KEY FIELD INVENTORY LOC# X
      BYTES=6,              FIELD LENGTH                    X
      START=1,              WHERE IT STARTS IN SEGMENT      X
      TYPE=C                ALPHAMERIC DATA
DBDGEN                      REQUIRED TO MARK DBD END
FINISH                      FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 8. Example of HDAM DBD for Inventory Data Base

---

# 2.3 Chapter 3. Defining HIDAM (and Primary INDEX) Data Bases

A HIDAM data base in auxiliary storage is actually comprised of two data bases. Both data bases must be defined. One is defined as the primary INDEX data base and the other is defined as the HIDAM data base.

Subtopics:



# 2.3.1 Defining the HIDAM Data Base

***For a HIDAM data base description,*** you must prepare an input stream of several DL/I control statements specifying:

- The symbolic VSAM data set name for this HIDAM data base.
- The segment types for the data base.
- The use of physical child and physical twin pointers between segments in the data base.
- The index relationship between the HIDAM data base and the primary INDEX data base.
- The fields for each segment type, one of which identifies a sequence field in the root segment.

Subtopics:



## 2.3.1.1 Input Structure and Rules

Figure 9 illustrates the rules for structuring a batch input stream to define a HIDAM data base. The figure shows the types of control statements required, the number of each type that may be specified, and their specific order of sequence. A separate input stream is required for each data base you define.

***To Define a HIDAM Data Base . . .***

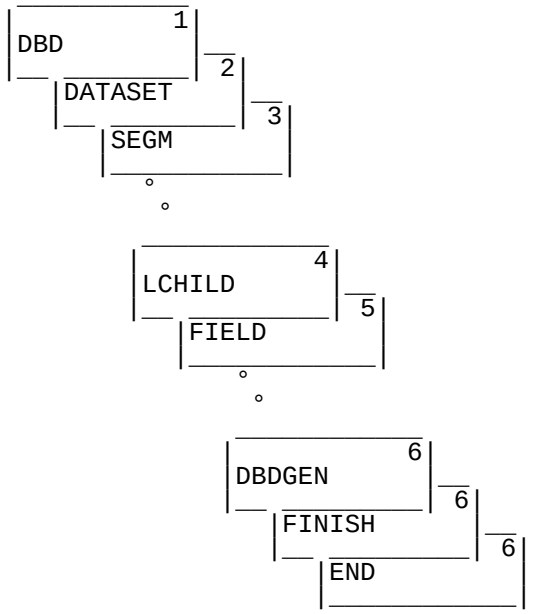



Figure 9. Batch Input Stream for HIDAM Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.
2. Code one DATASET statement. It must follow the DBD statement.
3. Code one SEGM statement for each segment type in the data base. Place them after the DATASET statement in the same sequence as the segments appear in the hierarchic order of the HIDAM data base being defined. The maximum number allowed is 255.
4. Code one LCHILD statement to establish the primary index relationship. Put the LCHILD statement after the SEGM statement that defines the root segment.
5. Code at least one FIELD statement. Use it to identify a unique sequence field within the root segment. Optionally, you may define a maximum of 255 fields per segment and 1020 fields per data base. Use one FIELD statement for each field you define. Place them after the SEGM statement defining the segment that contains those fields.
6. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

# 2.3.1.2 Control Statements Format

This section explains how to code DL/I control statements to define a HIDAM data base. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introduction.

Subtopics:



## 2.3.1.2.1 DBD Statement

The format of the DBD control statement when used to define a HIDAM data base is:

Table 17. DBD Statement Syntax for HIDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Data base name. 1-7 alphameric characters; the first must be alphabetic. |
| | ,ACCESS= | **H I D A M** | HIDAM = Hierarchical Indexed Direct Access Method |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

*Coding Example*

```
      DBD                                                  X
            NAME=STDIDBP,                                  X
            ACCESS=HIDAM,                                  X
            IMSCOMP=NO
```

**NAME=**
Specifies the name of the HIDAM data base. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Specifies the access method used for the data base. Here you must specify HIDAM.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

**Note:** Changing the IMSCOMP= entry for an existing data base without unloading and reloading the data base may cause unpredictable results, due to a different record size used by DL/I

```
internally in the VSAM CI.
```

---

---

## 2.3.1.2.2 DATASET Statement

The format of the DATASET control statement when used to define the characteristics of a HIDAM data base is:

Table 18. DATASET Statement Syntax for HIDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Name of data set; 1-7 alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | _ _ _ _ _ | VSAM control interval size; (512-8192 bytes in steps of 512 bytes; 8192-30720 bytes in steps of 2048 bytes. |
| | ,SCAN= | _ _ _ _ _ | Number of cylinders (or blocks if FBA) to be scanned when searching for space during segment insertions.(2) |
| | ,FRSPC= | (_ _ _<br>,_ _) | Leave every nth block free (0-100 excluding 1).<br>Leave this percent free in every block (0-99).(3) |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate a value during DBD generation.
2. The SCAN operand is optional. See the operand description for details.
3. The FRSPC operand is optional. If omitted, the default is 0.

*Coding Example*

```
DATASET,                                    X
      DD1=STDIDBC,                          X
      DEVICE=3380,                          X
      BLOCK=2048,                           X
      SCAN=2
```

**DD1=**
Specifies the name you want used as the symbolic VSAM data set name

for this HIDAM data base. The name must be 1-7 alphameric characters long.

**DEVICE=**
Identifies the physical device type used for storage of this data base. You must use one of the following:

FBA, 3380, 3375, 3350, 3340, 3330, or 2314.

**BLOCK=**
Specifies the VSAM control interval size. The size must be from 512-8192 bytes in steps of 512 bytes or from 8192-30720 bytes in steps of 2048 bytes. This operand is optional. If omitted, DL/I will determine the size during DBD generation.

**SCAN=**
Specifies the number of cylinders (for count-key-data devices) or blocks (for FBA devices) to be scanned in both directions when searching for available storage space during segment insertions. "Cylinders" may be specified as any integer from 0 to 255. If you specify "blocks," it can be any integer from 0 to 32767.

The SCAN operand is optional. If omitted for count-key-data devices, the default is 3 cylinders. If omitted for FBA devices, a default is calculated during DBD generation that is approximately equal to three cylinders.

**Note:** If SCAN=0 is specified, only the current cylinder for COUNT-KEY-DATA devices is scanned. For FBA devices the number of blocks that is equal to one logical cylinder is scanned. Scanning is performed in both directions from the current position. If space cannot be found for segment insertion within the bounds defined by the operand, space at the current 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.

The first parameter specifies that every *nth* block is to be left free. This is a number from 0 to 100, excluding 1. Zero is the default.

The second parameter specifies the percentage of each block to be left free. This is a number from 0 to 99. Zero is the default.

Either or both parameters may be specified to achieve any combination of free and/or partially free blocks.

***Example:*** A specification of FRSPC=(5, 40) would result in a data base load (or reload) in which every fifth block (5, 10, 15, etc.) would be left free and at least forty 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.

**Note:** 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.

---



---

## 2.3.1.2.3 SEGM Statement

The format of the SEGM control statement when used to define a segment in a HIDAM data base is:

Table 19. SEGM Statement Syntax for HIDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Segment name; 1-8 characters. |
| | ,PARENT= | ((_ _ _ _ _ _ _ _<br>,_ _ _ _)) | Name of this segment's parent.(1)<br>SNGL - for a PCF pointer.<br>DBLE - for both PCF and PCL pointers. |
| | ,BYTES= | _ _ _ _<br>or<br>(_ _ _ _ , _ _ _ _) | Fixed-length segment size.(2)<br>or<br>Variable-length segment size (maximum, minimum lengths). |
| | ,POINTER= | _ _ _ _ _ _ _ | TWIN for a PTF pointer.<br>TWINBWD for both PTF and PTB pointers.<br>NOTWIN for no pointers, but to ensure only one segment occurrence per parent. |
| | ,RULES= | (,_ _ _ _ _) | FIRST, LAST, or HERE. |
| | ,COMPRTN= | (_ _ _ _ _ _ _ _<br>,_ _ _ _<br>,_ _ _ _) | Name of segment compression routine.(3)<br>DATA for upward source compatibility to IMS/VS.<br>INIT indicates that initialization and termination processing control is required. |

**Notes:**

1. For a root segment the PARENT operand may be omitted or else PARENT=0 may be specified. For a dependent segment the first parameter is required, but the second parameter is optional. If you omit it, SNGL is used by default. If both parameters of the PARENT operand are specified, note the double parentheses which must be coded. Do not code the double parentheses, however, if only the first parameter is specified.
2. The BYTES operand is optional. If omitted, DL/I assumes the segment is a fixed-length segment and calculates its length during the DBDGEN procedure.
3. The COMPRTN operand is optional. It is used to select the segment compression option for variable-length segments. See the operand description for details.

*Coding Examples*

```
SEGM                                          X
      NAME=STPIITM,                           X
      PARENT=0,                               X
      BYTES=56,                               X
      POINTER=TWIN
SEGM                                          X
      NAME=STSILOC,                           X
      PARENT=STPIITM,                         X
      BYTES=12,                               X
      POINTER=TWINBWD
```

**NAME=**

Specifies the name of the segment being defined. The name must be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. Duplicate segment names are not allowed within a DBD.

**PARENT=**

Specifies the name of the physical parent of this segment. For the root segment, either omit this operand or specify PARENT=0.

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 pointer* and *physical child last pointer* are used in this segment's parent. The second parameter is optional. If omitted, SNGL is the default. If both parameters are specified, note the double parentheses which must be coded.

**For More Information...**

about physical child first (PCF) and physical child last (PCL) pointers, see *DL/I DOS/VS Data Base Administration*.

**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. Segments residing in an HIDAM data base may be either fixed-length or variable-length.

*For fixed-length segments,* only one parameter is used. It specifies the number of bytes in the data portion of the segment.

*For variable-length segments,* two parameters are used. The first parameter specifies the maximum length of the data portion of a segment, including the 2-byte length field. The second parameter specifies the minimum length of the data portion of a segment, including the 2-byte length field. Four is the minimum specification.

The BYTES operand is optional. If it is omitted, DL/I assumes the segment to be a fixed-length segment and will calculate its length during DBD generation.

**For More Information...**

about determining segment lengths and about using variable-length segments, see *DL/I DOS/VS Data Base Administration*.

**POINTER= (or its abbreviation PTR)**
Specifies the pointer fields to be reserved in the segment's prefix area. These pointers are used to relate this segment to its physical twin segments.

**TWIN (or T)**
Reserves a 4-byte field in this segment's prefix for a physical twin forward (PTF) pointer.

**TWINBWD (or TB)**
Reserves two 4-byte fields in this segment's prefix for both a physical twin forward (PTF) pointer and physical twin backward (PTB) pointer. For performance reasons, always specify POINTER=TWINBWD for root segments of HIDAM data bases.

**NOTWIN (or NT)**
Ensures that no more than one occurrence of this segment type will exist under its parent. If coded for the root segment, only one root segment can exist in this data base.

The POINTER operand is optional. If omitted, the default is TWIN.

**For More Information...**

about physical twin forward (PTF), and physical twin backward (PTB) pointer, and the NOTWIN parameter, see *DL/I DOS/VS Data Base Administration*.

**RULES=**
This parameter of the RULES 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 (or F)**
States that a new occurrence is to be inserted before the first existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted before all existing occurrences of the same sequence field value.

**LAST (or L)**
States that a new occurrence is to be inserted after the last existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted after all existing occurrences of the same sequence field value. This is the default option.

**HERE (or H)**
Assumes the user has determined positioning by a previous DL/I call, and the new occurrence is inserted before the segment that satisfied the last call. If the segment has a non-unique 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.

**COMPRTN=**
This operand is used to select the segment compression option. It is supported only for variable-length segments.

The first parameter specifies the name of a user-supplied routine used to compress this segment. The name must be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. The phase name (type PHASE) should be unique.

The second parameter maintains upward source compatibility to IMS/VS. Either specify DATA or omit it; DATA is the default value.

The third parameter is optional. If INIT is specified, it indicates that initialization and termination processing control is required by the segment compression routine.

**For More Information...**

about segment compression routines, see *DL/I DOS/VS Data Base Administration*.

## 2.3.1.2.4 LCHILD Statement

The format of the LCHILD control statement when used to establish a primary index relationship in a HIDAM data base is:

Table 20. LCHILD Statement Syntax for HIDAM Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| LCHILD | NAME= | (° ° ° ° ° ° ° °<br>,° ° ° ° ° ° ° ) | Name of pointer segment in primary INDEX data base.<br>Name of primary INDEX data base. |
| | ,POINTER= | **I N D X** | |

*Coding Example*

```
LCHILD                                              X
      NAME=(STIIITM,STDIXIP),                       X
      POINTER=INDX
```

**NAME=**

The first parameter specifies the name of the index pointer segment defined in the primary INDEX data base.

The second parameter specifies the DBD name of the primary INDEX data base which contains the index pointer segment.

**POINTER=**

INDX identifies this segment (that is, the root segment) as an index target segment in the HIDAM data base.

## 2.3.1.2.5 FIELD Statement

The basic format of the FIELD control statement when used to define a field within a segment of a HIDAM data base is:

Table 21. FIELD Statement Syntax for HIDAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (° ° ° ° ° ° ° ° ,_ _ _ ,_) | Field name; 1-8 characters.<br>SEQ - identifies this as a sequence field.(1)<br>U - only unique values of this sequence field allowed.<br>M - duplicate values of this sequence field can occur in multiple occurrences of the segment. |
| | ,BYTES= | _ _ _ | Field length; 1-256 bytes.(2) |
| | ,START= | _ _ _ _ _ _ _ _ | Field starting position.(3) |
| | ,TYPE= | _ | Field data type.(4) |

**Notes:**

1. The parameters SEQ and U or M are optional. Specify them only if you want to identify this field as a sequence field in a segment type.
2. See "Rules and Restrictions" in the BYTES operand description.
3. The START operand is optional. If omitted, DL/I will determine the field's starting position during DBD generation.
4. The TYPE operand is optional. If omitted, TYPE=C is assumed.

*Coding Example*

```
       FIELD                                          X
             NAME=(STQIINQ,SEQ,U),                    X
             BYTES=6,                                 X
             START=1,                                 X
             TYPE=C
```

**NAME=**

The first parameter specifies the name of the field being defined within a segment type. The name specified can be referred to by an application program in a DL/I call SSA. Duplicate field names must not be defined for the same segment type. The field name can be 1-8 alphameric characters long. It is recommended that you start the field name with an alphabetic character.

The presence of the word SEQ as a second parameter of this operand identifies this field as a sequence field in a segment type. As a general rule, a segment can have only one sequence field.

A unique sequence field must be provided for the root segment. It is optional for all dependent segment types.

**Note:** 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 operand in the SEGM statement.

The last parameter of this operand is optionally used in conjunction with the SEQ parameter. 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 or default.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 256. If this field is the sequence field in the root segment, however, the length cannot exceed 236 bytes.

***Rules and Restrictions***

- The BYTES operand must be specified for field data types X, P, C, and Z (see TYPE operand for field data types).
- The BYTES operand 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 operand for field data types E, D, and 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. Overlapping fields are permitted. If an overlapping field starts in the same position as a previously defined field, you may specify the name of the previously defined field, instead of a numeric value, to indicate the starting position (START=*fieldname*). Each field must not extend beyond the defined segment length (start position plus byte value).

The START operand is optional. if omitted, 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 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.
```

**Notes:**

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

2. It is recommended that you explicitly specify the field 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 during PSB generation.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 2.3.1.2.6 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. They must be included at the end of your DBD generation input stream.

Table 22. DBDGEN, FINISH, END Statements for HIDAM Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

 *© Copyright IBM Corp. 1981, 1989*

# 2.3.2 Control Statements Summary

Table 23 summarizes the basic DL/I control statements used to define HIDAM data bases. Only the control statements and operands pertinent to HIDAM are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 23. Summary of Control Statements Used to Code HIDAM DBDs

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS=HIDAM<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DEVICE=device-type<br>,DD1=filename<br>[,BLOCK=(control-interval-size)]<br>[,SCAN={cylinder}]<br>{blocks }<br>{*3* }<br>[,FRSPC=({free-blocks,percentage})]<br>{*0* ,*0* } |
| | SEGM | NAME=segment-name<br>[,PARENT={*0* }]<br>{((parent-name[{,*SNGL*}]))}<br>{,DBLE}<br>[,BYTES={fixed-length }]<br>{(max-var-length,min-var-length)}<br>[,POINTER={*TWIN* }]<br>{TWINBWD}<br>{NOTWIN }<br>[,RULES=({,FIRST})]<br>{,*LAST* }<br>{,HERE }<br>[,COMPRTN=(routine-name[,*D*,INIT])] |
| | LCHILD | NAME=(index-pointer-seg-name,dbd-name-of-index<br>,POINTER=INDX |
| | FIELD | NAME=(field-name[,SEQ[{,*U*}]])(1)<br>{,M}<br>[BYTES=length]<br>[,START=starting-position]<br>[,TYPE=data-type] |
| | DBDGEN | |
| | FINISH | |
| | END | |

(1)SEQ must be specified for root segment.

Subtopics:

---

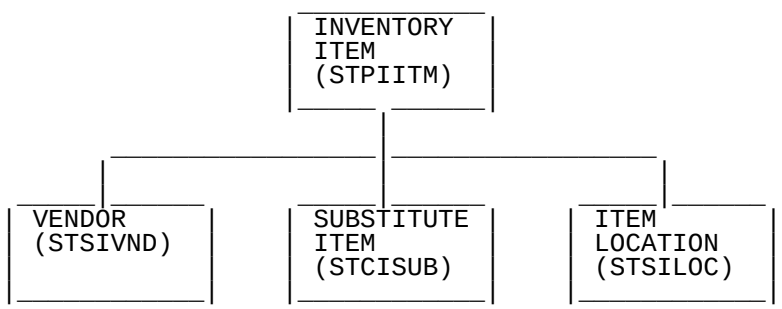
 

---

# 2.3.2.1 Example of HIDAM DBD Generations

Figure 10 shows an example of a HIDAM DBD for the Inventory data base. (The associated primary INDEX data base is shown in Figure 12.)

Inventory Data Base
(DBD Name - STDIDBP)




```
DBD                                                              X
      NAME=STDIDBP,           DATA BASE DESCRIPTION NAME         X
      ACCESS-HIDAM            HIERARCHICAL INDEXED DIRECT        X
      IMSCOMP=NO              IMS COMPATIBILITY NOT REQUIRED
DATASET                                                          X
      DD1=STDIDBC,            DLBL FILE NAME                     X
      DEVICE=3380,            DISK DEVICE                        X
      BLOCK=2048,             VSAM CONTROL INTERVAL SIZE         X
      SCAN=2                  # CYLINDERS SCAN FOR ISRT SPACE
SEGM                                                             X
      NAME=STPIITM,           SEGMENT NAME INVENTORY ITEM        X
      PARENT=0,               IT IS A ROOT SEGMENT               X
      BYTES=56,               DATA LENGTH                        X
      POINTER=TWINBWD         BOTH PHYS. TWIN FWD AND BWD
LCHILD                                                           X
      NAME=(STIIITM,STDIX1P), INDEX SEGM AND DBD NAME            X
      PTR=INDX                INDEX POINTERS
FIELD                                                            X
      NAME=(STQIINO,SEQ,U),   UNIQUE KEY FIELD (ITEM #)          X
      BYTES=6,                FIELD LENGTH                       X
      START=1,                WHERE IT STARTS IN SEGMENT         X
      TYPE=C                  ALPHAMERIC
SEGM                                                             X
      NAME=STSIVND,           AUTHORIZED VENDOR INFORMATION      X
      PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM.     X
      BYTES=106,              FIELD LENGTH                       X
      POINTER=TWIN            PHYSICAL TWIN FWD ONLY
FIELD                                                            X
      NAME=(STQVVNO,SEQ,U),   UNIQUE KEY FIELD (VENDOR #)        X
      BYTES=6,                FIELD LENGTH                       X
      START=1,                WHERE IT STARTS IN SEGMENT         X
      TYPE=C                  ALPHAMERIC DATA


SEGM                                                             X
      NAME=STCISUB,           SEG. NAME FOR SUB-ITEM INFO.       X
      PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM      X
      BYTES=56,               DATA LENGTH                        X
      POINTER=TWINBWD         BOTH PHYS. TWIN FWD AND BWD
FIELD                                                            X
      NAME=(STQCCNO,SEQ,U),   UNIQUE KEY FIELD (SUB-ITEM #)      X
      BYTES=6,                FIELD LENGTH                       X
      START=1,                WHERE IT STARTS IN SEGMENT         X
```

```
      TYPE=C                  ALPHAMERIC DATA
SEGM                                                          X
      NAME=STSILOC,           SEGMENT NAME INVENTORY LOCATION X
      PARENT=STPIITM,         PARENT IS INVENTORY ITEM SEGM.  X
      BYTES=12,               DATA LENGTH                     X
      POINTER=TWINBWD         BOTH PHYS. TWIN FWD AND BWD
FIELD                                                         X
      NAME=(STQILNO,SEQ,U),   UNIQUE KEY FIELD INVENTORY LOC# X
      BYTES=6,                FIELD LENGTH                    X
      START=1,                WHERE IT STARTS IN SEGMENT      X
      TYPE=C                  ALPHAMERIC DATA
DBDGEN                        REQUIRED TO MARK DBD END
FINISH                        FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 10. Example of HIDAM DBD for Inventory Data Base

---

# 2.3.3 Defining the Primary INDEX Data Base

***For a primary INDEX data base description,*** you must prepare an input stream of several DL/I control statements specifying:

- The symbolic VSAM data set name for this primary INDEX data base.
- The index relationship between the primary INDEX data base and the HIDAM data base.
- An index pointer segment containing the sequence field value of the root segment occurrence it indexes and a direct address pointer to the root segment occurrence.

Subtopics:



## 2.3.3.1 Input Structure and Rules

Figure 11 illustrates the rules for structuring a batch input stream to define a primary INDEX data base.

***To Define a Primary INDEX Data Base...***

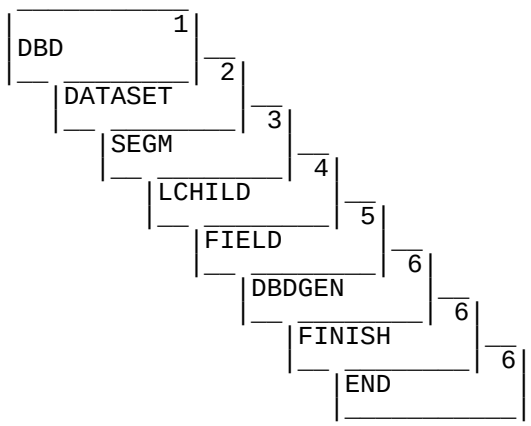



Figure 11. Batch Input Stream for Primary INDEX Data Base

1. Code one DBD statement. It must be the first statement in your input stream.

2. Code one DATASET statement. It must follow the DBD statement.

3. Code one SEGM statement to identify the index pointer segment. The SEGM statement must follow the DATASET statement.

4. Code one LCHILD statement to identify the index target segment. Place it after the SEGM statement.

5. Code one FIELD statement to identify the unique sequence field. Place it after the LCHILD statement.

6. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

# 2.3.3.2 Control Statements Format

This section explains how to code DL/I control statements to define a primary INDEX data base.

Subtopics:



## 2.3.3.2.1 DBD Statement

The format of the DBD control statement when used to define a primary INDEX data base is:

Table 24. DBD Statement Syntax for Primary INDEX Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Name of INDEX data base. |
| | ,ACCESS= | **I N D E X** | INDEX identifies this DBD as a primary INDEX DBD. |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

*Coding Example*

```
DBD                                                    X
      NAME=STDIX1P,                                    X
      ACCESS=INDEX
```

**NAME=**
Specifies the name of the primary INDEX data base. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Here you must specify INDEX. It identifies this DBD as an INDEX DBD.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

## 2.3.3.2.2 DATASET Statement

The format of the DATASET control statement when used to define the characteristics of a primary INDEX data base is:

Table 25. DATASET Statement Syntax for Primary INDEX Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Name of data set; 1-7 alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | _ _ _ _ | Number of VSAM logical records in one control interval (CI).(1) |
| | ,RECORD= | _ _ _ _ | Logical record length.(2) |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure.
2. The RECORD operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure.

*Coding Example*

```
       DATASET                                          X
             DD1=STDIX1C,                               X
             DEVICE=3380
```

**DD1**
Specifies the name you want used as the symbolic VSAM data set name for this primary INDEX data base. The name must be 1-7 alphameric characters long. This name is required for cross-reference to the DASD label information (DLBL).

**DEVICE=**
Identifies the physical device type used for storage of this data base. You must use one of the following:

FBA, 3380, 3375, 3350, 3340, 3330, or 2314.

**BLOCK=**
Specifies the number of VSAM logical records to be contained within

one control interval. The BLOCK operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure using a control interval size of 2048 bytes wherever possible.

**RECORD=**

Specifies the logical record length. The length cannot exceed 4088 bytes and must be an even number large enough to contain the index pointer segment plus 11 bytes.

The RECORD operand is optional. If omitted, the length is calculated during the DBDGEN procedure.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 2.3.3.2.3 SEGM Statement

The format of the SEGM control statement when used to define a primary INDEX data base is:

Table 26. SEGM Statement Syntax for Primary INDEX Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Name of index pointer segment. |
| | ,BYTES= | _ _ _ _ | Length of index pointer segment.(1) |

**Notes:**

1. The BYTES operand is optional. If omitted, DL/I will calculate the segment length during DBDGEN procedure.

*Coding Example*

```
SEGM                                                    X
      NAME=STIIITM,                                     X
      BYTES=6
```

**NAME=**
Specifies the name of the index pointer segment.

**BYTES=**
Specifies the length of the data portion of the index pointer segment.

The BYTES operand is optional. If omitted, the segment size is calculated during the DBDGEN procedure. This calculation is based on the end of the last defined field in the segment.

 *© Copyright IBM Corp. 1981, 1989*

## 2.3.3.2.4 LCHILD Statement

The format of the LCHILD statement when used to define a primary INDEX data base is:

Table 27. LCHILD Statement Syntax for Primary INDEX Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| LCHILD | NAME= | (° ° ° ° ° ° ° °<br>,° ° ° ° ° ° ° ° ) | Name of index target segment.<br>Name of its DBD. |
| | ,POINTER= | **S N G L** | SNGL. |
| | ,INDEX= | ° ° ° ° ° ° ° ° | Name of the indexed field. |

*Coding Example*

```
       LCHILD                                          X
             NAME=(STPIITM,                            X
             STDIDBP),                                 X
             POINTER=SNGL,                             X
             INDEX=STQIINQ
```

**NAME=**

The first parameter specifies the name of the index target segment defined in the HIDAM data base.

The second parameter specifies the name of the HIDAM data base which contains the index target segment.

**POINTER=**

SNGL must be specified. It causes a 4-byte direct address pointer to be put in the prefix of the index pointer segment. It points to the index target segment.

**INDEX=**

Specifies the name of the field being indexed. The indexed field must be defined in the root segment of the HIDAM data base of this index relationship.

## 2.3.3.2.5 FIELD Statement

When defining a primary INDEX data base, you must code one FIELD statement using the format shown here. This required FIELD statement defines a unique sequence field in the INDEX data base.

Optionally, additional fields may also be defined in the INDEX data base. These fields may be useful, for example, if you plan to use the INDEX data base as a separately processible data base. If these additional fields are desired, use the FIELD statement format previously shown for HIDAM data bases to define them.

Table 28. FIELD Statement Syntax for Primary INDEX Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (° ° ° ° ° ° ° ° ,**S E Q**) | Field name. SEQ must be specified. |
| | ,BYTES= | _ _ _ | Field length; 1-236 bytes.(1) |
| | ,TYPE= | _ | Field data type.(2) |

**Notes:**

1. See "Rules and Restrictions" in the BYTES operand description.
2. The TYPE operand is optional. If omitted, TYPE=C is assumed.

*Coding Example*

```
FIELD                                                  X
      NAME=(STYIINQ,SEQ,U),                            X
      BYTES=6,                                         X
      TYPE=C
```

**NAME=**

The first parameter specifies the name of this field. The name can be 1-8 alphameric characters long.

SEQ must be specified to identify this field as a sequence field.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 236.

***Rules and Restrictions***...

- The BYTES operand must be specified for field data types X, P, C, and Z (see TYPE operand for field data types).

- The BYTES operand 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 operand for field data types E, D, and L. These data types have implicit lengths of 4, 8, and 16, respectively.

**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.

**Notes:**

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

2. It is recommended that you explicitly specify the field 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 during PSB generation.

---



---

## 2.3.3.2.6 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. They must be included at the end of your DBD generation input stream.

Table 29. DBDGEN, FINISH, END Statements for Primary INDEX Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

 *© Copyright IBM Corp. 1981, 1989*

# 2.3.4 Control Statements Summary

Table 30 summarizes the DL/I control statements used to define primary INDEX data bases. Only the control statements and operands pertinent to Primary INDEX are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 30. Summary of Control Statements Used to Code Primary INDEX DBDs

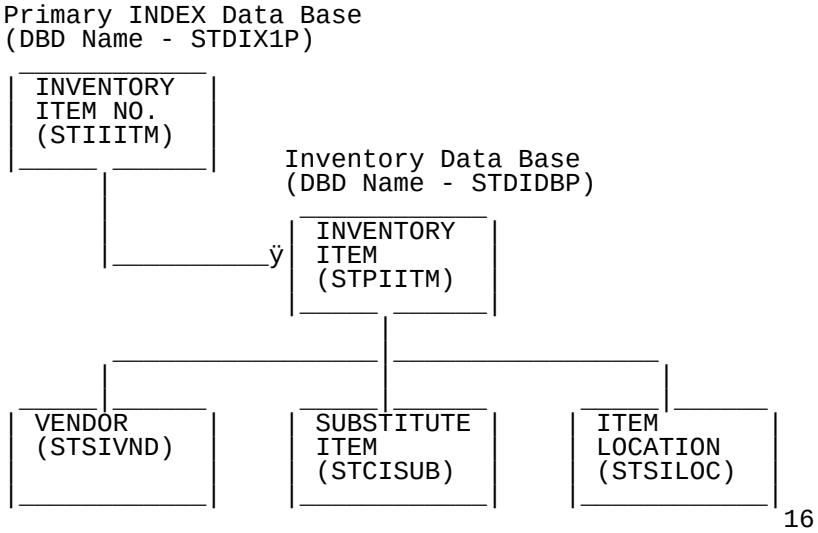
| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS=INDEX<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DD1=filename<br>,DEVICE=device-type<br>[,BLOCK=(number-of-logical-records-per-CI)]<br>[,RECORD=(logical-record-length)] |
| | SEGM | NAME=segment-name<br>[,BYTES=segment-length] |
| | LCHILD | NAME=(index-target-seg-name,dbd-name)<br>[,POINTER=*SNGL*]<br>,INDEX=indexed-field-name |
| | FIELD | NAME=(field-name,SEQ[,*U*])<br>[,BYTES=length<br>[,START=1]<br>[,TYPE=data-type] |
| | DBDGEN | |
| | FINISH | |
| | END | |

Subtopics:

- 2.3.4.1 Example of a Primary INDEX Data Base

# 2.3.4.1 Example of a Primary INDEX Data Base

Figure 12 shows an example of a primary INDEX data base. (An example of its associated HIDAM data base is shown in Figure 10.)




```
DBD                                                              X
      NAME=STDIX1P,           DATA BASE DESCRIPTION NAME         X
      ACCESS=INDEX            THIS IS AN INDEX
DATASET                                                          X
      DD1=STDIX1C,            DLBL FILE NAME                     X
      DEVICE=3380             DISK DEVICE
SEGM                                                             X
      NAME=STIIITM,           SEGMENT NAME OF THE INDEX          X
      BYTES=6                 LENGTH
LCHILD                                                           X
      NAME=(STPIITM,          TARGET SEGMENT ITEM NUMBER         X
      STDIDBP),               FOUND IN INVENTORY DATA BASE       X
      INDEX=STQIINO           INDEXED FIELD NAME
FIELD                                                            X
      NAME=(STYIINO,SEQ,U),   UNIQUE KEY FIELD                   X
      BYTES=6,                FIELD LENGTH                       X
      TYPE=C                  ALPHAMERIC DATA
DBDGEN                        TO MARK END OF DBD
FINISH                        FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 12. Example of Primary INDEX DBD for Inventory Data Base

# 2.4 Chapter 4. Defining Logical Relationships in HD, HDAM, and HIDAM

This chapter describes how to implement DL/I's logical relationship function in HD organization data bases. Two things must be done:

1. The HD, HDAM, or HIDAM *physical data bases* as previously described in Chapter 1, "Defining HD Data Bases," Chapter 2, "Defining HDAM Data Bases," or Chapter 3, "Defining HIDAM (and Primary INDEX) Data Bases" respectively, must be extended to establish the logical relationship. This means coding additional control statements and parameters to define those segment types involved in the logical relationship.

2. A new type of data base description called a LOGICAL DBD must be defined for the *logical* data base.

Subtopics:

# 2.4.1 Defining Logical Relationships in Physical Data Bases

Three segment types are needed to establish a logical relationship in a physical data base:

- Logical child
- Physical parent
- Logical parent

In addition, a special segment type called a *virtual logical child* is needed to define the logical child as seen from the logical path in a bidirectional logical relationship.

The following text describes the control statements used to define each of these segment types.

Subtopics:



# 2.4.1.1 Defining a Logical Child

The following format of the SEGM statement must be coded for each logical child in the logical relationship.

Table 31. SEGM Statement Syntax to Define a Logical Child

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Name of logical child segment. |
| | ,PARENT= | ((° ° ° ° ° ° ° ° ,_ _ _ _) ,(° ° ° ° ° ° ° ° ,_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _)) | Name of its physical parent. SNGL or DBLE. Name of its logical parent. VIRTUAL (or V) for IMS/VS source compatibility. DBD name of logical parent's data base.(1) |
| | ,BYTES= | _ _ _ _ | Fixed-length segment size.(2) |
| | ,POINTER= | (_ _ _ _ _ _ _ _ ,_ _ _ _ _ _ _ _ _) | TWIN or TWINBWD or NOTWIN.(3) LTWIN or LTWINBWD.(3) |
| | ,RULES= | (_ _ V) ,_ _ _ _ _) | (First): insertion rules: P, L, or V. (Second): deletion rules: P, L, or V. (Third): replacement rules: must be V.(4). FIRST, LAST, or HERE. |

**Notes:**

1. The last parameter of the PARENT operand may be omitted only if the logical parent is defined in the same data base as the logical child.
2. The BYTES operand is optional. If omitted, DL/I automatically calculates a segment size.
3. The POINTER operand is optional. If omitted, the default is POINTER=(TWIN,LTWIN). Be aware that if you explicitly specify any one POINTER keyword, no defaults are taken. The default takes place only if the POINTER operand is omitted.
4. For logical child segments, the third value, the replace rule, must be V. If P or L is specified, it will be ignored, and V will be assumed instead.

*Coding Example*

```
       SEGM                                                  X
             NAME=STCISUB,                                   X
             PARENT=((STPIITM,                               X
```

```
|                SNGL),                                          X      |
|                (STPIITM,                                       X      |
|                V,                                              X      |
|                STDIDBP)),                                      X      |
|                BYTES=6,                                        X      |
|                POINTER=TWINBWD,                                X      |
|                RULES=(PPV,                                     X      |
|                HERE)                                                  |
|                                                                       |
```

**NAME=**
Specifies the name of the logical child segment. The name can be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. Duplicate segment names are not allowed within a DBD.

**PARENT=**
The first parameter specifies the name of the physical parent segment of this logical child.

The second parameter controls the physical child pointer(s) in the physical parent. SNGL specifies only a *physical child first pointer* is used. DBLE specifies both a *physical child first pointer* and *physical child last pointer* are used. This parameter is optional. If omitted, SNGL is the default.

The third parameter specifies the name of the logical parent segment of this logical child.

V (or VIRTUAL, the default value) maintains upward source compatibility to IMS/VS.

The last parameter specifies the DBD name of the logical parent's data base. If the logical parent is defined in the same data base as this logical child, this parameter may be omitted.

**BYTES=**
Specifies the length of the data portion of the segment in bytes. The logical child always contains the logical parent's concatenated key in the first *n* bytes, and its length must be included here. The BYTES operand is optional. If omitted, DL/I automatically calculates a segment size large enough to contain all *defined* fields.

**POINTER= (or its abbreviation PTR)**
Specifies the pointer fields to be reserved in this segment's prefix area.

The first parameter is used to relate this segment to its physical twin segments. The keyword options which may be specified are:

**TWIN (or T)**
Reserves a 4-byte field in this segment's prefix for a physical twin forward (PTF) pointer.

**TWINBWD (or TB)**
Reserves two 4-byte fields in this segment's prefix for both a physical twin forward (PTF) pointer and physical twin backward (PTB) pointer.

**NOTWIN (or NT)**
Ensures that no more than one occurrence of this segment type will exist under its physical parent. If coded for the root segment, only one root segment can exist in this data base.

The second parameter is used to relate this segment to its logical twin segments. The keyword options which may be specified are:

**LTWIN (or LT)**
Reserves a 4-byte field in this segment's prefix for a logical twin forward (LTF) pointer.

**LTWINBWD (or LTB)**
Reserves two 4-byte fields in this segment's prefix for both a logical twin forward (LTF) pointer and a logical twin backward (LTB) pointer.

The POINTER operand is optional for a logical child segment. If omitted, the default is POINTER=(TWIN,LTWIN). Specification of only the first parameter results in a default of no logical twin pointers for the second parameter.

Specification of only the second parameter results in a default of NOTWIN for the first parameter.

The following table summarizes the keyword options that may be specified for the POINTER operand. For more information about pointers, see *DL/I DOS/VS Data Base Administration*.

| **Valid for Physical Segments and Logical Parent Segments** | **Valid for Logical Child Segments** | **Valid for Virtual Logical Child Segments** |
|---|---|---|
| *TWIN*<br>TWINBWD<br>NOTWIN | TWIN<br>TWINBWD<br>NOTWIN<br>LTWIN<br>LTWINBWD<br>*TWIN,LTWIN*<br>(TWIN,LTWINBWD)<br>(TWINBWD,LTWIN)<br>(TWINBWD,LTWINBWD)<br>(NOTWIN,LTWIN)<br>(NOTWIN,LTWINBWD) | PAIRED |
| **Notes:**<br>° If you do not specify the POINTER operand for a physical segment or logical parent segment, DL/I will default to TWIN. | **Notes:**<br>° If two keywords are selected, they may be specified in any order. They must, however, be enclosed in parentheses and separated by a comma.<br>° If you do not specify the POINTER operand for a logical child segment, DL/I will default to (TWIN,LTWIN). If you explicitly specify any one POINTER keyword option, no defaults are taken. The | **Notes:**<br>° No other keyword option is allowed for a virtual logical child segment. |

| | | |
|---|---|---|
| | default takes place only if the POINTER operand is omitted. | |

**RULES=**

The first parameter of this operand is of the format xxx, where x can be one of the characters P, L, or V. The P stands for physical rule, the L for logical rule, and the V stands for virtual rule. Each of the three positions in the first parameter can contain the same or different characters. If all three positions are omitted, the default value LLV is used. Likewise, if you omit the second and third positions, or just the third position, the default values xLV and xxV, respectively, will be used.

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, the third value, the replace rule, must be V. Any other rule specified will be changed to V during DBD generation.

**For More Information...**

to help you in determining when to specify P, L or V for the RULES operand, see *DL/I DOS/VS Data Base Administration*.

The second parameter of the RULES 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 (or F)**

States that a new occurrence is to be inserted before the first existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted before all existing occurrences of the same sequence field value.

**LAST (or L)**

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

**HERE (or H)**

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

# 2.4.1.2 Defining a Virtual Logical Child

The following format of the SEGM statement must be coded for each virtual logical child in a bidirectional logical relationship.

Table 32. SEGM Statement Syntax to Define a Virtual Logical Child

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Name of virtual logical child segment. |
| | ,PARENT= | ° ° ° ° ° ° ° ° | Name of logical parent. |
| | ,SOURCE= | ((° ° ° ° ° ° ° ° ,_ _ _ _ ,_ _ _ _ _ _ _ _)) | Name of real logical child segment. DATA (or D) for IMS/VS source compatibility. DBD name of real logical child's data base.(1) |
| | ,POINTER= | **P A I R E D** | PAIRED must be specified. |

**Notes:**

1. The last parameter of the SOURCE operand may be omitted only if the real logical child is defined in the same data base as the virtual logical child.

*Coding Example*

```
SEGM                                                    X
      NAME=STVICOR,                                     X
      PARENT=STPIITM,                                   X
      POINTER=PAIRED,                                   X
      SOURCE=((STCCMM,                                  X
      D,                                                X
      STDCDBP))
```

**NAME=**
Specifies the name of the virtual logical child. The name can be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. Duplicate segment names are not allowed within a DBD.

**PARENT=**
Specifies the name of the logical parent segment.

**POINTER=**

POINTER=PAIRED must be specified. It identifies this segment as a virtual logical child segment.

**SOURCE=**

The first parameter specifies the name of the real logical child segment to which this virtual logical child segment is *paired.*

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

The last parameter specifies the DBD name of the logical child's data base. If the logical child is defined in the same data base as this virtual logical child, this parameter may be omitted.

---

**Describing Fields of a VLC Segment**

FIELD statements may be used in HD organization DBD definitions to describe fields of a virtual logical child segment.

As a general rule, a segment can have only one sequence field. However, in the case of a bidirectional logical relationship, multiple FIELD statements may be used to define a logical twin sequence field for the virtual logical child segment, as described below.

A sequence field must be specified for a virtual logical child segment (even though the segment does not physically exist) if, when accessing a logical child segment from its logical parent path, the segments from the logical twin chain are required to be retrieved in a specific order. This sequence field can include any part of the segment as it appears when viewed from the logical parent (that is, the concatenated key of the real logical child's physical parent followed by any intersection data). Since it may be necessary to describe the sequence field of a logical child segment (when accessed from its logical parent segment) in separate pieces, multiple FIELD statements with the SEQ parameter present are permitted. Each statement must contain a unique fld-name1 parameter. However, when using the sequence field (or parts of it) as a qualification in an SSA, the first field name specified must be used and the key value specified in the SSA must be the same length as the combined lengths of all the fields which comprise the "scattered" sequence field. Note that this "scattered" sequence field is permitted only when specifying the sequence field for a virtual logical child segment.

If a sequence field is defined for a (real) logical child segment, it can only be part or all of the intersection data.

It is highly recommended that all segments which participate in a logical relationship have sequence fields. This includes physical and logical parents as well as logical child segments.

---

# 2.4.1.3 Defining Physical and Logical Parents

With the exception of one additional parameter, the SEGM statement as previously described for physical data bases in Chapter 1, "Defining HD Data Bases," Chapter 2, "Defining HDAM Data Bases" or Chapter 3, "Defining HIDAM (and Primary INDEX) Data Bases," is also used to define both the physical and the logical parent in a logical relationship.

As shown below, the additional parameter is the first parameter of the RULES operand. The RULES operand and both of its parameters have the same meaning as previously described for "Defining a Logical Child" in this chapter.

| | | | |
|---|---|---|---|
| .<br>.<br>. | | | |
| | ,RULES= | (_ _ _)<br>,_ _ _ _ _) | (First): insertion rules: P, L, or V.<br>(Second): deletion rules: P, L, or V.<br>(Third): replacement rules: P, L or V.<br>FIRST, LAST, or HERE.(4). |
| .<br>.<br>. | | | |

For each logical child segment type, an LCHILD statement must be coded and placed after the SEGM and/or FIELD statements of the logical parent.

Two formats of the LCHILD statement are shown below. Use the first format if the logical child segment is in a *unidirectional* logical relationship. Use the second format if the logical child segment is in a *bidirectional* logical relationship.

Table 33. LCHILD Statement Syntax to Define Physical and Logical Parents

| Statement | Keyword | Parameter | Description (See also Notes below). |
|---|---|---|---|
| LCHILD | NAME= | (° ° ° ° ° ° ° °<br>,_ _ _ _ _ _ _) | Name of logical child segment.<br>DBD name of logical child's data base.(1) |
| | ,POINTER= | **N O N E** | NONE (for unidirectional logical relationship). |

**Notes:**

1. The second parameter of the NAME operand may be omitted only if the logical child named in the first parameter resides in this data base.

*Coding Example*

```
LCHILD                                                    X
      NAME=(STCISUB,                                      X
      STDIDBP),                                           X
      POINTER=NONE
```

Table 34. LCHILD Statement Syntax to Define Physical and Logical Parents

| Statement | Keyword | Parameter | Description (See also Notes below). |
|---|---|---|---|
| LCHILD | NAME= | (° ° ° ° ° ° ° °<br>,_ _ _ _ _ _ _) | Name of logical child segment. DBD name of logical child's data base.(1) |
| | ,POINTER= | ° ° ° ° | SNGL or DBLE (for bidirectional logical relationship). |
| | ,PAIR= | ° ° ° ° ° ° ° ° | Name of virtual logical segment. |
| | ,RULES= | _ _ _ _ _ | FIRST, LAST, or HERE. |

**Notes:**

1. The second parameter of the NAME operand may be omitted only if the logical child named in the first parameter resides in this data base.

*Coding Example*

```
LCHILD                                                    X
      NAME=(STCCITM,                                      X
      STDCDBP),                                           X
      POINTER=DBLE,                                       X
      PAIR=STVICOR,                                       X
      RULES=LAST
```

**NAME=**

The first parameter specifies the name of the logical child segment. The segment must be defined by a SEGM statement included in this or another DBD generation.

The second parameter specifies the DBD name of the data base in which the logical child segment is defined. It may be omitted if the logical child segment is defined in this DBD generation.

**POINTER= (or its abbreviation PTR)**

Specifies the pointer fields to be reserved in the prefix of the logical parent segment.

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 logical child counter field will be reserved in the prefix of the logical parent segment.

SNGL or DBLE defines a *bidirectional* logical relationship between the logical child and the logical parent. SNGL reserves a 4-byte field in the prefix of the logical parent segment for a logical child first (LCF) pointer. DBLE reserves two 4-byte fields in the prefix for both a logical child first (LCF) pointer and logical child last (LCL) pointer.

**___ For More Information... _____**

about logical child first pointers and logical child last pointers, see *DL/I DOS/VS Data Base Administration*.

**PAIR=**

Specifies the name of the virtual logical child segment to which the real logical child segment is paired.

**RULES=**

Specifies the insertion rule to be used when inserting new occurrences of the logical child segment from the logical parent path (bi-directional logical relationships only). This value is significant only when no sequence field, or a non-unique sequence field, is defined for the logical child segment as viewed from its logical parent. RULES determines the sequence of segments in the logical twin chain. The meaning of each insertion rule is as follows:

**FIRST (or its abbreviation F)**

States that, if no sequence field is specified for the logical child, a new occurrence is inserted before the first existing occurrence of the logical child. If a non-unique sequence field is specified for the logical child, a new occurrence is to be inserted before all existing occurrences of the same sequence field value.

**LAST (or its abbreviation L)**

States that, if no sequence field is specified for the logical child, a new occurrence is inserted after the last existing occurrence of the logical child. This is the default option. If a non-unique sequence field is specified for the logical child, a new occurrence is to be inserted after all existing occurrences of the same sequence field value.

**HERE (or its abbreviation H)**

States that, if no sequence field is specified for the logical child, a new occurrence is inserted before the logical twin segment that satisfied the previous DL/I call. If a non-unique sequence field is specified for the logical child and data base position is not within occurrences of the segment with equivalent sequence field value, a new occurrence is inserted before all existing occurrences of the same sequence field value.

---

# 2.4.2 Control Statements Summary

The next three figures summarize the DL/I control statements used to code a logical relationship in a physical DBD. Table 35 shows the SEGM statement requirements for the *logical child,* Table 36 shows the SEGM statement requirements for a *virtual logical child,* and Table 37 shows the SEGM and LCHILD statement requirements for a *logical parent.*

Table 35. Summary of Control Statements Used to Code Logical Child Segments

| [Label] | Statement | Operand |
|---|---|---|
| | SEGM | NAME=lc-segment-name |
| | | ,PARENT=((pp-segment-name,{*SNGL*}),(lp-segment-name* |
| | | [,*V*,lp-db-name])) |
| | | {DBLE} |
| | | [,BYTES=fixed-length] |
| | | ,POINTER=({*TWIN* }),[{LTWIN }]) |
| | | {TWINBWD} {LTWINBWD} |
| | | {NOTWIN } |
| | | ,RULES=([{P}{P}{P}][{,FIRST}]) |
| | | {*L*}{*L*}{L} {,*LAST* } |
| | | {V}{V}{*V*} {,HERE } |

Table 36. Summary of Control Statements Used to Code Virtual Logical Child Segments

| [Label] | Statement | Operand |
|---|---|---|
| | SEGM | NAME=virt-lc-seg-name<br>,PARENT=lp-segment-name<br>,POINTER=PAIRED<br>,SOURCE=((real-lc-seg-name[,*D*,lc-db-name])) |

Table 37. Summary of Control Statements Used to Code Logical Parent Segments

| [Label] | Statement | Operand |
|---|---|---|
| | SEGM | NAME=segment-name<br>[,PARENT={*0* }]<br>{((parent-name[{,*SNGL*}]))}<br>{,DBLE}<br>[,BYTES={fixed-length }]<br>{max-var-length,min-var-length)}<br>[,POINTER={*TWIN* }]<br>{TWINBWD}<br>{NOTWIN }<br>[RULES=([{P}{P}{P}][{,FIRST}])]<br>{*L*}{*L*}{*L*} {,*LAST* }<br>{V}{V}{V} {,HERE }<br>[,COMPRTN=(routine-name[,*D*,INIT])] |
| | LCHILD(1) | NAME=(lc-segment-name[,db-name])<br>[,POINTER=*NONE*] |
| | LCHILD(2) | NAME=(lc-segment-name[,dbdname])<br>,POINTER={*SNGL*}<br>{DBLE}<br>,PAIR=virt-lc-seg-name<br>[RULES={FIRST}]<br>{*LAST* }<br>{HERE } |

(1)Unidirectional Logical Relationship (2)Bidirectional Logical Relationship

Subtopics:

- 2.4.2.1 Example of Physical DBDs with Logical Relationships

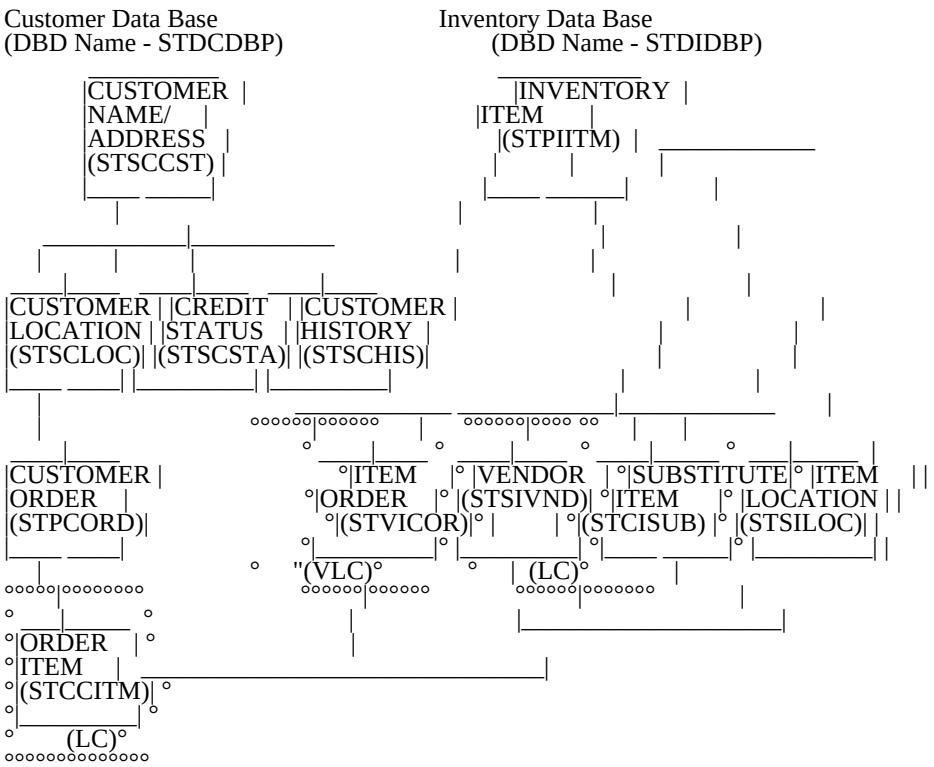
---

# 2.4.2.1 Example of Physical DBDs with Logical Relationships

Figure 13 shows how the physical DBDs for the Customer data base and the Inventory data base are extended to support logical relationships. Only those DBD statements are shown which are essential to the logical relationship function. Compare these extended DBDs with the basic DBD examples shown in Chapter 2, "Defining HDAM Data Bases."

Note the addition of the virtual logical child segment, ITEM ORDER, to the Inventory data base. Also, the SUBSTITUTE ITEM segment is defined as a logical child to eliminate the need for redundant data. In the HDAM DBD 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.




**Inventory Data Base**

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

```
DBD                                   X
    NAME=STDIDBP,         DATA BASE DESCRIPTION NAME      X
    °
    °
    °
DATASET                               X
```

```
            DD1=STDIDBC,          DLBL FILE NAME               X
            °
            °
            °
        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,SEQ,U),BYTES=6,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 FIGURE 10 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
* 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 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
```

Figure 13. Example of Physical DBDs Extended to Support Logical Relationships

---

# 2.4.3 Defining LOGICAL Data Bases

A LOGICAL DBD, based on one or more existing physical DBDs, defines a new view of logically related data bases. This view is always a hierarchical data structure.

Subtopics:



# 2.4.3.1 Input Structure and Rules

Figure 14 illustrates the rules for structuring a batch input stream to define a LOGICAL DBD. A separate input stream is required for each logical data base you define.

***To Define a LOGICAL Data Base . . .***

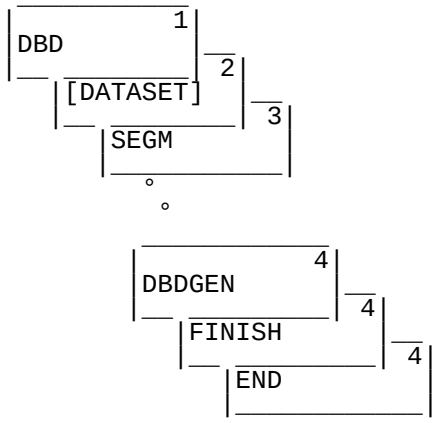



Figure 14. Batch Input Stream for LOGICAL Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.

2. Optionally, code one DATASET statement. If used, it must follow the DBD statement.

3. Code one SEGM statement for each segment type in the logical data base. Place them after the DATASET statement in the same sequence as the segments appear in the hierarchic order of the logical data base being defined. The maximum number of SEGM statements allowed is 255.

4. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

# 2.4.3.2 Control Statements Format

This section describes the control statements used to define a LOGICAL DBD.

Subtopics:



## 2.4.3.2.1 DBD Statement

Table 38. DBD Statement Syntax to Define LOGICAL Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | DBD name of logical data base. |
| | ,ACCESS= | **L O G I C A L** | LOGICAL must be specified. |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

*Coding Example*

```
DBD                                                          X
      NAME=STDCDBL,                                          X
      ACCESS=LOGICAL
```

**NAME=**
Specifies the name of the logical data base being defined.

**ACCESS=**
LOGICAL must be specified to define this DBD as a logical DBD.

## 2.4.3.2.2 DATASET Statement

Table 39. DATASET Statement Syntax to Define LOGICAL Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DATASET | LOGICAL | | The DATASET statement is optional. If used, code only the keyword 'LOGICAL'. No other keywords or parameters are required. |

*Coding Example*

```
DATASET                                                X
      LOGICAL
```

## 2.4.3.2.3 SEGM Statement

Table 40. SEGM Statement Syntax to Define LOGICAL Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Name of the logical segment. |
| | ,PARENT= | ° ° ° ° ° ° ° ° | Parent of the logical segment.(1) |
| | ,SOURCE= | ((° ° ° ° ° ° ° ° | Name of the source segment in the physical data base which represents this logical segment.(2) |
| | | ,_ _ _ _ | DATA (or D) for upward source compatibility with IMS/VS. |
| | | ,° ° ° ° ° ° ° ) | Name of the physical data base. (For concatenated segments only) |
| | | ,(_ _ _ _ _ _ _ _ | Name of the second source segment. |
| | | ,_ _ _ _ | DATA (or D) for upward source compatibility with IMS/VS. |
| | | ,_ _ _ _ _ _ _)) | Name of the physical data base. |

**Notes:**

1. The PARENT operand may be omitted for a root segment, or PARENT=0 may be specified.

2. A logical segment may be composed of one or two source segments. For one source segment, that is, a non-concatenated segment, specify only the first three parameters of the SOURCE operand. For two source segments, that is, a concatenated segment, specify all parameters shown.

*Coding Examples*

```
       SEGM                                                    X
             NAME=STPCORD,                                     X
             PARENT=STSCLOC,                                   X
             SOURCE=((STPCORD,                                 X
             D,                                                X
             STDCDBP))
       SEGM                                                    X
             NAME=STLCITM,                                     X
             PARENT=STPCORD,                                   X
             SOURCE=((STCCITM,                                 X
             D,                                                X
             STDCDBP),                                         X
             (STPIITM,                                         X
             D,                                                X
             STDIDBP))
```

**Note:** The first SEGM statement shown is an example of a non-concatenated segment. The second statement is an example of a concatenated segment.

**NAME=**

Specifies the name of this logical segment. It may or may not be the same segment name as in the physical DBD.

**PARENT=**

Specifies the name of the parent of this logical segment. The parent segment must be previously defined by a SEGM statement in this logical DBD.

If the logical segment being defined is the root segment of a logical data base, either omit this operand or specify PARENT=0.

**SOURCE=**

Specifies which segment(s) in a physical data base are used to represent the logical segment being defined. A logical segment may be composed of one or two source segments. For one source segment, that is, a non-concatenated segment, specify only the first three parameters of the SOURCE operand. For two source segments, that is, a concatenated segment, specify all parameters shown.

For non-concatenated segments, the first parameter defines the source segment.

For concatenated segments, the first parameter 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.

The second parameter, D (or DATA, the default), maintains upward source compatibility to IMS/VS.

The third parameter is the name of the physical DBD which contains the source segment defined in the first parameter.

The fourth parameter is used for the second part of the concatenated segment. It defines the name of the destination parent.

The fifth parameter, D (or DATA, the default), maintains upward source compatibility to IMS/VS.

The last parameter is the name of the physical DBD which contains the destination parent named in the fourth parameter.

---

## 2.4.3.2.4 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. They must be included at the end of your LOGICAL DBD generation input stream.

Table 41. DBDGEN, FINISH, END Statements to Define LOGICAL Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

 *© Copyright IBM Corp. 1981, 1989*

# 2.4.4 Control Statements Summary

Table 42 summarizes the DL/I control statements used to define a LOGICAL DBD.

Table 42. Summary of Control Statements Used to Code LOGICAL DBDs

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=logical-data-base-name,ACCESS=LOGICAL |
| | [DATASET | LOGICAL] |
| | SEGM | NAME=segment-name<br>[,PARENT={*0* }]<br>{parent-name}<br>,SOURCE=((first-source-seg,*D*,db-name)<br>[,(second-source-seg,*D*,db-name)]) |
| | DBDGEN | |
| | FINISH | |
| | END | |

Subtopics:

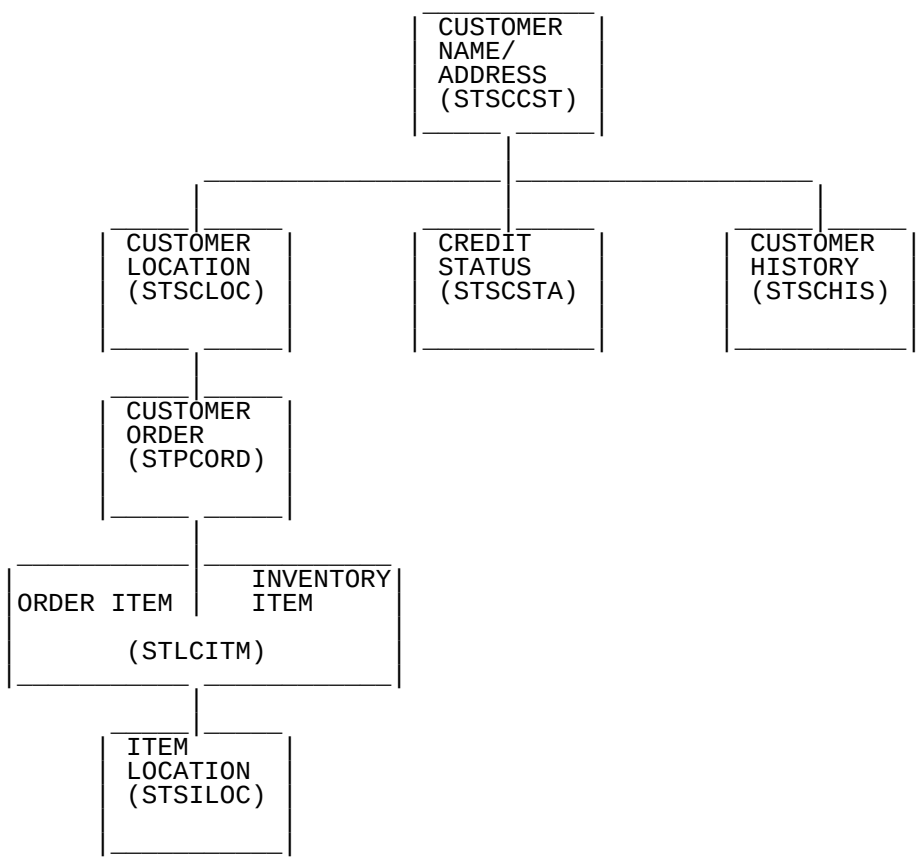
- 2.4.4.1 Examples of Logical DBDs

 *© Copyright IBM Corp. 1981, 1989*

# 2.4.4.1 Examples of Logical DBDs

Figure 15 shows the logical DBD for the Customer data base. Figure 16 shows the logical DBD for the Inventory data base.

Logical Customer Data Base
(DBD Name - STDCDBL)

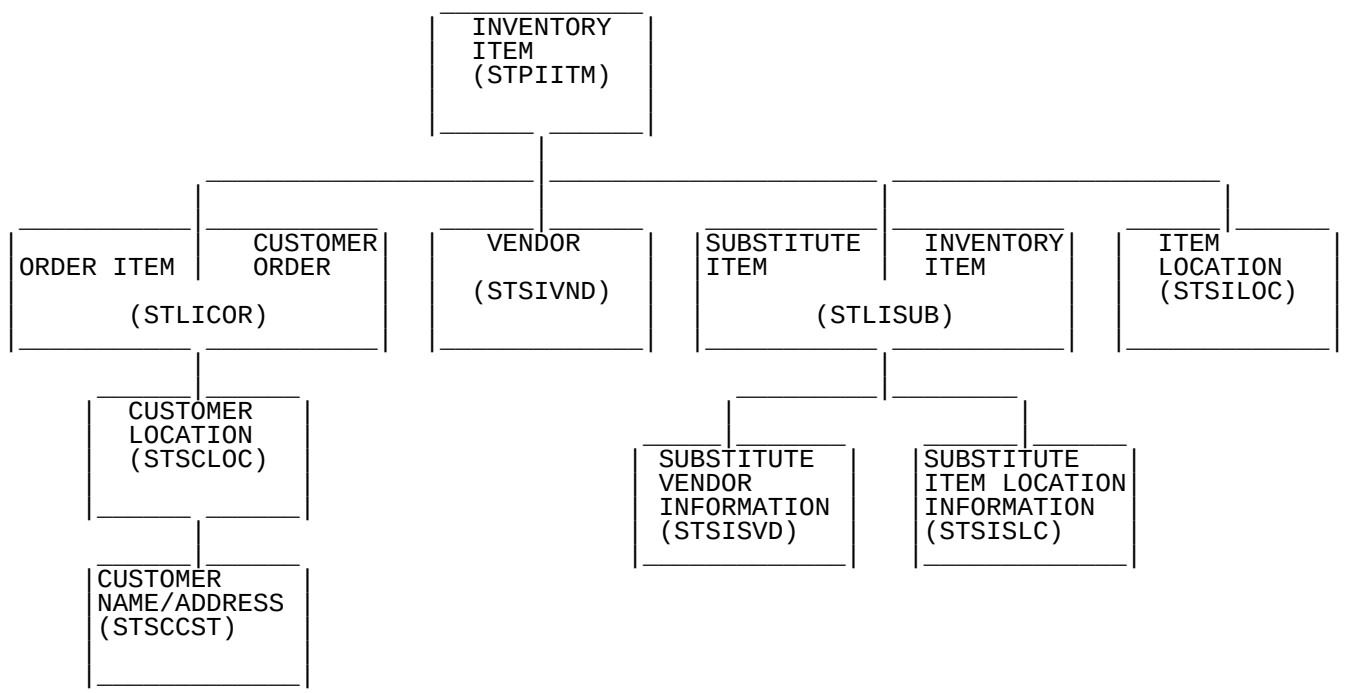



```
          DBD                                                        X
                NAME=STDCDBL,           LOGICAL DBD NAME             X
                ACCESS=LOGICAL          REQUIRED
          DATASET                                                    X
                LOGICAL                 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 15. Example of LOGICAL DBD for Customer Data Base

Logical Inventory Data Base
(DBD Name - STDIDBL)




```
      DBD                                                     X
      NAME=STDIDBL,           LOGICAL DBD NAME                X
      ACCESS=LOGICAL          REQUIRED
```

```
DATASET                                                          X
      LOGICAL                 OPTIONAL
SEGM                                                             X
      NAME=STPIITM,           SEGMENT NAME INVENTORY 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 16. Example of LOGICAL DBD for Inventory Data Base

---

# 2.5 Chapter 5. Defining Secondary Indexing in HDAM and HIDAM

This chapter describes how to implement DL/I's secondary indexing function in HDAM and HIDAM data bases. Two things must be done:

1. The HDAM or HIDAM *physical data bases* as previously described in Chapter 2, "Defining HDAM Data Bases" and Chapter 3, "Defining HIDAM (and Primary INDEX) Data Bases" respectively, must be extended. This means coding additional control statements and parameters to define those segment types involved in secondary indexing.

2. Another type of data base description called a secondary INDEX DBD must be defined.

**Note:** The secondary indexing function described in this chapter does not apply to HD data bases. (An HD data base is defined as such when you specify, in the DBD statement, that the access method is HD, that is, ACCESS=HD.) For information about defining secondary indexing in HD data bases, see Chapter 1, "Defining HD Data Bases."

Subtopics:



# 2.5.1 Defining Secondary Indexing in Physical Data Bases

Two segment types are used to establish secondary indexing in a physical data base. They are the *index target segment* and the *index source segment*. The coding requirements for these segment types are discussed separately in the rest of this section.

Subtopics:



# 2.5.1.1 Coding the Index Target Segment

Figure 17 is an extension of the batch input stream previously shown for HDAM and HIDAM data bases. You will note that this figure illustrates only those DL/I control statements that are used to define an index target segment. Instructions for coding these statements follow the illustration.

***To Define an Index Target Segment . . .***

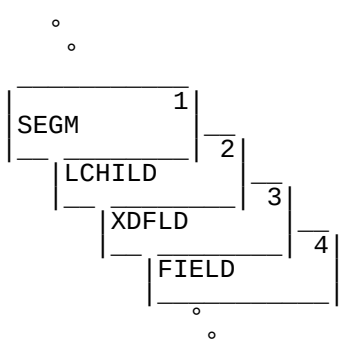



Figure 17. Batch Input Stream for Index Target Segments

1. Code a standard SEGM statement as previously described for HDAM and HIDAM data bases. No additional parameters are required. An index target segment is recognized as such because of the LCHILD and XDFLD statements that follow the SEGM statement. The index target segment cannot be a logical child or a dependent of a logical child.
2. Code one LCHILD statement to identify the index pointer segment in the secondary INDEX data base. Place the statement after the SEGM statement that defines the index target segment. The total number of LCHILD statements used in a single data description must not exceed 255.

   **Note:** Three types of LCHILD statements could occur below the root segment of a HIDAM data base. One type for the primary index, one for the definition of a logical child under its logical parent, and one for the definition of the index target segment. There could also be multiple occurrences of LCHILD statements for both logical relationships and secondary indexes. The first LCHILD statement of a HIDAM data base must be used to establish the primary index relationship.

3. Code one XDFLD statement. Put it after the related LCHILD statement. A maximum of 32 XDFLD statements are allowed per SEGM statement. The total number of XDFLD statements must not exceed 254 per DBD.
4. Code standard FIELD statements as previously described for HDAM and HIDAM data bases.

---

# 2.5.1.2 LCHILD Statement

This is the format of the LCHILD statement when it is used to reference index target segments in HDAM and HIDAM data bases:

Table 43. LCHILD Statement Syntax to Define Secondary Indexing

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| LCHILD | NAME= | (° ° ° ° ° ° ° °<br>,° ° ° ° ° ° ° ) | Name of pointer segment in secondary INDEX data base.<br>DBD name of secondary INDEX. |
| | ,POINTER= | **I N D X** | INDX must be specified. |

*Coding Example*

```
       LCHILD                                                  X
             NAME=(STIININ,                                    X
             STDIX1P),                                         X
             POINTER=INDX
```

**NAME=**

*The first parameter* specifies the name of the index pointer segment defined in the secondary INDEX data base.

*The second parameter* specifies the DBD name of the secondary INDEX data base which contains the index pointer segment.

**POINTER= (or PTR=)**

INDX identifies this LCHILD statement as an index type.

# 2.5.1.3 XDFLD Statement

This is the format of the XDFLD statement when used for secondary index relationships in HDAM and HIDAM data bases:

Table 44. XDFLD Statement Syntax to Define Secondary Indexing

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| XDFLD | NAME= | ° ° ° ° ° ° ° ° | Name of indexed field. |
| | ,SEGMENT= | _ _ _ _ _ _ _ _ | Name of index source segment.(1) |
| | ,SRCH= | (° ° ° ° ° ° ° °<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _) | Name of search field within source segment.(2)<br>Field name 2.<br>Field name 3.<br>Field name 4.<br>Field name 5. |
| | ,SUBSEQ= | (_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _) | Name of subsequence data field within source segment.(3)<br>Field name 2.<br>Field name 3.<br>Field name 4.<br>Field name 5. |
| | ,DDATA= | (_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _) | Name of duplicate data field within source segment.(4)<br>Field name 2.<br>Field name 3.<br>Field name 4.<br>Field name 5. |
| **Note:** The next two operands can be used to suppress creation of an index entry. A secondary index must not necessarily contain an entry for every index source segment occurrence. If both the NULLVAL and EXTRTN operands are specified, indexing of a segment is performed only if neither causes suppression of an index entry. | | | |
| | ,NULLVAL= | _ _ _ _ _ _ _ _ _ _ _ _ | Value on which to suppress creation of index pointer segment.(5) |
| | ,EXTRTN= | _ _ _ _ _ _ _ _ | Phase name of user-supplied index maintenance exit routine.(6) |

**Notes:**

1. The SEGMENT operand may be omitted if the index source segment is also the index target segment.
2. The SRCH operand requires at least one search field to be specified.

Optionally, up to four additional fields may be defined. If two or more names are included, they must be separated by commas and enclosed in parentheses.

3. The SUBSEQ operand is required if the search field data does not make up a unique sequence field.
4. The DDATA operand is optional. It is of use only when the index data base is processed as a data base itself.
5. The NULLVAL operand is optional. Use it only if you want to suppress the creation of index pointer segments based on a particular value in the search field(s).
6. The EXTRTN operand is optional. It may be used to specify the name of a user-supplied routine to suppress creation of an index pointer segment. See the operand description for details.

*Coding Examples*

```
       XDFLD                                                  X
             NAME=STXCMNA,                                    X
             SEGMENT=STSCCST,                                 X
             SRCH=STUCCNM,                                    X
             SUBSEQ=/CKSCCST
       XDFLD                                                  X
             NAME=STXCRDN,                                    X
             SEGMENT=STPCORD,                                 X
             SRCH=STQCODN,                                    X
             DDATA=/CKPCORD
```

**NAME=**

Specifies the name of the secondary index field. The name specified must be 1-8 alphameric characters. It must be unique among all field names specified for the above index target segment.

**SEGMENT=**

Identifies the index source segment for this secondary index relationship. The specified name must be the name of a segment that is a dependent of the index target segment. The segment name specified must not be a logical child segment. If this operand is omitted, the index target segment is assumed to be the index source segment.

**SRCH=**

Specifies which field or fields of the index source segment are to be used as the search field of a secondary index. You must specify at least one search field. Optionally, up to four additional fields may be named. (Be aware that the search field(s) specified must be defined for the index source segment by FIELD statements.) If two or more names are included, they must be separated by commas and enclosed in parentheses. The sequence in which the names are specified is the sequence in which the field values will be concatenated in the index pointer search field. The sum of the lengths of all SRCH fields (plus all SUBSEQ fields, if any) must be equal to the length of the key sequence field specified for the index pointer segment. The sum cannot exceed 236 bytes.

**SUBSEQ=**

Specifies which, if any, fields of the index source segment are to be used as subsequence fields of a secondary index. You may specify 1-5 field names. (Those specified must be defined in the index source segment by FIELD statements.) If two or more names are included, they must be separated by commas and enclosed in parentheses. The sequence in which the names are specified is the sequence in which field values will be concatenated in the index pointer segment subsequence fields. This operand is required only when the search field data does not make up a unique sequence field.

**Note:** /CK and /SX system-related fields may be referenced in the SUBSEQ operand.

**DDATA=**

Specifies which, if any, fields of the index source segment are to be maintained as duplicate data fields of a secondary index. You may specify 1-5 field names. (Those specified must be defined in the index source segment by FIELD statements.) If two or more names are included, they must be separated by commas and enclosed in parentheses. The sequence in which the names are specified is the sequence in which field values will be concatenated in the index pointer segment subsequence fields. This operand is optional. It is of use only when the index data base is processed as a data base itself.

**Note:** /SX system-related fields may be referenced in the DDATA operand.

**NULLVAL=**

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.

The value must be a 1-byte self-defining term (X'10',C'Z', 5 or B'00101101') or the words BLANK or ZERO. BLANK is equivalent to C'' or X'40'. ZERO is equivalent to X'00' or 0, but not C'0'. 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 NULLVAL=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 composes the search field is considered to be a separate packed value. For example, if the NULLVAL=X'9F' were specified in a case where the search field was composed of three 2-byte packed source fields, there would be no index entries with the search field value of X'999F999F999F' since these and only these would be suppressed.

Also, with the same NULLVAL=X'9F', if the search field were one 6-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.

**EXTRTN=**

Specifies the name of a user-supplied index maintenance exit routine that is used to suppress the creation of selected index pointer segments. The routine receives control whenever DL/I attempts to insert, delete or replace an index entry because of changes occurring in the indexed data base. This exit routine can inspect the affected index source segment and decide whether or not an index pointer segment should be generated.

---

# 2.5.1.4 Coding the Index Source Segment

Figure 18 is an extension of the batch input stream previously shown for HDAM and HIDAM data bases. You will note that this figure illustrates only those DL/I control statements that are used to define an index source segment.

***To Define an Index Source Segment . . .***

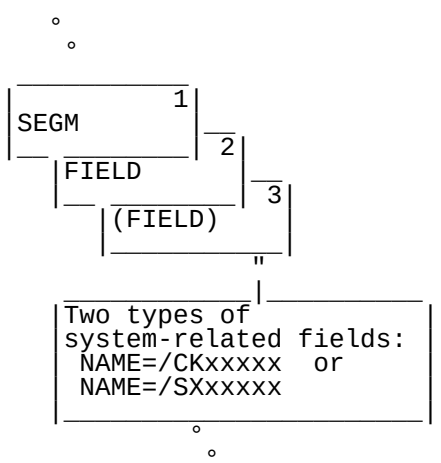



Figure 18. Batch Input Stream for Index Source Segments

1. Code a standard SEGM statement as previously described for HDAM and HIDAM data bases. The segment is recognized as an index source segment because it is defined hierarchically below the index target segment and because it is identified as an index source segment in the SEGMENT= operand of a preceding XDFLD statement.

   **Note:** Keep in mind that the index source segment can be the same segment as the index target segment, or it can be a dependent of the index target segment.

2. Code normal FIELD statements as previously described for HDAM and HIDAM data bases.
3. You may also code FIELD statements to describe system-related fields.

   System-related fields are optionally used for secondary indexing to force uniqueness of the secondary index entries. There are two types of system-related fields. Each requires a different format of the FIELD statement. A description of the two types and instructions for coding both formats follow this figure.

   **Note:** If you code a FIELD statement to describe a system-related field, it "must not" precede the FIELD statement that defines the sequence field for the corresponding statement.



---

# 2.5.1.5 FIELD Statement

In addition to the normal FIELD statements for the segment the FIELD statement may be used, in the definition of an index source segment, to force uniqueness of the secondary index entries. In this case, the FIELD statement is used to define a *system-related field* which may be appended to the key of the index entry by using the SUBSEQ operand of the XDFLD statement, thus making it unique.

There are two types of system-related fields: one refers to the index source segment's concatenated key, and the other refers to VSAM RBA (relative byte address).

1. When referring to the concatenated key, the format of the FIELD statement is:

Table 45. FIELD Statement Syntax to Define an Index Source Segment

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| FIELD | NAME= | **/ C K** ° ° ° ° ° | Name of system-related field: /CK followed by 1-5 alphameric characters. |
| | ,BYTES= | ° ° ° | Length of system-related field. |
| | ,START= | ° ° ° | Numeric starting position. |

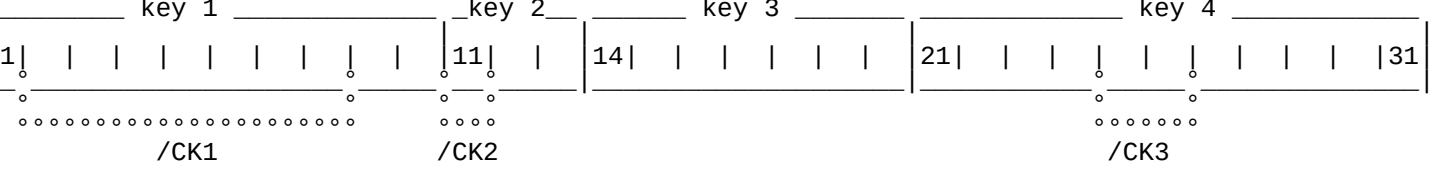
**NAME=**
Specifies the name of the system-related field. This field type is made up of all or a portion of the concatenated key of the index source segment. The name can be up to eight alphameric characters long, but must begin with the characters /CK. The fourth through eighth characters permit unique identification of the field. More than one /CKxxxxx field can be specified for an index source segment. They may be non-contiguous or overlap each other.

**BYTES=**
Specifies the length of this system-related field. It is specified in bytes and can be either the complete length of the concatenated key, or a portion of it.

**START=**
Specifies the starting position of this system-related field. If the field is made up of all the concatenated key, the start position is 1. If the field is made up of only a portion of the concatenated key, the start position of the portion is relative to the beginning of the key.

***An Example of Concatenated Key . . .***

Assume the following concatenated key of an index source segment:




If the uniqueness of the secondary index key has to be achieved by bytes 2-8 of the first or root key, byte 1 of the second key, and bytes 5-6 of the fourth key, the FIELD statements specifying this are as follows:

```
FIELD                                                  x
      NAME=/CK1,                                       x
      BYTES=7,                                         x
      START=2
FIELD                                                  x
      NAME=/CK2,                                       x
      BYTE=1,                                          x
      START=11
FIELD                                                  x
      NAME=/CK3,                                       x
      BYTES=2,                                         x
      START=25
```

The three system-related fields defined can then be specified for use in the subsequence and duplicate data fields of index pointer segments. This is done by including their /CK names in the SUBSEQ operand of an XDFLD statement.

---

2. When referring to the index source segment's VSAM RBA, the format of the
FIELD statement is:

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| FIELD | NAME= | **/ S X** ° ° ° ° ° | Name of system-related field: /SX followed by 1-5 alphameric characters. |

**NAME=**
Specifies the name of the system-related field. The name can be up to eight alphameric characters long, but must begin with the characters /SX. The fourth through eighth characters ensure uniqueness of sequence field keys in a secondary index. Only one /SX field can be specified for a segment.

---

# 2.5.2 Control Statements Summary

The next two figures summarize the DL/I control statements used to code secondary indexing in a physical DBD.

Table 46 shows the LCHILD and XDFLD statements when used to code an index target segment. Table 47 shows the FIELD statement formats when used to code an index source segment.

Table 46. Summary of Control Statements Used to Code Index Target Segments

| [Label] | Statement | Operand |
|---|---|---|
| | LCHILD | NAME=(index-ptr-seg-name,index-dbd-name)<br>,POINTER=INDX |
| | XDFLD | NAME=xdfld-name<br>[,SEGMENT=index-source-seg-name]<br>,SRCH={srch-fld }<br>{(srch-fld1,srch-fld2[,...,srch-fld5])}<br>[,SUBSEQ={sub-seg-fld }]<br>{(ss-fld1,ss-fld2[,...,ss-fld5])}<br>[,DDATA={dup-data-fld }]<br>{(dd-fld1,dd-fld2[,...,dd-fld5])}<br>[,NULLVAL=value]<br>[,EXTRTN=routine-name] |

Table 47. Summary of Control Statements Used to Code Index Source Segments

| [Label] | Statement | Operand |
|---|---|---|
| | FIELD(1) | NAME=/CKname<br>,BYTES=bytes<br>,START=position |
| | FIELD(2) | NAME=/SXname |

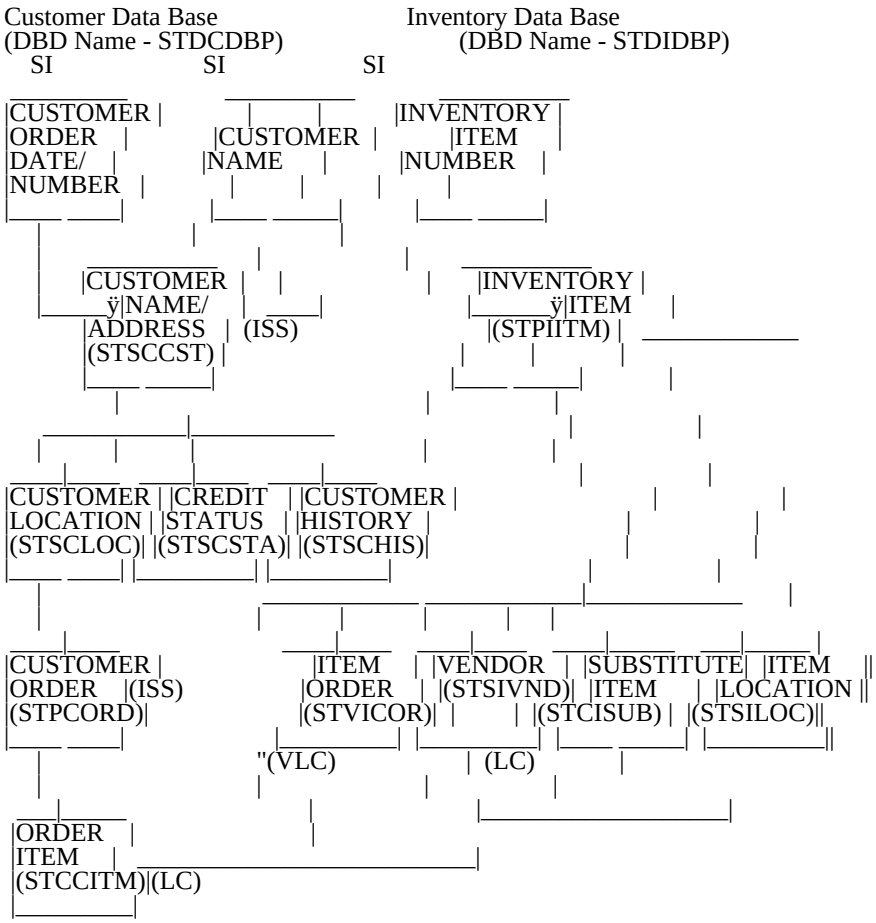
(1)Use this format when referring to the index source segment's concatenated key.
(2)Use this format when referring to the index source segment's VSAM RBA (relative byte address).

Subtopics:

- 2.5.2.1 Example of Physical DBDs with Secondary Indexing

# 2.5.2.1 Example of Physical DBDs with Secondary Indexing

Figure 19 shows how the physical DBDs for the Customer data base and the Inventory data base are extended to support secondary indexing.




**Note:** The DBD names for the three secondary indexes shown in this illustration are as follows:

- Inventory Item Number = STDIX1P
- Customer Name = STDCX1P
- Customer Order Date/Number = STDCX2P

The DBDs for these indexes are not included as part of this figure. You can find them in Figure 21.

**Inventory Data Base**

```
* IN THIS SECTION WE WILL REPEAT THE FIGURE 20
* 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
      DBD   NAME=STDIDBP,ACCESS=HDAM,RMNAME=(DLZHDC30,3,100,400)
      DATASET DD1=STDIDBC,DEVICE=3380,BLOCK=2048,SCAN=2
      SEGM  NAME=STPIITM,PARENT=0,BYTES=56,POINTER=TWIN,RULES=(PPV)
      FIELD NAME=(STQIINO,SEQ,U),BYTES=6,START=1,TYPE=C
* THE FOLLOWING LCHILD AND XDFLD MACROS ARE FOR THE SECONDARY INDEX
      LCHILD                                        X
          POINTER=INDX,          SECONDARY INDEX              X
          NAME=(STIININ,         SEGMENT NAME IN INDEX DBD      X
          STDIX1P)             DBD NAME OF INDEX DBD
      XDFLD                                         X
          NAME=STXININ,          INDEXED FIELD NAME           X
          SEGMENT=STPIITM,       SOURCE SEGMENT OF INDEX       X
          SRCH=STQIINO           INDEXED DATA WITHIN SOURCE SEGM
* THERE ARE NO FURTHER CHANGES IN THIS DBD FOR THE SECONDARY INDEX
      LCHILD POINTER=DBLE,NAME=(STCCITM,STDCDBP),PAIR=STVICOR,      X
          RULES=LAST
      LCHILD POINTER=NONE,NAME=(STCISUB,STDIDBP)
      FIELD NAME=STFIIDS,BYTES=25,TYPE=C   ITEM DESCRIPTION
      FIELD NAME=STFIIQH,BYTES=6,TYPE=C    QUANTITY ON HAND
      FIELD NAME=STFIIQO,BYTES=6,TYPE=C    QUANTITY ON ORDER
      FIELD NAME=STFIIQR,BYTES=6,TYPE=C    QUANTITY ON RESERVE
      FIELD NAME=STFIIPR,BYTES=6,TYPE=C    COST PER ITEM
      FIELD NAME=STFIIUN,BYTES=1,TYPE=C    UNIT OF ISSUE
      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
```

**Customer Data Base**

```
* IN THIS SECTION WE WILL REPEAT THE
* FIGURE 20 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
      DBD   NAME=STDCDBP,ACCESS=HDAM,RMNAME=(DLZHDC10,3,100,600)
      DATASET DD1=STDCDBD,DEVICE=3380,BLOCK=2048,SCAN=2
      SEGM NAME=STSCCST,PARENT=0,BYTES=110,POINTER=TWIN
      FIELD NAME=(STQCCNO,SEQ,U),BYTES=6,START=1,TYPE=C
      FIELD                                         X
          NAME=STUCCNM,          INDEX SOURCE SEG SEARCH FLD     X
          BYTES=25,              FIELD LENGTH                 X
          START=7,               WHERE IT STARTS IN SEGMENT      X
          TYPE=C                 ALPHAMERIC DATA


* THE FOLLOWING LCHILD AND XDFLD MACROS ARE FOR THE SECONDARY INDEXES
      LCHILD                                        X
          POINTER=INDX,          SECONDARY INDEX              X
          NAME=(STICMNA,         SEGMENT NAME IN INDEX DBD       X
          STDCXIP)               DBD NAME OF INDEX DBD
      XDFLD                                         X
          NAME=STXCMNA,          INDEXED FIELD NAME           X
          SEGMENT=STSCCST,       SOURCE SEGMENT OF INDEX       X
          SRCH=STUCCNM,          INDEXED DATA WITHIN SOURCE SEGM X
          SUBSEQ=/CKSCCST        INDEX SUBSEQUENCE FIELD
      FIELD                                         X
          NAME=/CKSCCST,         CONCATENATED KEY             X
          BYTES=6,               FIELD LENGTH                 X
          START=1                WHERE IT STARTS IN SEGMENT
      LCHILD                                        X
          POINTER=INDX,          SECONDARY INDEX              X
          NAME=(STIRCRDN,        SEGMENT NAME IN INDEX DBD       X
          STDCX2P)               DBD NAME OF INDEX DBD
      XDFLD                                         X
          NAME=STXCRDN,          INDEXED FIELD NAME           X
          SEGMENT=STPCORD,       SOURCE SEGMENT OF INDEX       X
          SRCH=STQCODN,          INDEXED DATA WITHIN SOURCE SEGM X
          DDATA=/CKPCORD         FIELD NAME OF CONCATENATED KEY
      SEGM  NAME=STSCLOC,PARENT=STSCCST,BYTES=106,POINTER=TWINBWD
```

```
        FIELD NAME=(STQCLNO,SEQ,U),BYTES=6,START=1,TYPE=C
        SEGM  NAME=STPCORD,PARENT=STSCLOC,BYTES=51,POINTER=TWINBWD,   X
            RULES=(PPV)
        FIELD NAME=(STQCODN,SEQ,U),BYTES=12,START=1,TYPE=C
* THE FOLLOWING FIELD MACRO IS FOR THE DUPLICATE DATA
* FIELD THAT WILL BE CARRIED IN THE SECONDARY INDEX
* DATA BASE. IT DEFINES THE FIRST 12 BYTES OF THE ORDER
* SEGMENT FULLY CONCATENATED KEY. THE FIRST 6 BYTES ARE
* CUSTOMER NUMBER (STQCCNO) AND NEXT 6 ARE LOCATION
* NUMBER (STQCLNO). HAVING THIS DATA CARRIED IN SECONDARY
* INDEX WILL ALLOW US TO HAVE CUSTOMER AND LOCATION
* AVAILABLE TO US WHEN PROCESSING THE SECONDARY INDEX
* BY ITSELF. ALSO DL/I AUTOMATICALLY MAINTAINS THIS
* DATA WHEN PROCESSING THE CUSTOMER DATA BASE.
        FIELD                                   X
            NAME=/CKPCORD,         MUST START WITH /CK          X
            BYTES=12,            LENGTH DESIRED               X
            START=1              STARTING AT THIS POSITION
* THERE ARE NO FURTHER CHANGES IN THIS DBD FOR THE SECONDARY INDEX
        SEGM  NAME=STCCITM,PARENT=((STPCORD),(STPIITM,V,STDIDBP)),    X
            BYTES=38,POINTER=TWINBWD,RULES=(PPV)
        FIELD NAME=STKCIIN,BYTES=6,TYPE=C
        FIELD NAME=(STQCILI,SEQ,U),BYTES=2,START=7,TYPE=C
        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 19. Example of Physical DBDs Extended to Support Secondary Indexing

---

# 2.5.3 Defining the Secondary INDEX Data Base

This section discusses the secondary INDEX data base. This data base contains the *index pointer segments* that contain pointers in their prefix to the index target segments.

Subtopics:



# 2.5.3.1 Input Structure and Rules

Figure 20 illustrates the rules for structuring a batch input stream to define a secondary INDEX data base.

***To Define a Secondary INDEX Data Base . . .***

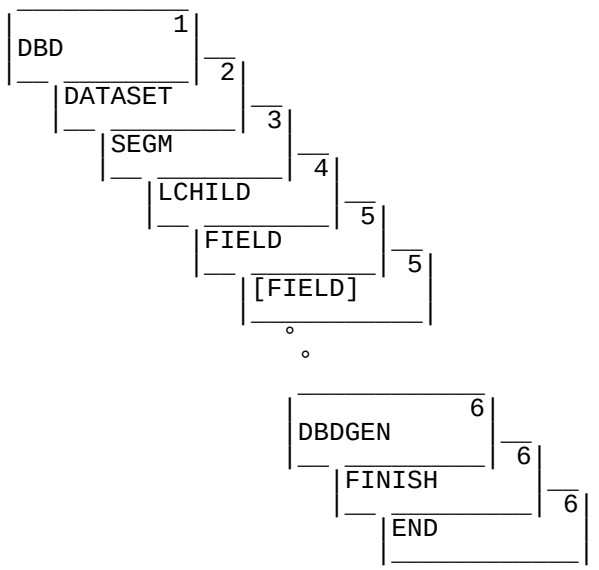



Figure 20. Batch Input Stream for Secondary INDEX Data Base

1. Code one DBD statement. It must be the first statement in your input stream.

2. Code one DATASET statement. It must follow the DBD statement.

3. Code one SEGM statement to identify the index pointer segment. The SEGM statement must follow the DATASET statement.

4. Code one LCHILD statement to identify the index target segment. Place it after the SEGM statement.

5. Code one FIELD statement to identify a unique sequence field. Place it after the LCHILD statement. Optionally, additional fields may also be defined in INDEX data bases.

6. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

 *© Copyright IBM Corp. 1981, 1989*

# 2.5.3.2 Control Statements Format

```
This section explains how to code DL/I control statements to define a
secondary INDEX data base.
```

 *© Copyright IBM Corp. 1981, 1989*

# 2.5.3.3 DBD Statement

The format of the DBD control statement when used to define a secondary INDEX data base is:

Table 48. DBD Statement Syntax to Define a Secondary INDEX Data Base

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Name of INDEX data base. |
| | ,ACCESS= | **I N D E X** | INDEX identifies this DBD as a secondary INDEX DBD. |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

*Coding Example*

```
DBD                                                    X
      NAME=STDIX1P,                                    X
      ACCESS=INDEX
```

**NAME=**
Specifies the name of the secondary INDEX data base. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Here must specify INDEX. It identifies this DBD as an INDEX DBD.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

# 2.5.3.4 DATASET Statement

The format of the DATASET control statement when used to define the characteristics of a secondary INDEX data base is:

Table 49. DATASET Statement Syntax to Define Secondary INDEX Data Base

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Name of data set; 1-7 alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | _ _ _ _ | Number of VSAM logical records in one control interval (CI).(1) |
| | ,RECORD= | _ _ _ _ | Logical record length.(2) |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure.
2. The RECORD operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure.

*Coding Example*

```
DATASET                                                    X
      DD1=STDIX1C,                                         X
      DEVICE=3380
```

**DD1**

Specifies the name you want used as the symbolic VSAM data set name for this secondary INDEX data base. The name must be 1-7 alphameric characters long. This name is required for cross-reference to the DASD label information (DLBL).

**DEVICE=**

Identifies the physical device type used for storage of this data base. You must use one of the following:

FBA, 3380, 3375, 3350, 3340, 3330, or 2314.

**BLOCK=**

Specifies the number of VSAM logical records to be contained within

one control interval. The BLOCK operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure using a control interval size of 2048 bytes wherever possible.

**RECORD=**
Specifies the VSAM logical record length. The RECORD operand is optional. If omitted, the length is calculated during the DBDGEN procedure.

**For more information...**

about determining the size in bytes of the BLOCK and RECORD operands see chapter 4 in *DL/I DOS/VS Data Base Administration*.

# 2.5.3.5 SEGM Statement

The format of the SEGM control statement when used to define a secondary INDEX data base is:

Table 50. SEGM Statement Syntax to Define Secondary INDEX Data Base

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Name of index pointer segment. |
| | ,BYTES= | _ _ _ _ | Length of index pointer segment.(1) |

**Notes:**

1. The BYTES operand is optional. If omitted, DL/I will calculate the segment length during DBDGEN procedure.

*Coding Example*

```
SEGM                                                    X
      NAME=STIININ,                                     X
      BYTES=6
```

**NAME=**
Specifies the name of the index pointer segment.

**BYTES=**
Specifies the length of the data portion of the index pointer segment. The length must be long enough to contain all search fields, subsequence fields, and duplicate data fields.

**For more information...**

about determining the size in bytes of the BYTES operand, see chapter 5 in *DL/I DOS/VS Data Base Administration*.

The BYTES operand is optional. If omitted, the segment size is calculated during the DBDGEN procedure.

# 2.5.3.6 LCHILD Statement

The format of the LCHILD statement when used to define a secondary INDEX data base is:

Table 51. LCHILD Statement Syntax for Secondary INDEX Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| LCHILD | NAME= | (° ° ° ° ° ° ° °<br>,° ° ° ° ° ° ° ) | Name of index target segment.<br>Name of its DBD. |
| | ,POINTER= | **S N G L** | SNGL. |
| | ,INDEX= | ° ° ° ° ° ° ° ° | Name of the indexed field. |

*Coding Example*

```
LCHILD                                                  X
      NAME=(STPIITM,                                    X
      STDIDBP),                                         X
      POINTER=SNGL,                                     X
      INDEX=STXININ
```

**NAME=**

The first parameter specifies the name of the index target segment defined in the HDAM or HIDAM data base.

The second parameter specifies the name of the HDAM or HIDAM data base which contains the index target segment.

**POINTER=**

SNGL must be specified. It causes a 4-byte direct address pointer to be put in the prefix of the index pointer segment. It points to the index target segment.

**INDEX=**

Specifies the name of the field being indexed. This operand must correspond to the NAME parameter for the XDFLD statement in the HDAM or HIDAM data base of this index relationship.

# 2.5.3.7 FIELD Statement

When defining a secondary INDEX data base, you must code one FIELD statement using the format shown here. This required FIELD statement defines a unique sequence field in the INDEX data base.

Optionally, additional fields may also be defined in the INDEX data base. These fields may be useful, for example, if you plan to use the INDEX data base as a separate processible data base. If these additional fields are desired, use the FIELD statement format previously shown for HDAM or HIDAM data bases to define them.

Table 52. FIELD Statement Syntax for Secondary INDEX Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (° ° ° ° ° ° ° ° ,**S E Q**) | Field name. SEQ must be specified. |
| | ,BYTES= | _ _ _ | Field length; 1-236 bytes.(1) |
| | ,TYPE= | _ | Field data type.(2) |

**Notes:**

1. See "Rules and Restrictions" in the BYTES operand description.
2. The TYPE operand is optional. If omitted, TYPE=C is assumed.

*Coding Example*

```
          FIELD                                                X
                NAME=(STYIINO,SEQ,U),                          X
                BYTES=6,                                       X
                TYPE=C
```

**NAME=**

The first parameter specifies the name of this field. The name can be 1-8 alphameric characters long.

SEQ must be specified to identify this field as a sequence field.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 236. The length of this field must be equal to the sum of the lengths of the search and subsequent fields identified on the corresponding XDFLD statement in the HDAM or HIDAM data base.

***Rules and Restrictions...***

- The BYTES operand must be specified for field data types X, P, C, and Z (see TYPE operand for field data types).

- The BYTES operand 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 operand for field data types E, D, and L. These data types have implicit lengths of 4, 8, and 16, respectively.

**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.

**Notes:**

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

2. It is recommended that you explicitly specify the field 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 during PSB generation.

---

# 2.5.3.8 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. They must be included at the end of your DBD generation input stream.

Table 53. DBDGEN, FINISH, END Statements for Secondary INDEX Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

 *© Copyright IBM Corp. 1981, 1989*

# 2.5.4 Control Statements Summary

Table 54 summarizes the DL/I control statements used to define secondary INDEX data bases. Only the control statements and operands pertinent to secondary INDEX are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 54. Summary of Control Statements Used to Code Secondary DBDs

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS=INDEX<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DD1=filename(1)<br>,DEVICE=device-type<br>[,BLOCK=(number-of-KSDS-records-per-CI)]<br>[,RECORD=(KSDS-record-length)] |
| | SEGM | NAME=segment-name<br>[,PARENT=*0*]<br>[,BYTES=segment-length] |
| | LCHILD | NAME=(index-target-seg-name,dbd-name)<br>[,POINTER=*SNGL*]<br>,INDEX=indexed-field-name |
| | FIELD | NAME=(field-name,SEQ)<br>[,BYTES=length<br>[,TYPE=data-type] |

| | | |
|---|---|---|
| | DBDGEN | |
| | FINISH | |
| | END | |

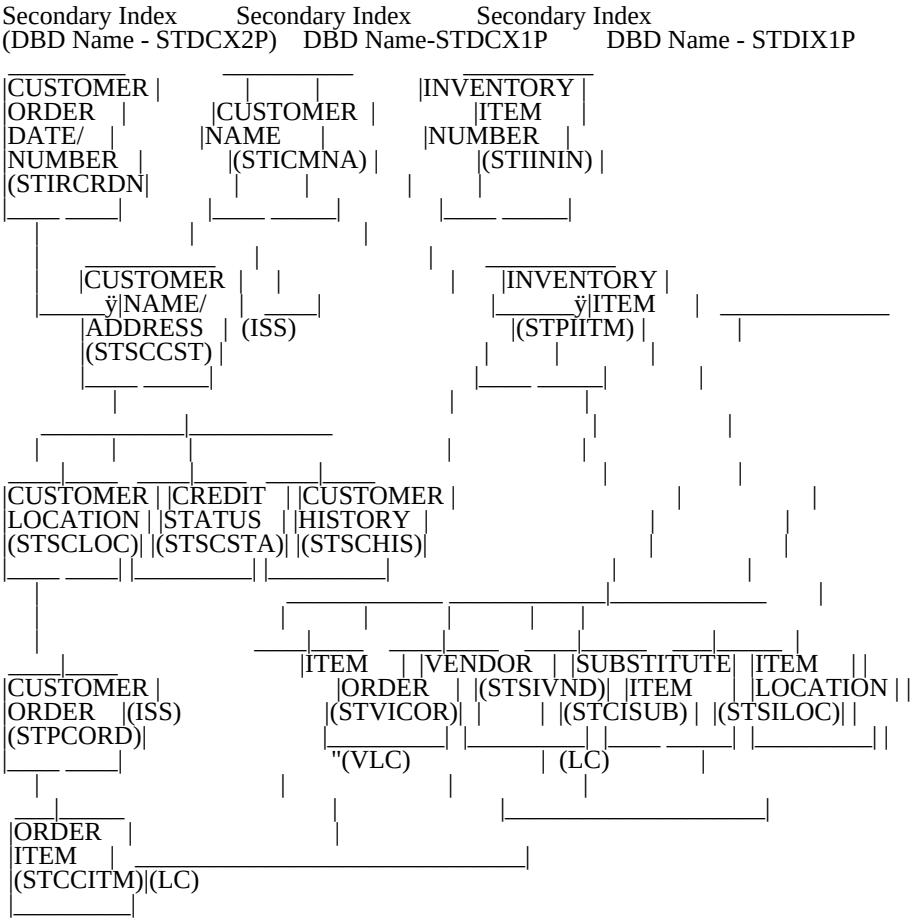
Subtopics:

- 2.5.4.1 Examples of a Secondary INDEX Data Base

---

# 2.5.4.1 Examples of a Secondary INDEX Data Base

Figure 21 shows three examples of a secondary INDEX data base. These examples are associated with the physical DBDs for the Customer data base and the Inventory/Inventory data base that are shown in Figure 19 in topic 2.5.2.1.




**Note:** The DBDs for the Customer and Inventory data bases shown in this illustration are not included as part of this figure. You can find them in Figure 19.

**Secondary Index DBD - Customer Order Date/Number**

```
DBD                                    X
     NAME=STDCX2P,         DATA BASE DESCRIPTION NAME     X
     ACCESS=INDEX          THIS IS AN INDEX
DATASET                                X
     DD1=STDCX2C,          DLBL FILE NAME                 X
     DEVICE=3380           DISK DEVICE
SEGM                                   X
     NAME=STIRCRDN,        SEGMENT NAME OF INDEX          X
     PARENT=0,             IT IS A ROOT SEGMENT           X
```

```
    BYTES=24              LENGTH INCL. INDEX & DUP DATA
LCHILD                                X
    NAME=(STSCCST,        TARGET SEGMENT CUST. NAME/ADDR  X
    STDCDBP),             FOUND IN THE CUSTOMER DATA BASE X
    INDEX=STXCRDN,        INDEXED FIELD NAME            X
    POINTER=SNGL          SPECIFIES THIS IS INDEX PTR SEG
FIELD                                 X
    NAME=(STYCRDS,SEQ,U),   UNIQUE KEY FIELD (ALSO THE INDX)X
    BYTES=12,             FIELD LENGTH                  X
    START=1,              WHERE IN SEGMENT IT STARTS      X
    TYPE=C                ALPHAMERIC DATA
DBDGEN                    TO MARK END OF DBD
FINISH                    FOR SOURCE COMPAT WITH IMS/VS
END
```

**Secondary Index DBD - Customer Name**

```
DBD                                   X
    NAME=STDCX1P,         DATA BASE DESCRIPTION NAME      X
    ACCESS=INDEX          THIS IS AN INDEX
DATASET                               X
    DD1=STDCX1C,          DLBL FILE NAME                X
    DEVICE=3380           DISK DEVICE
SEGM                                  X
    NAME=STICMNA,         SEGMENT NAME OF THE INDEX       X
    PARENT=0,             IT IS A ROOT SEGMENT          X
    BYTES=31              LENGTH INCL. INDEX & DUP DATA
LCHILD                                X
    NAME=(STSCCST,        TARGET SEGMENT CUST. NAME/ADDR  X
    STDCDBP),             FOUND IN THE CUSTOMER DATA BASE X
    INDEX=STXCMNA,        INDEXED FIELD NAME            X
    POINTER=SNGL          SPECIFIES THIS IS INDEX PTR SEG
FIELD                                 X
    NAME=(STYCMNS,SEQ,U),   UNIQUE KEY FIELD (ALSO THE INDX)X
    BYTES=31,             FIELD LENGTH                  X
    START=1,              WHERE IN SEGMENT IT STARTS      X
    TYPE=C                ALPHAMERIC DATA
DBDGEN                    TO MARK END OF DBD
FINISH                    FOR SOURCE COMPAT WITH IMS/VS
END
```

**Secondary Index DBD - Inventory Item Number**

```
DBD                                   X
    NAME=STDIX1P,         DATA BASE DESCRIPTION NAME      X
    ACCESS=INDEX          THIS IS AN INDEX
DATASET                               X
    DD1=STDIX1C,          DLBL FILE NAME                X
    DEVICE=3380           DISK DEVICE
SEGM                                  X
    NAME=STIININ,         SEGMENT NAME OF THE INDEX       X
    PARENT=0              IT IS A ROOT SEGMENT          X
    BYTES=6               LENGTH
LCHILD                                X
    NAME=(STPIITM,        TARGET SEGMENT ITEM INFORMATION X
    STDIDBP),             FOUND IN THE ITEM DATA BASE     X
    INDEX=STXININ,        INDEXED FIELD NAME            X
    POINTER=SNGL          SPECIFIES THIS IS INDEX PTR SEG
FIELD                                 X
    NAME=(STYIINO,SEQ,U),   UNIQUE KEY FIELD(ALSO THE INDX) X
    BYTES=6,              FIELD LENGTH                  X
    START=1,              WHERE IN SEGMENT IT STARTS      X
    TYPE=C                ALPHAMERIC DATA
DBDGEN                    TO MARK END OF DBD
FINISH                    FOR SOURCE COMPAT WITH IMS/VS
END
```

Figure 21. Example of Secondary Index DBDs

---

---

# 2.6 Chapter 6. Defining SHISAM and HISAM Data Bases

This chapter describes how to define a data base description (DBD) for SHISAM and HISAM data bases.

***For a SHISAM data base description,*** you must prepare an input stream of several DL/I control statements specifying:

- The symbolic VSAM data set (KSDS) name for this SHISAM data base.
- One segment type for the data base (root segment only).
- The fields for the segment type, one of which identifies a sequence field.

***For a HISAM data base description,*** you must prepare an input stream of several DL/I control statements specifying:

- The symbolic VSAM name for one primary data set (KSDS).
- The symbolic VSAM name for one overflow data set (ESDS).
- The segments types for the data base.
- The fields for each segment type, one of which identifies a sequence field in the root segment.

Subtopics:



# 2.6.1 Input Structure and Rules

Figure 22 and Figure 23 illustrate the rules for structuring batch input streams defining SHISAM and HISAM data bases, respectively. These figures show the types of control statements required, the number of each type that may be specified, and their specific order of sequence. A separate input stream is required for each data base you define.

***To Define a SHISAM Data Base . . .***

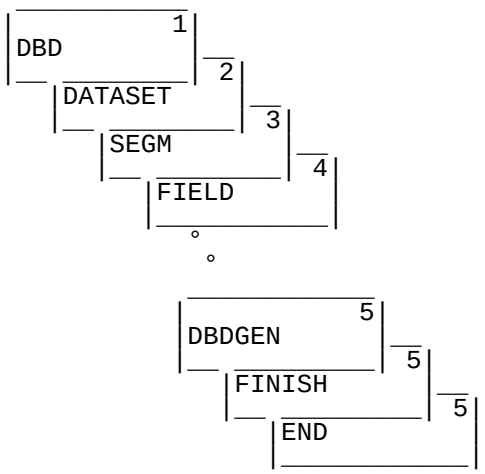



Figure 22. Batch Input Stream for SHISAM Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.

2. Code one DATASET statement. It must follow the DBD statement.

3. Code one SEGM statement. It must follow the DATASET statement.

4. Code at least one FIELD statement. Use it to identify a "sequence field" within the root segment. Optionally, you may define up to 255 fields. Code one FIELD statement for each field you define. The FIELD statement(s) must follow the SEGM statement.

5. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

***To Define a HISAM Data Base . . .***

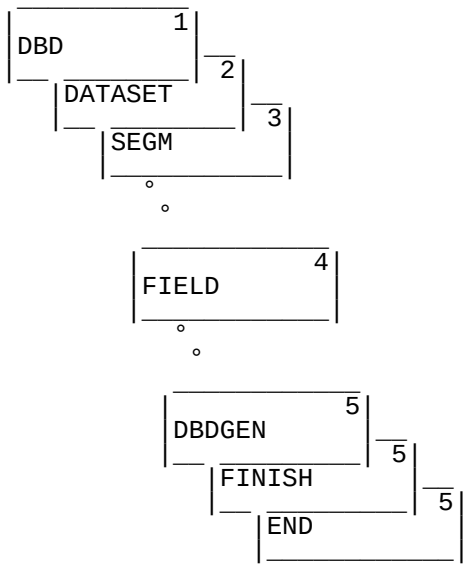



Figure 23. Batch Input Stream for HISAM Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.

2. Code one DATASET statement. It must follow the DBD statement.

3. Code one SEGM statement for each segment type in your data base. Place them after the DATASET statement in the same sequence as the segments appear in the hierarchic order of the data base being defined. The maximum number of SEGM statements allowed per data base is 255.

4. Code at least one FIELD statement. Use it to identify a "sequence field" within the root segment. Optionally, you may define a maximum of 255 fields per segment type and 1020 fields in a HISAM data base. Use one FIELD statement for each field you define. The FIELD statements must follow the SEGM statement defining the segment containing those fields.

5. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

---

# 2.6.2 Control Statements Format

This section explains how to code DL/I control statements used to define SHISAM and HISAM data bases. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introduction.

Subtopics:



## 2.6.2.1 DBD Statement

The format of the DBD statement when used to define a SHISAM or HISAM data base is:

Table 55. DBD Statement Syntax for SHISAM/HISAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DBD | NAME= | ° ° ° ° ° ° ° | Name of INDEX data base. |
| | ,ACCESS= | ° ° ° ° ° ° ° | SHISAM or HISAM. |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

*Coding Example*

```
DBD                                                    X
      NAME=SKILINV,                                    X
      ACCESS=HISAM,                                    X
      IMSCOMP=NO
```

**NAME=**
Specifies the name of the data base. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Specifies the access method used for the data base. Here you must specify either SHISAM or HISAM.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned, or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

# 2.6.2.2 DATASET Statement

Two formats of the DATASET statement are shown below. Use the first format, if you are defining a SHISAM data base. Use the second format, if you are defining a HISAM data base.

Table 56. DATASET Statement Syntax for SHISAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Symbolic VSAM data set name for this SHISAM data base; 1-7 alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | _ _ _ _ | Number of VSAM logical records in one control interval (CI).(1) |
| | ,RECORD= | _ _ _ _ | Logical record length.(2) |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate a value during the DBDGEN procedure.
2. The RECORD operand is optional If omitted, DL/I will calculate a value during the DBDGEN procedure.

*Coding Example*

```
DATASET                                                    X
      DD1=EMPNAME,                                         X
      DEVICE=3380,                                         X
      BLOCK=2,                                             X
      RECORD=1024
```

Table 57. DATASET Statement Syntax for HISAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DD1= | ° ° ° ° ° ° ° | Name of VSAM primary data set; 1-7 alphameric characters. |
| | ,OVFLW= | ° ° ° ° ° ° ° | Name of VSAM overflow data set; 1-7 |

| | | | |
|---|---|---|---|
| | | | alphameric characters. |
| | ,DEVICE= | ° ° ° ° | Physical device type used for storage of this data base. |
| | ,BLOCK= | (_ _ _ _<br>,_ _ _ _) | Maximum number of logical records per primary data set control interval (CI).(1)<br>Maximum number of logical records per overflow data set control interval (CI).(1) |
| | ,RECORD= | (_ _ _ _<br>,_ _ _ _) | Logical record length in primary data set.(2)<br>Logical record length in overflow data set.(2) |

**Notes:**

1. The BLOCK operand is optional. If omitted, DL/I will calculate the values during the DBDGEN procedure.
2. The RECORD operand is optional If omitted, DL/I will calculate the values during the DBDGEN procedure.

*Coding Example*

```
       DATASET                                                  X
             DD1=SKHISAM,                                       X
             OVFLW=HISAMOV,                                     X
             DEVICE=3380,                                       X
             BLOCK=(2,2),                                       X
             RECORD=(1024,1024)
```

**DD1=**

For SHISAM, this operand specifies the symbolic VSAM data set name for the data base.

For HISAM, this operand specifies the symbolic VSAM name for the primary data set.

The DD1 name must be 1-7 alphameric characters long.

**OVFLW=**

For HISAM, this operand specifies the symbolic VSAM name for the overflow data set. The name must be 1-7 alphameric characters long. Skip this operand for SHISAM data bases.

**DEVICE=**

Specifies the device type used for storage of this data base. You must use one of the following:

FBA, 3380, 3375, 3350, 3340, 3330, or 2314.

**BLOCK=**

For SHISAM, this operand specifies the maximum number of logical records per VSAM control interval (CI).

For HISAM, this operand has two parameters. The first parameter specifies the maximum number of logical records per VSAM control interval in the primary data set. The second parameter specifies the

maximum number of logical records per VSAM control interval in the overflow data set.

The BLOCK operand is optional. If omitted, the value(s) are calculated during DBD generation using a control interval size of 2048 bytes wherever possible.

**RECORD=**

For SHISAM, this operand specifies the logical record length in the data set. The length must be an even number large enough to contain the root segment.

For HISAM, this operand has two parameters. The first parameter specifies the logical record length in the primary data set. This length must be an even number large enough to contain at least the root segment plus seven additional bytes for the segment prefix and the DL/I control information. The second parameter specifies the logical record length in the overflow data set. Its length must be at least as large as the length specified in the first parameter.

The RECORD operand is optional. If omitted, the length(s) are calculated during DBD generation.

---

---

# 2.6.2.3 SEGM Statement

The format of the SEGM statement when used to define a SHISAM or HISAM data base is:

Table 58. SEGM Statement Syntax for SHISAM/HISAM Data Base

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= | ° ° ° ° ° ° ° ° | Segment name; 1-8 characters. |
| | ,BYTES= | _ _ _ _ | Segment length.(1) |
| | ,PARENT= | _ _ _ _ _ _ _ _ | Name of this segment's parent.(2) |
| | ,RULES= | (,_ _ _ _ _) | FIRST, LAST, or HERE. |

**Notes:**

1. The BYTES operand is optional. If omitted, DL/I will calculate the segment length during the DBDGEN procedure.
2. The PARENT operand must be specified if you are defining a dependent segment. For a root segment it may be omitted, or PARENT=0 may be specified.

*Coding Example*

```
SEGM                                                    X
      NAME=NAME,                                        X
      BYTES=20,                                         X
      PARENT=SKILL
```

**NAME=**

Specifies the name of the segment being defined. The name can be 1-8 characters long. The first character must be alphabetic; the rest can be alphameric. Duplicate segment names are not allowed within a DBD.

**BYTES=**

Specifies the number of bytes in this segment. Specify only the length of the data portion; do not include the length of the segment prefix area, if any.

For SHISAM, the maximum segment length you can specify is is 4088 bytes. For HISAM, the maximum is 4082 bytes.

The BYTES operand is optional. If omitted, the segment size is

calculated during DBD generation.

**PARENT=**

Specifies the name of this segment's parent segment. For a root segment, either omit this operand or specify PARENT=0. The first SEGM statement in any data base description defines a root segment.

**RULES=**

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 operand 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 (or F)**

States that a new occurrence is to be inserted before the first existing occurrence of this segment type. If the segment has a non-unique sequence field, a new occurrence is inserted before all existing occurrences of the same sequence field value.

**LAST (or L)**

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

**HERE (or H)**

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

---

---

# 2.6.2.4 FIELD Statement

The format of the FIELD statement when used to define a SHISAM or HISAM data base is:

Table 59. FIELD Statement Syntax for SHISAM/HISAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (° ° ° ° ° ° ° ° ,_ _ _ ,_) | Field name; 1-8 characters.<br>SEQ - identifies this as a sequence field.(1)<br>U - only unique values of this sequence field allowed.<br>M - duplicate values of this sequence field can occur in multiple occurrences of the segment. |
| | ,BYTES= | _ _ _ | Field length; 1-256 bytes.(2) |
| | ,START= | _ _ _ _ _ _ _ _ | Field starting position.(3) |
| | ,TYPE= | _ | Field data type.(4) |

**Notes:**

1. The parameters SEQ and U or M are specified only if the field is to be defined as a sequence (key) field in the segment. A unique sequence field must be provided for the root segment.
2. See "Rules and Restrictions" in the BYTES operand description.
3. The START operand is optional. If omitted, DL/I will determine the field's starting position during the DBDGEN procedure.
4. The TYPE operand is optional. If omitted, TYPE=C (character) is assumed.

*Coding Example*

```
FIELD                                                    X
      NAME=(STDCLEVL,SEQ,U),                             X
      BYTES=20,                                          X
      START=1,                                           X
      TYPE=C
```

**NAME=**

Specifies the name of a field that may be referred to by an application program in a DL/I call SSA. Duplicate field names must not be defined for the same segment. The field name must be 1-8 alphameric characters long. It is recommended that you start the field name with an alphabetic character.

The 'SEQ' parameter identifies this field as a sequence (key) field, in the segment. A unique sequence field must be provided for the root segment. A sequence field is optional for dependent segments.

If SEQ is specified, 'U' or 'M' should also be specified. If omitted, the default is U. U indicates that only unique values of this sequence field are allowed. M indicates that duplicate values of this sequence field can occur in multiple occurrences of the segment. M cannot be specified for the root segment.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 256.

***Rules and Restrictions. . .***

- The BYTES operand must be specified for field data types X, P, C, or Z. (See TYPE operand for field data types.)
- The BYTES operand is optional for field data types H and F. If omitted, DL/I assumes a 2 byte length for type H and a 4 byte length for type F.
- Do not specify the BYTES operand for field data types E, D, and L. These data types have implicit lengths of 4, 8, and 16, respectively.
- The sequence field for a root segment of a SHISAM or HISAM data base may not exceed 236 bytes. (This is a sort field size restriction in the DOS/VSE sort/merge program used by the data base change accumulation utility.

**START=**

Specifies the starting position of this field. It is specified in terms of bytes relative to the beginning of the segment within which this field is defined. For the first byte of a segment it is one. Each field must not extend beyond the defined segment length.

Overlapping fields are permitted. An overlapping field can be defined in either of two ways. You can specify the starting position in bytes as already described (START=byte), or, if the field starts at the same position as a previously defined field, you can simply specify the name of the previous field (START=name).

The START parameter is optional. If omitted, DL/I places the 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 contained in the field. Valid data types are:

C - Character
X - Hexadecimal
P - Packed decimal
Z - Zoned decimal
H - Halfword binary
F - Fullword binary
E - Floating point (short)
D - Floating point (long)
L - Floating point (extended)

The TYPE operand is optional. If omitted, TYPE=C is the default.

```
Note:  It is recommended that you explicitly specify the field data
       type.  Failure to do so could result in problems if an attempt
       is made to convert the field data type to another type during
       PSB generation.
```

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 2.6.2.5 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. The formats of these statements are shown below:

Table 60. DBDGEN, FINISH, END Statements for SHISAM/HISAM Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

Coding Example

```
DBDGEN
FINISH
END
```

# 2.6.3 Control Statements Summary

Table 61 summarizes the DL/I control statements used to define SHISAM and HISAM data bases. Only those control statements and operands pertinent to SHISAM and HISAM are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 61. Summary of Control Statements Used to Code SHISAM and HISAM DBDs

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS={SHISAM}<br>{HISAM }<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DEVICE=device-type<br>,DD1=filename<br>,OVFLW=filename(1)<br>[,BLOCK=(number-of-records-per-DDI-CI[,*<br>number-of-records-per-OVFLW-CI])]<br>[,RECORD=(DD1-record-length[,OVFLW-record-leng h])] |
| | SEGM | NAME=segment-name<br>[,PARENT={parent-name}]<br>{*0* }<br>[,BYTES=length]<br>[,RULES=({,FIRST})]<br>{,*LAST* }<br>{,HERE } |
| | FIELD | NAME=(field-name[,SEQ[{,*U*}]])(2)<br>{,M}<br>[,BYTES=length]<br>[,START=starting-position]<br>[,TYPE=data-type] |
| | DBDGEN | |
| | FINISH | |
| | END | |

(1)Symbolic VSAM name for overflow data set. Used for HISAM data bases only.
(2)Unique SEQ must be specified for root segment.

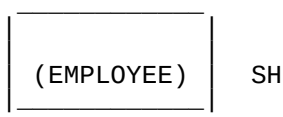
Subtopics:

- 2.6.3.1 Examples of SHISAM and HISAM DBD Generation

# 2.6.3.1 Examples of SHISAM and HISAM DBD Generation

Figure 24 and Figure 25 illustrate examples of the DBD generation process describing SHISAM and HISAM data bases.

Employee Name Data Base
(DBD Name - NAMEFIL)

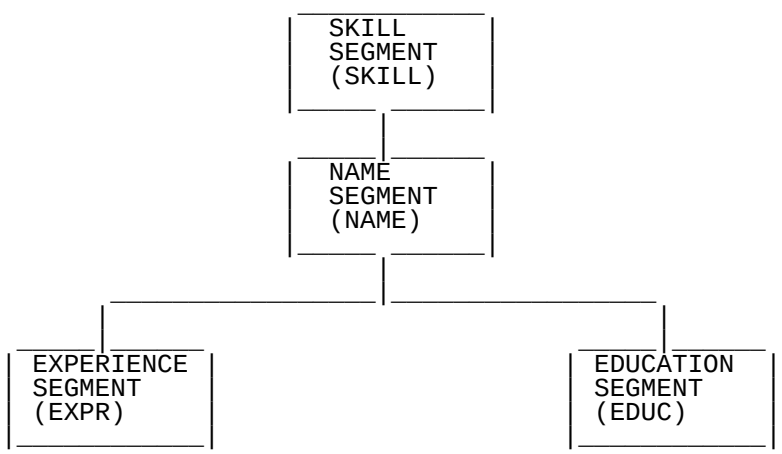



SHISAM contains only root segments.

```
            DBD                                                   X
                  NAME=NAMEFIL,                                   X
                  ACCESS=SHISAM
            DATASET                                               X
                  DD1=EMPNAME,                                    X
                  DEVICE=3380,                                    X
                  BLOCK=2,                                        X
                  RECORD=1024
            SEGM                                                  X
                  NAME=EMPLOYEE,                                  X
                  BYTES=550
            FIELD                                                 X
                  NAME=(MANNBR,SEQ,U),                            X
                  BYTES=15,                                       X
                  START=61,                                       X
                  TYPE=C
            DBDGEN
            FINISH
            END
```

Figure 24. Example of SHISAM DBD

Skills Inventory Data base
(DBD Name - SKILINV)



```
DBD                                                    X
      NAME=SKILINV,                                    X
      ACCESS=HISAM
DATASET                                                X
      DD1=SKHISAM,                                     X
      OVFLW=HISAMOV,                                   X
      DEVICE=3380
SEGM                                                   X
      NAME=SKILL,                                      X
      BYTES=31,                                        X
      PARENT=0
FIELD                                                  X
      NAME=(TYPE,SEQ,U),                               X
      BYTES=21,                                        X
      START=1,                                         X
      TYPE=C
FIELD                                                  X
      NAME=STDCODE,                                    X
      BYTES=10,                                        X
      START=22,                                        X
      TYPE=C
SEGM                                                   X
      NAME=NAME,                                       X
      BYTES=20,                                        X
      PARENT=SKILL
FIELD                                                  X
      NAME=(STDCLEVL,SEQ,U),                           X
      BYTES=20,                                        X
      START=1,                                         X
      TYPE=C
SEGM                                                   X
      NAME=EXPR,                                       X
      BYTES=20,                                        X
      PARENT=NAME
FIELD                                                  X
      NAME=PREVJOB,                                    X
      BYTES=10,                                        X
      START=1,                                         X
      TYPE=C


FIELD                                                  X
      NAME=CLASSIF,                                    X
      BYTES=10,                                        X
      START=11,                                        X
      TYPE=C
SEGM                                                   X
      NAME=EDUC,                                       X
      BYTES=75,                                        X
      PARENT=NAME
FIELD                                                  X
      NAME=GRADLEVL,                                   X
      BYTES=10,                                        X
      START=1,                                         X
      TYPE=C
FIELD                                                  X
      NAME=SCHOOL,                                     X
      BYTES=65,                                        X
      START=11,                                        X
      TYPE=C
DBDGEN
FINISH
END
```

Figure 25. Example of HISAM DBD

---



---

# 2.7 Chapter 7. Defining SHSAM and HSAM Data Bases

This chapter describes how to define a data base description (DBD) for SHSAM and HSAM data bases.

***For a SHSAM data base description,*** you must prepare an input stream of several DL/I control statements specifying:

- The symbolic filename of a SAM input file. The application program retrieves segments from this data base.
- The symbolic filename of a SAM output file. The application program loads segments to this data base.
- One segment type for the data base (root segments only).
- The fields for the segment type.

***For an HSAM data base description,*** you must prepare an input stream of several DL/I control statements specifying:

- The symbolic filename of a SAM input file. The application program retrieves segments from this data base.
- The symbolic filename of a SAM output file. The application program loads segments to this data base.
- The segments types for the data base.
- The fields within each segment type.

Subtopics:



# 2.7.1 Input Structure and Rules

Figure 26 and Figure 27 illustrate the rules for structuring the batch input streams defining SHSAM and HSAM data bases, respectively. These figures show the types of control statements required, the number of each type that may be specified, and their specific order of sequence. A separate input stream is required for each data base you define.

***To Define a SHSAM Data Base . . .***

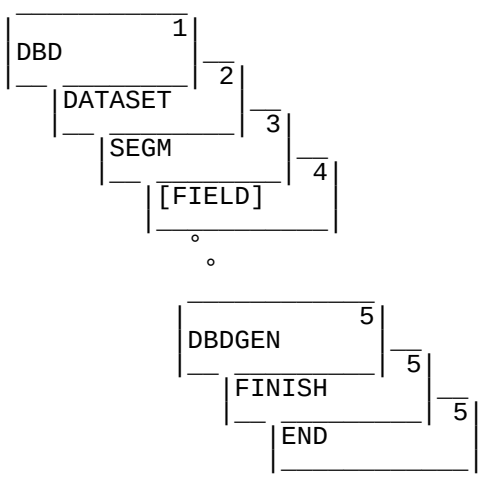



Figure 26. Batch Input Stream for SHSAM Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.

2. Code one DATASET statement. It must follow the DBD statement.

3. Code one SEGM statement. It must follow the DATASET statement.

4. FIELD statements are optional for SHSAM data bases. They may be omitted if it is not necessary to define fields within the segment. If specified, code one FIELD statement for each field you want defined. A maximum of 255 fields may be defined in a single SHSAM data base definition. The FIELD statements must follow the SEGM statement.

5. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown, at the end of your input stream.

***To Define a HSAM Data Base . . .***

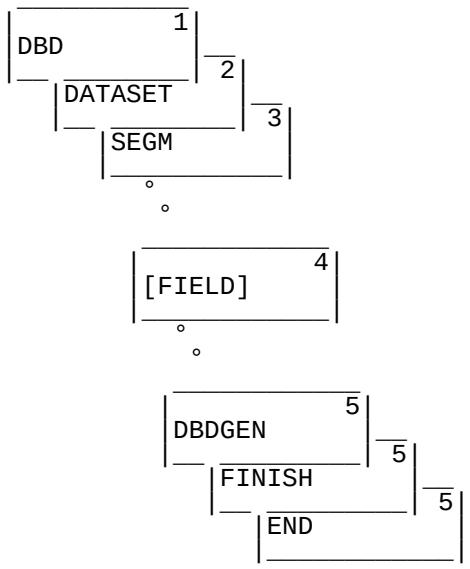



Figure 27. Batch Input Stream for HSAM Data Bases

1. Code one DBD statement. It must be the first statement in your input stream.

2. Code one DATASET statement. It must follow the DBD statement.

3. Code one SEGM statement for each segment type in your data base. Place them in the input stream in the same sequence as the segments appear in the hierarchic order of the data base you are defining. The maximum number of SEGM statements allowed per data base is 255.

4. FIELD statements are optional for HSAM data bases. They may be omitted if it is not necessary to define fields within segments. If specified, code one FIELD statement for each field you want defined. Place them after the SEGM statement that defines the segment containing those fields. A maximum of 255 fields per segment type and 1020 fields total may be defined in a single HSAM data base definition.

5. Code one DBDGEN statement, one FINISH statement, and one Assembler END statement. Place them in the sequence shown at the end of your input stream.

---

# 2.7.2 Control Statements Format

This section explains how to code DL/I control statements to define SHSAM and HSAM data bases. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introduction.

Subtopics:



## 2.7.2.1 DBD Statement

The format of the DBD statement when used to define a SHSAM or HSAM data base is:

Table 62. DBD Statement Syntax for SHSAM/HSAM Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBD | NAME= proc. | ° ° ° ° ° ° ° | Name of INDEX data base. |
| | ,ACCESS= proc. | ° ° ° ° ° | SHSAM or HSAM. |
| | ,IMSCOMP= | _ _ _ | YES = Make data base compatible to IMS/VS.<br>NO = Format compatibility not required. |

*Coding Example*

```
       DBD                                                    X
             NAME=SKILINV,                                    X
             ACCESS=HSAM,                                     X
             IMSCOMP=NO
```

**NAME=**
Specifies the name of the data base. The name must be 1-7 characters long. The first character must be alphabetic; the rest can be alphameric. Do not use the character '@'.

**ACCESS=**
Specifies the access method used for the data base. Here you must specify either SHSAM or HSAM.

**IMSCOMP=**
Indicates whether this DL/I data base is to be created in a format compatible with the Information Management System (IMS). YES is recommended if either eventual migration from DL/I to IMS is planned, or, if this data base is to be alternately accessed by DL/I and IMS.

NO means format compatibility is not required. NO is the default if the IMSCOMP operand is omitted.

# 2.7.2.2 DATASET Statement

The format of the DATASET statement when used to define a SHSAM or HSAM data base is:

Table 63. DATASET Statement Syntax for SHSAM/HSAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DATASET | DEVICE= proc. | ° ° ° ° | Physical device type for storage of this data base. |
| | ,DD1= proc. | ° ° ° ° ° ° ° | Name of input file; 1-7 alphameric characters. |
| | ,DD2= proc. | ° ° ° ° ° ° ° | Name of output file; 1-7 alphameric characters. |
| | ,DEVADDR= | (**S Y S** _ _ _<br>**,S Y S** _ _ _) | Symbolic tape unit address for input file.(1)<br>Symbolic tape unit address for output file.(1) |
| | ,RECORD= | (_ _ _ _ _<br>,_ _ _ _ _) | Length of input record.(2)<br>Length of output record.(2) |

**Notes:**

1. The DEVADDR operand must be specified if DEVICE=TAPE. Otherwise, omit it.
2. The RECORD operand is optional. If omitted, DL/I will determine the record length values during DBD generation.

*Coding Example*

```
       DATASET                                                  X
             DEVICE=TAPE,                                       X
             DD1=SKILINV,                                       X
             DD2=HSAMOUT,                                       X
             DEVADDR=(SYS010,SYS011),                           X
             RECORD=(1000,1000)
```

**DEVICE=**
Specifies the device type used for storage of this data base. You must use one of the following:
FBA, 3380, 3375, 3350, 3340, 3330, 2314, or TAPE.

**DD1=**

Specifies the symbolic filename of the sequential input file; 1-7 alphameric characters long.

**DD2=**

Specifies the symbolic filename of the sequential output file; 1-7 alphameric characters long.

**DEVADDR=**

Specifies the symbolic tape units to be associated with the symbolic filenames specified in the DD1 and DD2 operands. Use the form SYSnnn for the assignments, where nnn is replaced with a number from 000 through 254. Do not use the same number for both assignments.

**RECORD=**

Specifies the input logical record length and the output logical record length. The lengths must be numeric values divisible by two. This operand is optional; if omitted, a record length less than or equal to 32766 bytes for tape units, 16K for FBA devices, or the track size for DASD is substituted during DBD generation.

---

# 2.7.2.3 SEGM Statement

The format of the SEGM statement when used to define a SHSAM or HSAM data base is:

Table 64. SEGM Statement Syntax for SHSAM/HSAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| SEGM | NAME= proc. | ° ° ° ° ° ° ° ° | Segment name; 1-8 characters. |
| | ,BYTES= | _ _ _ _ _ | Segment length.(1) |
| | ,PARENT= | _ _ _ _ _ _ _ _ | Name of this segment's parent.(2) |

**Notes:**

1. The BYTES operand is optional. If omitted, DL/I will calculate the segment length during DBD generation.
2. The PARENT operand must be specified if you are defining a dependent segment. For a root segment it may be omitted, or PARENT=0 may be specified.

*Coding Example*

```
SEGM                                                    X
      NAME=NAME,                                        X
      BYTES=20,                                         X
      PARENT=SKILL
```

**NAME=**

Specifies the name of the segment being defined. The name can be 1-8 characters long. The first character must be alphabetic; the reset can be alphameric. Duplicate segment names are not allowed within a DBD generation.

**BYTES=**

Specifies the number of bytes in this segment. Specify only the length of the data portion; do not include the length of the segment prefix area, if any.

For SHSAM, the maximum segment length you can specify is either the track size of the physical device or, 32766 bytes. For HSAM, the maximum is either the track size minus 2 or, 32764 bytes.

The BYTES operand is optional. If omitted, the segment size is calculated during DBD generation.

**PARENT=**
Specifies the name of this segment's parent segment. For a root segment, the PARENT operand may be omitted, or PARENT=0 specified. The first SEGM statement in any data base description defines a root segment.

---

---

# 2.7.2.4 FIELD Statement

The format of the FIELD statement when used to define a SHSAM or HSAM data base is:

Table 65. FIELD Statement Syntax for SHSAM/HSAM Data Bases

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| FIELD | NAME= | (_ _ _ _ _ _ _ _<br>,_ _ _<br>,_) | Field name; 1-8 characters.<br>SEQ - identifies this as a sequence field.(1)<br>U - only unique values of this sequence field allowed.<br>M - duplicate values of this sequence field can occur in multiple occurrences of the segment. |
| | ,BYTES= | _ _ _ | Field length; 1-256 bytes.(2) |
| | ,START= | _ _ _ _ _ _ _ _ | Field starting position.(3) |
| | ,TYPE= | _ | Field data type.(4) |

**Notes:**

1. Field statements are optional for SHSAM and HSAM data bases.
2. See "Rules and Restrictions" in the BYTES operand description.
3. The START operand is optional. If omitted, DL/I will determine the field's starting position during DBD generation.
4. The TYPE operand is optional. If omitted, TYPE=C (character) is assumed.

*Coding Example*

```
       FIELD                                                  X
             NAME=STDCLEVL,                                   X
             BYTES=20,                                        X
             START=1,                                         X
             TYPE=C
```

**NAME=**

Specifies the name of the field being defined within a segment type. The name can be 1-8 alphameric characters long. It is recommended that you start the field name with an alphabetic character. Duplicate field names must not be defined within the same segment type.

The 'SEQ' parameter identifies this field as the sequence field. If SEQ is specified, 'U' or 'M' should also be specified. If omitted, the default is 'U'. 'M' indicates that duplicate values of this sequence field can occur in multiple occurrences of the segment.

**BYTES=**

Specifies the length of this field in terms of bytes. If specified, it must be a number from 1 through 256.

***Rules and Restrictions. . .***

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

**START=**

Specifies the starting position of the field in terms of bytes relative to the beginning of the segment. Start position for the first byte of a segment is one.

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=field-name). The name of the previously defined field must start with an alphabetic character.

If it is not specified, DL/I places the 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.

**TYPE=**

Specifies the type of data contained in the field. Valid data types are:

C - Character
X - Hexadecimal
P - Packed decimal
Z - Zoned decimal
H - Halfword binary
F - Fullword binary
E - Floating point (short)
D - Floating point (long)
L - Floating point (extended)

The TYPE operand is optional. If omitted, TYPE=C is assumed. It is recommended, however, that you explicitly specify the field data type. Failure to do so could result in problems if an attempt is made to convert the field data type to another type during PSB generation. (See TYPE operand in SENFLD Statement.)

---

# 2.7.2.5 DBDGEN, FINISH, and END Statements

The DBDGEN, FINISH, and END statements are required. The format of these statements are shown below:

Table 66. DBDGEN, FINISH, END Statements for SHSAM/HSAM Data Bases

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DBDGEN | | | This statement is required. It indicates the end of control statements defining the data base. |
| FINISH | | | This statement must be included for source-level compatibility with IMS/VS. |
| END | | | This statement is required. It indicates the end of input statements to the VSE Assembler. |

*Coding Example*

```
DBDGEN
FINISH
END
```

# 2.7.3 Control Statements Summary

Table 67 summarizes the DL/I control statements used to define SHSAM and HSAM data bases. Only those control statements and operands pertinent to SHSAM and HSAM are illustrated. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 67. Summary of Control Statements Used to Code SHSAM and HSAM DBDs

| [Label] | Statement | Operand |
|---|---|---|
| | DBD | NAME=data-base-name<br>,ACCESS={SHSAM}<br>{HSAM }<br>[,IMSCOMP={YES}]<br>{*NO* } |
| | DATASET | DEVICE=device-type<br>,DD1=input-filename<br>,DD2=output-filename<br>[,DEVADDR=(SYSnnn,SYSnnn)](1)<br>[,RECORD=(input-length[,output-length])] |
| | SEGM | NAME=segment-name<br>[,PARENT={parent-name}]<br>{*0* }<br>[,BYTES=length] |
| | [FIELD | NAME=(field-name[,SEQ[{,*U*}]])]<br>{,M}<br>[,BYTES=length]<br>[,START=starting-position]<br>[,TYPE=data-type] |
| | DBDGEN | |
| | FINISH | |
| | END | |

(1)Must be specified if DEVICE=TAPE

Subtopics:

- 2.7.3.1 Examples of SHSAM and HSAM DBD Generation

# 2.7.3.1 Examples of SHSAM and HSAM DBD Generation

Figure 28 and Figure 29 illustrate examples of the DBD generation process describing SHSAM and HSAM data bases.

```
Employee Name Data Base
(DBD Name - NAMEFIL)
 _____________
|             |  SHSAM contains only root segments.
|             |  Any SAM file defined with
| (EMPLOYEE)  |  RECFORM=FIXUNB may be defined as a
|_____________|  SHSAM data base.
```

```
            DBD                                                   X
                  NAME=NAMEFIL,                                   X
                  ACCESS=SHSAM
            DATASET                                               X
                  DEVICE=TAPE,                                    X
                  DD1=INFILE,                                     X
                  DD2=OUTFILE,                                    X
                  DEVADDR=(SYS010,SYS011)
            SEGM                                                  X
                  NAME=EMPLOYEE,                                  X
                  BYTES=550
            DBDGEN
            FINISH
            END
```

Figure 28. Example of SHSAM DBD

```
Skills Inventory Base
(DBD Name - SKILINV)
                    _____________
                   |  SKILL      |
                   |  SEGMENT    |
                   |  (SKILL)    |
                   |_____ _______|
                        |
                    ____|_______
                   |  NAME       |
                   |  SEGMENT    |
                   |  (NAME)     |
                   |_____ _______|
                        |
          ______________|_______________
         |                              |
    _____|_______                 _____|_______
   | EXPERIENCE  |               | EDUCATION   |
   | SEGMENT     |               | SEGMENT     |
   | (EXPR)      |               | (EDUC)      |
   |_____________|               |_____________|
```

```
            DBD                                                   X
                  NAME=SKILINV,                                   X
                  ACCESS=HSAM,                                    X
                  IMSCOMP=NO
            DATASET                                               X
                  DD1=SKILINV,                                    X
```

```
      DD2=HSAMOUT,                                    X
      DEVICE=TAPE,                                    X
      DEVADDR=(SYS010,SYS011),                        X
      RECORD=(1000,1000)
SEGM                                                  X
      NAME=SKILL,                                     X
      BYTES=31,                                       X
      PARENT=0
FIELD                                                 X
      NAME=TYPE,                                      X
      BYTES=21,                                       X
      START=1,                                        X
      TYPE=C
FIELD                                                 X
      NAME=STDCODE,                                   X
      BYTES=10,                                       X
      START=22,                                       X
      TYPE=C
SEGM                                                  X
      NAME=NAME,                                      X
      BYTES=20,                                       X
      PARENT=SKILL
FIELD                                                 X
      NAME=STDCLEVL,                                  X
      BYTES=20,                                       X
      START=1,                                        X
      TYPE=C
SEGM                                                  X
      NAME=EXPR,                                      X
      BYTES=20,                                       X
      PARENT=NAME
FIELD                                                 X
      NAME=PREVJOB,                                   X
      BYTES=10,                                       X
      START=1,                                        X
      TYPE=C
FIELD                                                 X
      NAME=CLASSIF,                                   X
      BYTES=10,                                       X
      START=11,                                       X
      TYPE=C
SEGM                                                  X
      NAME=EDUC,                                      X
      BYTES=75,                                       X
      PARENT=NAME
FIELD                                                 X
      NAME=GRADLEVL,                                  X
      BYTES=10,                                       X
      START=1,                                        X
      TYPE=C
FIELD                                                 X
      NAME=SCHOOL,                                    X
      BYTES=65,                                       X
      START=11,                                       X
      TYPE=C
DBDGEN
FINISH
END
```

Figure 29. Example of HSAM DBD

---

# 2.8 Chapter 8. Doing a DBD Generation

This chapter assumes you have completed the *definition* step of DBDGEN and are now ready to do the *generation* step. The generation step may be invoked anytime after DL/I has been installed in your DOS/VSE system.

DBDGEN is run as a standard system job. Note that it has two steps. Step 1 is a macro assembly execution. This step reads and expands your input statements and produces a DBD object module and an assembly listing. The DL/I macros used for step 1 reside in the 'DL/I prod lib'. In step 2, the generated DBD is cataloged and link-edited as a load module (type PHASE) in the 'DL/I user lib' for subsequent processing of the data base it defines. A link-edit listing is also produced.

**Note:** In the case of an HSAM DBD, two unresolved references will be detected during the link-edit. This is not an error. The needed I/O modules will be included during execution of the ACB creation and maintenance utility.

Remember that each data base used with DL/I requires a separate data base description (DBD) generation.

Subtopics:



# 2.8.1 Preparing the Job Control Statements

The job control statements used to invoke the DBDGEN procedure are shown below. The placement of your DBD generation input statements within the job control statements is also shown.

```
// JOB DBDGEN
// OPTION CATAL,NODECK
// LIBDEF *,SEARCH='DL/I prod lib'
// LIBDEF PHASE,CATALOG='DL/I user lib'
// EXEC ASSEMBLY
        DBD      |
        DATASET  |
        ACCESS   |
        SEGM     |
        FIELD    | Put your DBD generation input statements here
        LCHILD   |
        DBDGEN   |
        FINISH   |
        END
/*
// EXEC LNKEDT
/&
```

Only the minimum requirements for the job control statements are shown. They may contain additional information. Refer to *VSE/Advanced Functions System Control Statements* for detailed information about these statements.

 *© Copyright IBM Corp. 1981, 1989*

# 2.8.2 Analyzing the Output

A standard assembly listing is created each time the DBD generation procedure is executed.

***An Important Note About this Listing . . .***

The DBDGEN assembly listing contains recommended values for some parameters you will be asked to specify when you define the data sets of the data base to VSAM. See Chapter 12, "Defining VSAM Data Sets," for details.

Any errors found, such as invalid parameters or the omission or invalid sequence of input statements, will cause the DBD generation procedure to terminate prematurely and diagnostic messages to be printed.

You must then correct the errors and rerun the job. Every DBDGEN must be error free.

Diagnostic messages used to identify DBDGEN error conditions are described in *DL/I DOS/VS Messages and Codes*. As shown in the following list, each control statement uses a different prefix to identify its diagnostic messages. In all cases, the variable *nnn* is replaced with a unique identification number when a particular message is issued:

DBD statements use DBD*nnn*
DATASET statements use DMAN*nnn*
ACCESS statements use ACC*nnn*
SEGM statements use SEGM*nnn*
FIELD statements use FLD*nnn*
LCHILD statements use LCHD*nnn*
XDFLD statements use XDFLD*nnn*
DBDGEN statements use DGEN*nnn*
FINISH statements use FIN*nnn*

# 3.0 Part 3. Describing an Application Program View of a Data Base

This section explains how to make DL/I data bases accessible to your application programs. The procedure you will use to do this is called *Program Specification Block (PSB) Generation.* For brevity, it is simply called *PSBGEN.*

PSBGEN is a two-step procedure: The *definition* step and the *generation* step.

The definition step is described in Chapter 9, "Defining a Program Specification Block." For this step you must code a series of control statements to define a PSB. Each application program using DL/I data bases requires a PSB. Special PSBs are also required when "loading" DL/I data bases.

After you complete your PSB definition, Chapter 10, "Doing a PSB Generation" explains how to do the generation step.

**Creating a PSB Interactively**

The PSB definition and generation procedures described here can also be done interactively on a 3270-type terminal by using the Interactive Macro Facility (IMF). See *DL/I DOS/VS Interactive Definition Facility and Utilities* for more information.

Subtopics:

# 3.1 Chapter 9. Defining a Program Specification Block

Each application program using DL/I data bases needs a program specification block (PSB).

This chapter explains how to code the control statements used to define PSBs.

The first section describes the requirements for basic PSBs. A basic PSB refers to one that uses segment sensitivity and is typical of the PSB you would define for an application program to process DL/I data bases. The remaining sections explain other PSB requirements such as:

- Coding a PSB for loading a data base.
- Coding a PSB for a logical data base.
- Coding a PSB for a secondary index.
- Coding a PSB with field level sensitivity.

Subtopics:



# 3.1.1 Coding Basic PSBs

Subtopics:



# 3.1.1.1 Input Structure and Rules

Figure 30 illustrates the rules for structuring a batch input stream defining a basic PSB. The figure shows the types of control statements required, the number of each type that may be specified, and their specific order of sequence.

***To Define a Basic PSB . . .***

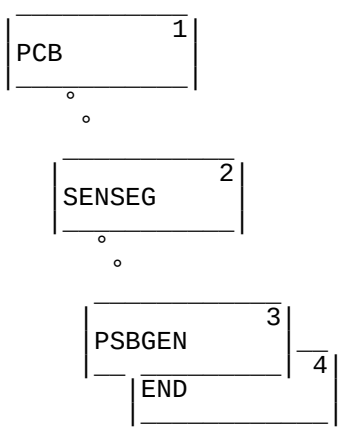



Figure 30. Batch Input Stream for Basic PSBs

1. Code one PCB statement for each physical or logical hierarchical data structure the application program intends to use. A maximum of 255 PCBs may be coded in a single PSB definition. Index data bases, when being used as indexes, do not require a PCB statement; only when they are processed as a normal data base in their own right is one required.

2. Code one or more SENSEG statements for each PCB statement to define the hierarchically related set of data segments to which the application program is sensitive. Put the SENSEG statements after the PCB statements to which they are related. Keep the SENSEG statements in the same hierarchical sequence as in the DBD. However, only those segments in the hierarchical path to any required segment should be coded.

3. Code one PSBGEN statement. Put it after the last set of PCB/SENSEG statements in your input stream.

4. Code one Assembler language END statement. Put it at the end of your input stream.

# 3.1.2 Control Statement Formats

This section explains how to code DL/I control statements to define a basic PSB. The coding conventions for the PSB are exactly the same as for the DBD.

Subtopics:



# 3.1.2.1 PCB Statement

The format of the PCB statement is:

Table 68. PCB Statement Syntax for PSB Generation

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| PCB | TYPE= | **D B** | DB must be specified. |
| | ,DBDNAME= proc. | ° ° ° ° ° ° ° | Name of data base (DBD) to be accessed via this PCB. |
| | ,PROCOPT= proc. | ° ° ° ° | Processing options: G I R D A GO N T P E L LS (1) |
| | ,KEYLEN= proc. | ° ° ° ° | Longest concatenated key. |
| | ,POS= | _ _ _ _ _ _ _ _ | SINGLE (or S). MULTIPLE (or M) |

**Notes:**

1. L and LS can be used only when coding a unique PSB for the original loading of a data base.

*Coding Example*

```
PCB                                                    X
      TYPE=DB,                                         X
      DBDNAME=STDCDBP,                                 X
      PROCOPT=AP,                                      X
      KEYLEN=50
```

**TYPE=DB**
is required for all DL/I data base PCBs.

**DBDNAME=**
specifies the name of the data base (DBD) which is accessed via this PCB.

**PROCOPT=**
specifies the processing options on sensitive segments declared in this PCB that may be used in an associated application program. This operand allows a maximum specification of four characters. The

letters you may specify 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 PROCOPT=A.

A - All, include the above four functions.

N - Returns status code 'GG' instead of abnormal termination if invalid pointers are detected. N can be used only with GO and GOP (as GON or GONP).

T - Same as 'N', except that an automatic retry of the call is performed (which may cause a wait until concurrent updates in the same data base record are completed) if invalid pointers are detected. (Status code 'GG' will be returned if the retry fails). T can be used only with GO and GOP (as GOT or GOTP).

**Note:** No automatic retry will be performed for S(HISAM) data bases. If new PROCOPTs are specified within a PSB that is used in batch environment, this will be ignored.

P - Required if a path call is to be used for a retrieval, replacement, or update. In an HLPI command, this means using a FROM or INTO option for a parent segment. In a CALL statement, a D command code is used in an SSA. P cannot be used with L or LS.

E - Exclusive use of the data base or segment. E can be used only with G, I, R, D, A, and P.

O - Inhibits locking (enqueueing) by program isolation during retrievals of the same segment types in a data base. O can be used only with G and GP.

L, LS - Can be used only when coding a unique PSB for the original loading of a data base. For more information about processing options L and LS, see "Coding PSBs for Loading Data Bases" later in this chapter.

**More about PROCOPTs N and T**

An abnormal program termination (with message DLZ798I or DLZ801I) can occur with PROCOPT=GO(P) if another program is updating pointers while this program is following the pointers. To reduce the number of abnormal terminations of this type, a PROCOPT=GO(P) operand in a PCB statement can be coded with an additional 'N' or 'T'. This specification is valid for all access methods supported by DL/I, but has no effect on (S)HSAM data base organizations.

If the application program receives a 'GG' status code, it must decide whether to terminate processing or to continue.

**Note:** When issuing a 'GG' status code, DL/I sets the position to the beginning of the data base. The key feedback area contains the key of the last successfully accessed root segment.

For a description of the 'GG' status code, see *DL/I DOS/VS Messages and Codes*.

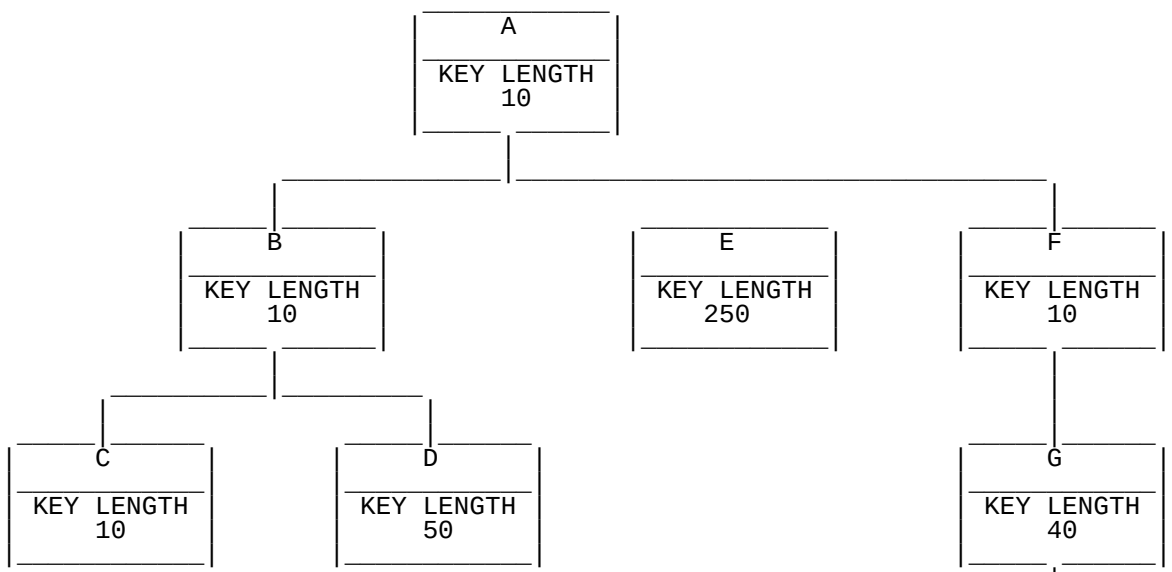
Status code 'GG' (if any) will be returned from both the DL/I HLPI interface and DL/I call interface. However, if the HLPI interface is used, the following must be considered: currently, status codes GA, GB, GE, GK, II, LB, NE, TG, and blank will be returned in DIBSTAT. They do not necessarily indicate errors since the result of the command may be valid even though the data operand may not be what the caller expected. All other status codes indicate conditions that would cause the program to abend. The new status code 'GG' will also be returned in DIBSTAT. But, because the result of the command is not what the caller expected, the application program should handle this situation as described above.

**For More Information...**

about processing options, see *DL/I DOS/VS Data Base Administration*.

**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. Figure 31 explains the definition of KEYLEN.

**Note:** When using secondary indexes with multiple search fields where the search fields are longer than the target KEYLEN, the search field length should replace the key length when calculating the total KEYLEN of a path.




| DATA BASE HIERARCHICAL PATHS | CONCATENATED KEY LENGTH PER PATH |
|---|---|
| 1 = A + B + C | = 30 BYTES |
| 2 = A + B + D | = 70 BYTES |
| 3 = A + E | = 260 BYTES * |
| 4 = A + F + G + H + J | = 120 BYTES |

* In this example, the third hierarchical path (A + E) with a total of 260 bytes is the longest concatenated key length. Therefore, KEYLEN=260 or greater must be specified in the PCB statement.

Figure 31. Concatenated Key Length for Data Base Hierarchical Paths

**POS=**

indicates whether single or multiple positioning is to be used. Specify SINGLE or S if you want DL/I to maintain only one position in the data base for this PCB. This single position will then be used to satisfy all subsequent GN calls. Specify MULTIPLE or M if you want a unique position to be maintained in each hierarchical path in the data base.

**For More Information...**

about positioning, see the appropriate DL/I manual for application programmers. If you are using HLPI, see *DL/I DOS/VS Application Programming: High Level Programming Interface*. If you are using the CALL or RPG interface, see *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

# 3.1.2.2 SENSEG Statement

The format of the SENSEG statement is:

Table 69. SENSEG Statement Syntax for PCB Generation

| Statement | Keyword | Parameter | Description (See also Notes below). |
|---|---|---|---|
| SENSEG | NAME= proc. | ° ° ° ° ° ° ° ° | Name of segment type. |
| | ,PARENT= | _ _ _ _ _ _ _ _ | Name of this segment's parent.(1) |
| | ,PROCOPT= | _ _ _ _ | Processing options: G I R D A P E (2) |

**Notes:**

1. The PARENT operand must be specified if you are defining a dependent segment. For a root segment it may be omitted, or PARENT=0 may be specified.
2. The PROCOPT operand is optional. If omitted, the PCB PROCOPT operand is used as the default.

**Coding Example**

```
SENSEG                                          X
      NAME=STSCCST,                             X
      PARENT=0
SENSEG                                          X
      NAME=STSCLOC,                             X
      PARENT=STSCCST
```

**NAME=**
is the name of the segment type as defined through a SEGM statement during DBD generation.

**PARENT=**
specifies the name of this segment's parent segment. For a root segment, the PARENT operand may be omitted, or PARENT=0 specified. The first SENSEG statement after a PCB statement always identifies the root segment.

**PROCOPT=**
Identifies 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. The SENSEG PROCOPT overrides the PCB PROCOPT.

In a CALL statement, the processing option 'P' is required, if command code 'D' (path call) is to be used in SSAs for retrieval or insert calls. In an HLPI path call, 'P' must be specified during PSB generation for retrieval, replace, or insert commands. 'P' cannot be used with L or LS. For more information about the processing options L and LS, see "Coding PSBs for Loading Data Bases" later in this chapter.

**Note:** Consider always coding PROCOPT=G on the PCB statement and using the SENSEG statement PROCOPT operand to override this specification as required.

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 3.1.2.3 PSBGEN Statement

The format of the PSBGEN statement is:

Table 70. PSBGEN Statement Syntax for PCB Generation

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| PSBGEN | LANG=<br>proc. | ° ° ° ° ° | Specify one: COBOL PL/I ASSEM<br>RPG |
| | ,PSBNAME=<br>proc. | ° ° ° ° ° ° ° | Name of this PSB. |

*Coding Example*

```
PSBGEN                                                  X
      LANG=ASSEM,                                       X
      PSBNAME=STBCPHA
```

**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. PL/I may be specified as PL/I, PL/1, PLI, or PL1.

**Note:** Online application programs and HLPI application programs are not restricted to the specific programming language specified in the LANG operand of PSBGEN statements.

**PSBNAME=**

Specifies the name of this PSB. The name must be 1-7 characters long. The first character must be alphabetic; the rest may be alphameric. Do not use the character '@'.

## 3.1.2.4 END Statement

The format of the END statement is:

Table 71. END Statement Syntax for PCB Generation

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| END | | | This Assembler language statement is required at the end of the PSB input stream. It indicates the end of data. |

```
Coding Example
       END
```

 © *Copyright IBM Corp. 1981, 1989*

# 3.1.3 Control Statements Summary

Table 72 summarizes the DL/I control statements used to define a basic PSB. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 72. Summary of Control Statements Used to Code PSBs

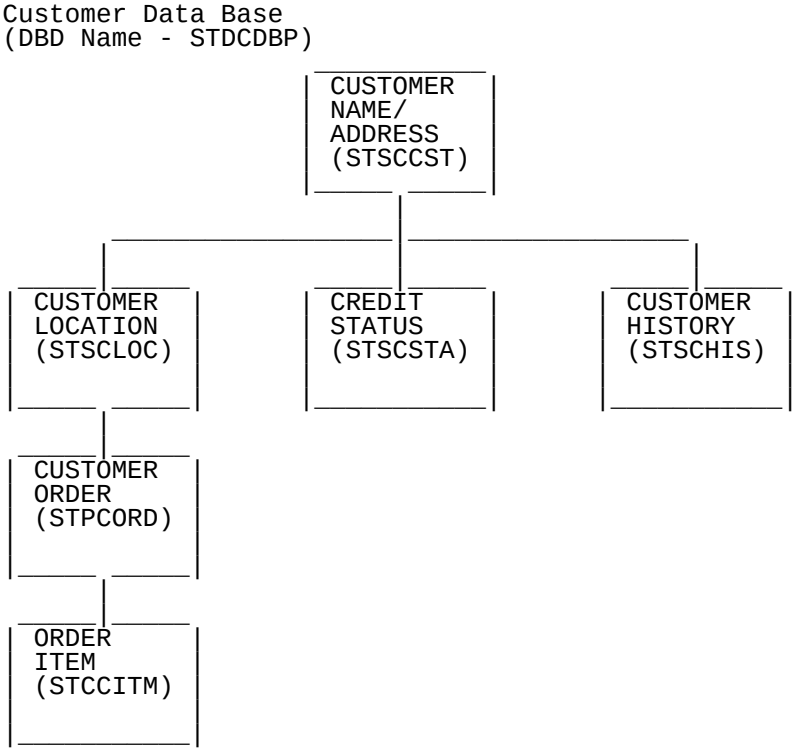
| [Label] | Statement | Operand |
|---|---|---|
| | PCB | TYPE=DB<br>,DBNAME=dbname<br>,PROCOPT={{[G][I][R][D]}{[P][E]}}<br>{{A }{[P][E]}}<br>{L[S] }<br>{GO[P] }<br>{GON[P] }<br>{GOT[P] }<br>,KEYLEN=value<br>[,POS={MULTIPLE}]<br>{*SINGLE* } |
| | SENSEG | NAME=segment-name<br>[,PARENT={*0* }]<br>{parent-name}<br>[,PROCOPT={{[G][I][R][D]}{[P][E]}}]<br>{{A }{[P][E]}} |
| | PSBGEN | LANG={COBOL}<br>{PL/I }<br>{ASSEM}<br>{RPG }<br>,PSBNAME=psbname |
| | END | |

Subtopics:

- 3.1.3.1 Example Basic PSB

# 3.1.3.1 Example Basic PSB

Figure 32 is an example of a PSB for an application program to process the Customer data base.




```
            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               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
```

Figure 32. Example of PSB to Process Customer Data Base

# 3.1.4 Coding PSBs for Loading Data Bases

A unique *load PSB* is required for the original loading of each DL/I data base you intend to use. This is in addition to the basic PSBs coded for application programs to process the data bases.

The coding requirements for a load PSB are similar to that of the basic PSB already described. The differences are described next:

- *For the PCB statement,* you must specify either PROCOPT=L or PROCOPT=LS. No other processing options are allowed for a load PSB. Specify L for the original loading of all data bases except where the access method is HD indexed or HIDAM, or HSAM or SHSAM when a sequence field is defined in the root segment. In these cases, specify PROCOPT=LS. This option loads the segments in ascending sequence.

- *For the SENSEG statement,* omit the PROCOPT operand. This operand is invalid if the PCB PROCOPT is L or LS.

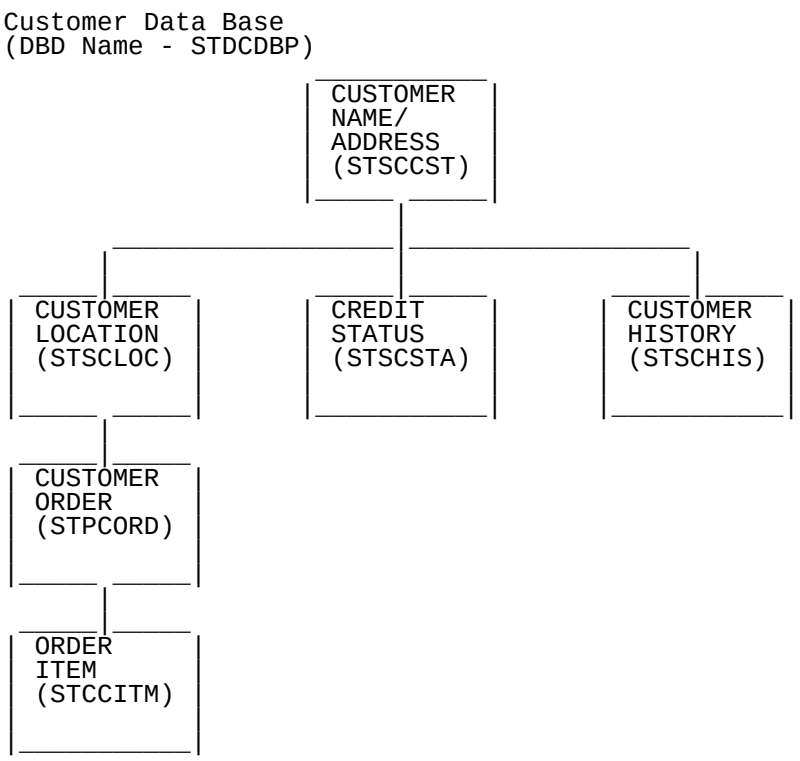
Subtopics:



# 3.1.4.1 Example Load PSB

Figure 33 is an example of the load PSB for the Customer data base.




```
            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
```

Figure 33. Example of PSB to Load Customer Data Base

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 3.1.5 Coding PSBs for Logical Data Bases

To use a logical data base, the application 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. (The DBD name is specified in the DBDNAME operand of the PCB statement and the segment is specified in the NAME operand of the SENSEG statement.)

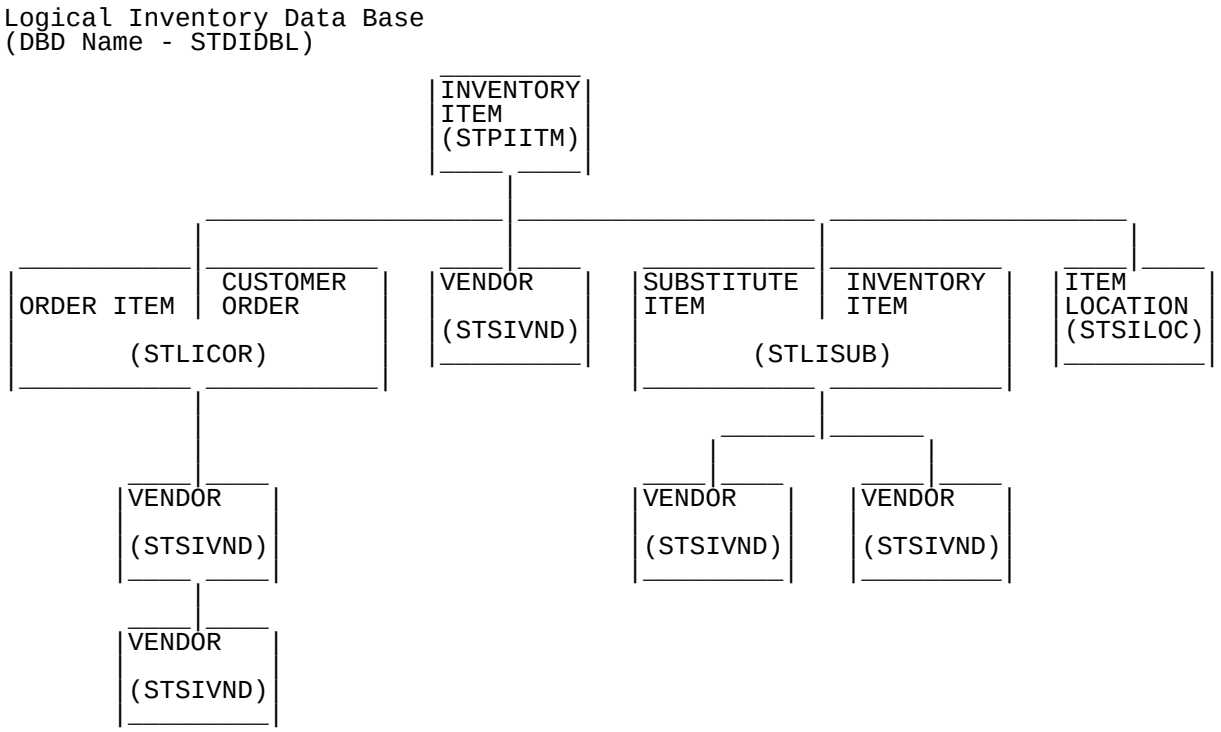
You should code SENSEG statements only for the segments the program actually needs and the segments in the hierarchical path to those segments. All of this is based on the logical DBD, so the hierarchical path may well include physical and logical paths.

Subtopics:



# 3.1.5.1 Example PSBs for Logical Data Bases

Figure 34 and Figure 35 show sample PSBs for logical data bases.

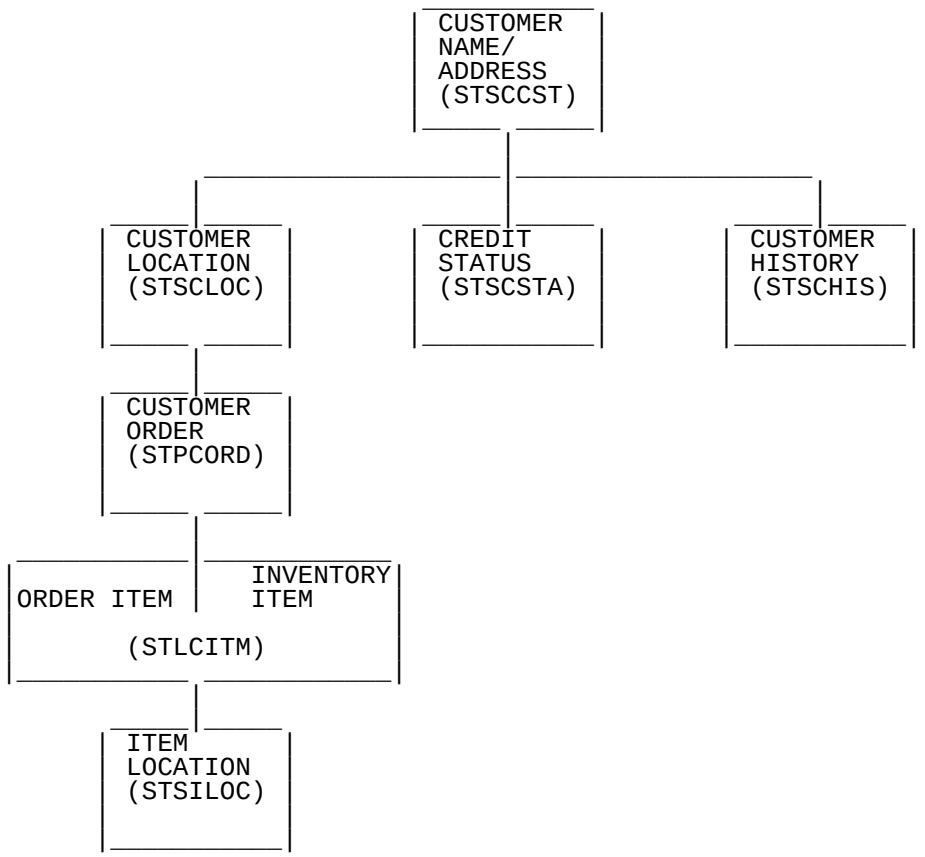



```
PCB                                                              X
      TYPE=DB,                REQUIRED                           X
      DBDNAME=STDIDBL,        LOGICAL DBD NAME                   X
      PROCOPT=AP,             A=ALL FUNCS.,P=PATH CALL POSSIB    X
      KEYLEN=50               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=STSCLOC,           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=STSISVD,                                              X
      PARENT=STLISUB
SENSEG                                                           X
      NAME=STSISLC,                                              X
      PARENT=STLISUB
```

```
            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
```

Figure 34. Example of PSB for Logical Inventory Data Base

Logical Customer Data Base
(DBD Name - STDCDBL)




```
            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 DISCUSSION
            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 35. Example of PSB for Logical Customer Data Base

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 3.1.6 Coding PSBs for Secondary Indexes

To use a secondary index, an application program uses a basic PSB as previously described, but the following additional parameter must be coded in the PCB statement:

**PROCSEQ=index-dbdname**
Specifies the name of the secondary index data base. If this operand is used, subsequent SENSEG statements must reflect the secondary data structure of segment types in the indexed data base. Thus, for example, the first SENSEG statement must name the index target segment as the root segment. This operand is invalid if PROCOPT is L or LS.

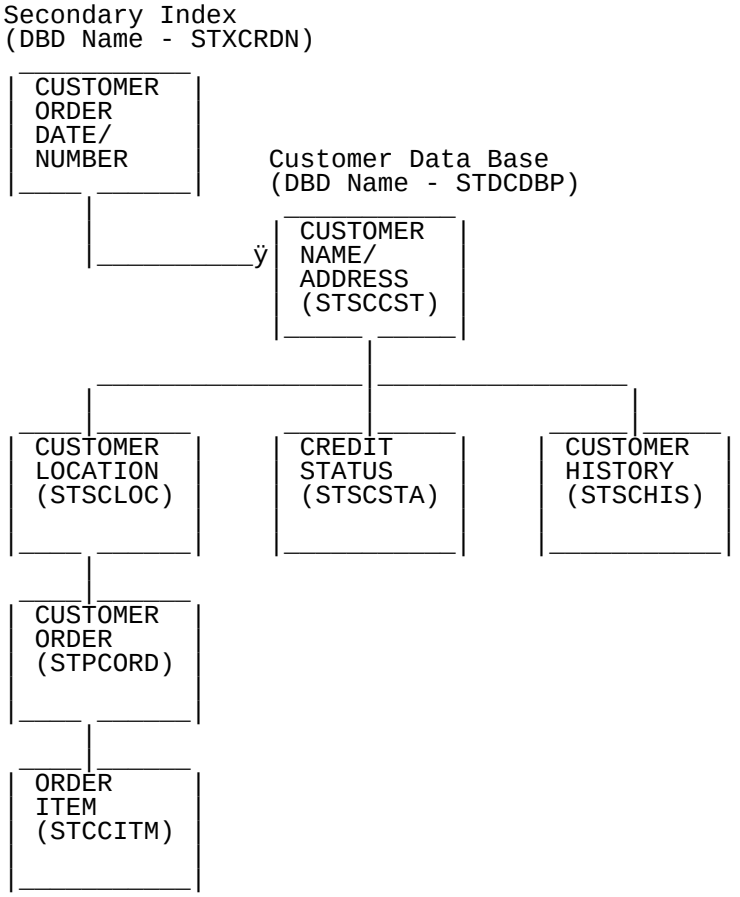
**Note:** For HD data bases, secondary indexes are defined using ACCESS control statements. The REF operand in ACCESS is used to name the secondary index when it is defined. For HDAM and HIDAM data bases, separate DBD definitions are required for secondary indexes. In this case, the NAME operand in the DBD statement is used to name the secondary index when it is defined.

Subtopics:



# 3.1.6.1 Example PSB for Secondary Index

Figure 36 shows a sample PSB using the PROCSEQ operand to name a secondary index.




```
PCB     TYPE=DB,DBDNAME=STDCDBP,PROCOPT=G,KEYLEN=50,POS=S,       X
        PROCSEQ=STXCRDN
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  LANG=ASSEM,PSBNAME=STBICLD
END
```

Figure 36. Example of PSB for a Secondary Index

# 3.1.7 Coding PSBs with Field Level Sensitivity

This section describes how to code PSBs with *field level sensitivity.* Two additional types of control statements are available:

- SENFLD statements are optionally used to identify only those fields in a physical segment to which an application program is sensitive and to build a new view of the segment for its exclusive use.

- VIRFLD statements are optionally used to define fields in the application program's view of a segment that do not exist in the physical view.

These statements are used in addition to those previously described for a basic PSB. (See "Coding Basic PSBs" at the beginning of this chapter.)

Figure 37 shows the structure of a batch input stream when defining a PSB with field level sensitivity.

***To Define a PSB with Field Level Sensitivity . . .***

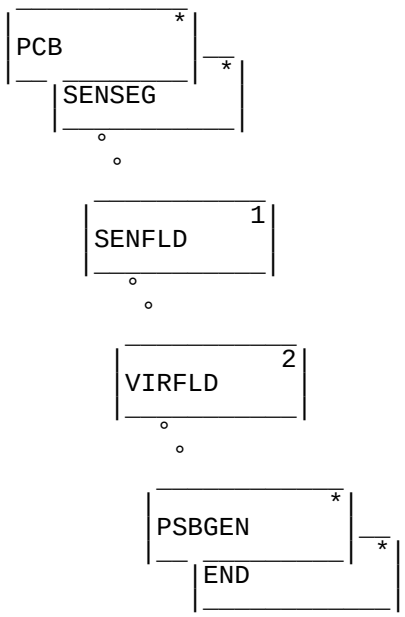



Figure 37. Batch Input Stream for PSBs with Field Level Sensitivity

1. Code one SENFLD statement for each field to which the application program is sensitive. Place them after the SENSEG statement for the

segment containing those fields.

2. Code one VIRFLD statement for each virtual field you want included in the application program's view of a segment. Place them along with the SENFLD statements after the appropriate SENSEG statements.

° The requirements for the PCB, SENSEG, and PSBGEN statements, and the Assembler language END statement are the same as those described for the basic PSB in Figure 30.

Subtopics:

- 3.1.7.1 Control Statements Format

---

---

# 3.1.7.1 Control Statements Format

This section explains how to code SENFLD and VIRFLD statements.

Subtopics:



### 3.1.7.1.1 SENFLD Statement

The format of the SENFLD statement is:

Table 73. SENFLD Statement Syntax for PCB Generation

| Statement | Keyword | Parameter | Description (See also Notes below). |
|---|---|---|---|
| SENFLD | NAME= proc. | ° ° ° ° ° ° ° ° | Name of this 'sensitive' field. |
| | ,BYTES= | _ _ _ | Length of this field.(1) |
| | ,START= | _ _ _ _ _ _ _ _ | Starting position of this field within the segment.(2) |
| | ,TYPE= | _ | Type of data.(3) |
| | ,RTNAME= | _ _ _ _ _ _ _ _ | Phase name of user-written field exit routine.(4) |
| | ,REPLACE= | _ _ _ | YES or NO. |

**Notes:**

1. The BYTES operand is optional. See the operand description for details.
2. The START operand is optional. If omitted, DL/I determines the fields starting position during PSB generation.
3. The TYPE operand is optional. If omitted, DL/I defaults to the data type specified in the physical view.
4. The RTNAME operand is optional. A field-exit routine is not required.

**NAME=**
is the name of a field you want included in your new view of a physical segment. The name specified must be identical to the related field name previously defined in a FIELD statement during DBD generation. This field name must start with an alphabetic character.

**BYTES=**
is 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 operand for field data types, E, D, and L. These data types have implicit lengths of 4, 8, and 16, respectively.

- The BYTES operand is optional for field data types H or F. If

omitted, DL/I assumes a field length of 2 for type H and 4 for type F.

- The BYTES operand is also optional for field data types X, P, C, and Z. If omitted, DL/I defaults to the same field length as in the physical view.

**START=**
is 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 1.

Subfields are permitted and can be defined 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 that defined field must start with an alphabetic character.

The START operand is optional. If omitted, 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=**
is the type of data that is to be contained in this field in your new view of the segment. It can be the same data type specified for this field in the FIELD statement during DBD generation, or it can be converted (in most cases) to another data type. 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 (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)

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.

***Rules and Restrictions***

- *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 status code 'KA' is returned.

  Character fields are padded with blanks on the right to extend field lengths. Truncation also occurs from the right. Truncation of non-blank characters results in the return of status code 'KB'. Only character field length conversion is performed if both the physical and user's view data types are 'C'.

- *Format checking*

  To ensure valid formats, packed and zoned decimal fields are pre-scanned prior to conversion. An invalid format results in the setting of the converted field to the null value, and the return of status code 'KC'.

- *Data type 'C'*

  Data contained in a field specified as type 'C' is considered to be in an "as is" format, and no conversion is made when the related field is specified as containing data of a different type. For instance, if a field in a physical segment is specified as type 'C; and the field in the application's view is specified as 'P', the data from the physical field is treated as though it is packed decimal. Only any necessary length adjustments are made.

- *Non-supported conversions*

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

- *Conversion Errors*

  If not corrected (reset) by the user exit routine, an uncorrected status code results in an immediate termination of the request. No more fields or segments are processed. No provision is made for correction of errors by the application program. See *User Field Exit Routine* in *DL/I DOS/VS Data Base Administration* for additional information about resetting the conversion status code.

**RTNAME=**

is the phase name of a user-written field exit routine in a 'DL/I user lib' that is given control whenever this field is retrieved or stored. See *User Field Exit Routine* in *DL/I DOS/VS Data Base Administration*, SH24-5011, for additional information.

This parameter, if specified, must be 1 through 8 characters. The first position must be alphabetic and the remainder alphanumeric.

**REPLACE=**
(or its abbreviation REPL) indicates whether or not the application program using this PSB may modify this field. The default value is REPLACE=YES. If you specify REPLACE=NO and an application program attempts to replace this field with a new value, DL/I returns a status code of KE.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 3.1.7.1.2 VIRFLD Statement

The format of the VIRFLD statement is:

Table 74. VIRFLD Statement Syntax for PCB Generation

| Statement | Keyword | Parameter | Description (See also Notes below). |
|---|---|---|---|
| VIRFLD | NAME= proc. | ° ° ° ° ° ° ° ° | Name of this 'virtual' field. |
| | ,BYTES= | _ _ _ | Length of this field.(1) |
| | ,START= | _ _ _ _ _ _ _ _ | Starting position of this field within its segment.(2) |
| | ,TYPE= | _ | Type of data.(3) |
| | ,VALUE= | _ _ _ _ _ _ _ _ | Initial value for this field.(4) |
| | ,RTNAME= | _ _ _ _ _ _ _ _ | Phase name of user-written field exit routine.(5) |

**Notes:**

1. The BYTES operand is optional. See the operand description for details.
2. The START operand is optional. If omitted, DL/I determines the fields starting position during PSB generation.
3. The TYPE operand is conditional. It must be specified if the VALUE operand is also specified.
4. The VALUE operand is optional. An initial value is not required for a virtual field.
5. The RTNAME operand is optional. A field-exit routine is not required.

**NAME=**
is the name of a virtual field you want included in your new view of a segment. This parameter must be 1 through 8 characters. The first position must be alphabetic and the remainder alphanumeric.

**BYTES=**
is the length (in bytes) of this virtual field. If specified, it must be a numeric value in the range of 1 through 256.

***Rules and Restrictions***

° The BYTES operand must be specified for field data types X, P, C, or Z (see TYPE parameter for field data types).

° The BYTES operand is optional for field data types H or F. If

omitted, DL/I assumes a field length of 2 for type H and 4 for type F.

- Do not specify the BYTES operand for field data types E, D, or L. These data types have implicit lengths of 4, 8 and 16, respectively.

**START=**
is the starting position of this virtual field. The starting position can be specified in terms of bytes relative to the beginning of your new view of the segment within which this virtual field is defined. In this case, it must be a numeric value in the range of 1 though 32767. For the first byte of a segment it is one.

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 that defined field must start with an alphabetic character.

The START operand is optional. If omitted, 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=**
is the type of data that is to be contained in the application program's view of this virtual field. This parameter must be specified if the VALUE parameter is used. The valid data types are:

| | |
|---|---|
| X | Hexadecimal (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.

***Rules and Restrictions***

- The TYPE parameter must be specified if the VALUE parameter is specified.

- If the VALUE parameter is specified for field data 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:

  For types X, H, F, and P - binary zeros on left

  For type Z - EBCDIC zeros on left

  For types E, D, and L - binary zeros on right

For type C - EBCDIC blanks on right.

**RTNAME=**
is the name of a user-written field exit routine in a 'DL/I user lib' that is given control whenever this virtual field is retrieved or stored. See *User Field Exit Routine* in *DL/I DOS/VS Data Base Administration*.

This parameter, if specified, must be 1 through 8 characters. The first position must be alphabetic and the remainder alphanumeric.

---

---

# 3.1.8 Control Statements Summary

Table 75 summarizes the DL/I control statements used to define a PSB with Field Level Sensitivity. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 75. Summary of Control Statements Used to Code Field Level Sensitivity

| [Label] | Statement | Operand |
|---|---|---|
| | SENFLD | NAME=field-name<br>[,BYTES=length]<br>[,START=starting-position]<br>[,TYPE={X\|H\|F\|P\|Z\|C\|E\|D\|L}]<br>[,RTNAME=routine-name]<br>[,REPLACE={*YES*}]<br>{NO } |
| | VIRFLD | NAME=field-name<br>[,BYTES=length]<br>[,START=starting-position]<br>[,TYPE={X\|H\|F\|P\|Z\|C\|E\|D\|L}]<br>[,VALUE=initial-value]<br>[,RTNAME=routine-name] |

# 3.2 Chapter 10. Doing a PSB Generation

This chapter assumes you have completed the *definition* step of PSBGEN and are now ready to do the *generation* step. The generation step may be invoked anytime after DL/I has been installed in your system.

PSBGEN is run as a standard system job. Note that it has two steps. Step 1 is a macro assembly execution. This step reads and expands your input statements and produces a PSB object module and an assembly listing. The DL/I macros used for step 1 reside in the 'DL/I prod lib'. In step 2, the generated PSB is cataloged and link-edited as a load module (type PHASE) in the 'DL/I user lib'.

Subtopics:



# 3.2.1 Preparing the Job Control Statements

The job control statements used to invoke the PSBGEN procedure are shown below. The placement of your PSB generation input statements within the job control statements is also shown.

```
// JOB PSBGEN
// OPTION CATAL,NODECK
// LIBDEF *,SEARCH='DL/I prod lib'
// LIBDEF PHASE,CATALOG='DL/I user lib'
// EXEC ASSEMBLY
        PCB     |
        SENSEG  |
        SENFLD  |  Put your PSB generation input statements here
        VIRFLD  |
        PSBGEN  |
        END
/*
// EXEC LNKEDT
/&
```

Only the minimum requirements for the job control statements are shown. They may contain additional information. Refer to *VSE/Advanced Functions System Control Statements* for detailed information about these statements.

# 3.2.2 Analyzing the Output

A standard assembly listing is created each time the PSB generation procedure is executed.

Any errors found, such as invalid parameters or the omission or invalid sequence of input statements will cause the PSB generation procedure to terminate prematurely and diagnostic messages to be printed.

You must then correct the errors and rerun the job. Every PSBGEN must be error free.

Diagnostic messages used to identify PSBGEN error conditions are described in *VSE/SP Messages and Codes*. As shown in the following list, each control statement uses a different prefix to identify its diagnostic messages. In all cases, the variable *nnn* is replaced with a unique identification number when a particular message is issued:

PCB statements use PCB*nnn*
SENSEG statements use SEG*nnn*
SENFLD statements use SFLD*nnn*
VIRFLD statements use VFLD*nnn*
PSBGEN statements use PGEN*nnn*

# 4.0 Part 4. Building Internal Control Blocks from DBDs and PSBs

This part assumes you have defined and generated your DBDs and PSBs as described in Part 2, "Describing the Characteristics of DL/I Data Bases" and Part 3, "Describing an Application Program View of a Data Base" respectively, and are now ready to use them to build internal control blocks.

To help you do this, DL/I provides a utility program. It is called the "Application Control Blocks Creation and Maintenance Utility." For brevity, this utility is simply called "block building" or "ACBGEN."

Chapter 11, "Doing the ACBGEN Procedure" in topic 4.1 explains what you must do to run ACBGEN.

Subtopics:



# 4.1 Chapter 11. Doing the ACBGEN Procedure

This chapter describes what you must do to run the ACBGEN utility.

The purpose of this utility is to create the internal control blocks that are necessary so that DL/I can provide the correct data base management services for your application programs.

You may invoke the ACBGEN utility anytime after DL/I has been installed in your system and after the required DBDs and PSBs have been generated and link-edited as load modules.

The application control blocks creation and maintenance utility is executed as a standard system application program.

Input to the utility consists of:

- *Job control statements.* The job control statements are read from the SYSIPT device and are used to execute the utility and, as a separate job step, to catalog and link-edit the object modules.

- *BUILD control statements.* One or more BUILD control statements must be placed within job control statements. These statements identify the PSBs used as input, whether the created blocks are to be written to SYSPCH or SYSLNK, whether DMBs (data management blocks) are to be generated, and whether the DL/I Documentation Aid facility is to be run.

  There is no maximum to the number of BUILD statements that may be submitted in a single job execution.

- *Previously-built control blocks.* The PSBs (named in the BUILD control statements) and their related DBDs.

Output from the utility consists of:

- *Messages and statistics.* The messages and statistics are written on the SYSLST device. The diagnostic messages used to describe ACBGEN error conditions are contained in the *DL/I DOS/VS Messages and Codes* manual.

- *Control blocks.* The newly built control blocks are cataloged and link-edited as a separate job step. These blocks contain one DMB and one utility PSB for each DBD, and one expanded PSB for each original PSB.

- *SQL/DS tables.* If the DL/I Documentation Aid facility is selected by specifying the USERID parameter of the control statement, DBD and PSB information is stored into SQL/DS (Structured Query Language/Data

System) tables. This facility is only available to users who have SQL/DS and the Interactive SQL (ISQL) facility installed on their VSE system and who have defined all required VSAM space for the tables.

The utility dynamically builds the name of the newly built control blocks. To avoid duplicate names, the new DMB and PSB names contain eight characters; the previously generated DBD or PSB name extended, if necessary, to seven characters with at-signs (@) followed by D for DMB or P for PSB. Additionally, some of the DL/I recovery and reorganization utility programs require a special PSB. For these PSBs, the first seven characters of the standard eight-character PSB name are identical to the DMB name, and the eighth character is U.

**Be aware that . . .**

You must rerun the ACBGEN utility each time you modify and regenerate a DBD or PSB. It is recommended that you use the DMB=YES operand on the BUILD statement to generate all DMBs when you do the rerun.

Subtopics:

# 4.1.1 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 for documenting DL/I data base definitions.

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, see "DL/I Documentation Aid (DA) Facility" later in this section.

# 4.1.2 Preparing BUILD Control Statements

The control statement requirements for this program are free form. A statement is coded as a card image and is contained in columns 1-71. The control statement may contain a name starting in column 1. The operation field must be preceded by and followed by one or more blanks. The operand field is composed of one or more PSB names and optionally an output destination and/or a DMB generation control parameter. It must be preceded by and followed by one or more blanks. Commas, parentheses, and blanks can be used only as delimiting characters. Comments may be written following the last operand of a control statement, separated from the operand by one or more blanks.

A control statement or PSB operand may be contained on more than one line by inserting a comma after the last PSB name of the first line, inserting a character other than a blank in column 72, and continuing the statement in column 16 of the next line. Columns 1-15 of the continuation line must be blank.

The format of the BUILD control statement is:

| | | |
|---|---|---|
| [label] | BUILD | PSB=(psbname,...)<br>[,OUT=LINK]<br>[,DMB={*COND*}]<br>{YES }<br>{NO }<br>[,USERID=SQLDBA/password] |

**label**
is optional and is useful only for documentation purposes. If specified it must be a 1- to 8-character alphameric value.

**BUILD**
indicates that blocks are to be built for the named PSBs.

**PSB=(psbname,...)**
means blocks are to be built for all PSBs named. As many of this type of control card as required may be submitted.

**OUT=LINK**
if specified in any BUILD statement the output destination of all the created control block(s) is SYSLNK. If the parameter is omitted, the output of all the control blocks is on SYSPCH.

**DMB={COND}**
**{YES }**
**{NO }**
controls the generation of DMBs for data bases referenced by the named PSBs. The default, COND, indicates that only those DMBs not currently present in the DL/I prod lib (or DL/I user lib) will be generated.

If you specify DMB=YES, all DMBs will be generated. If DMB=NO is

specified, no DMBs will be generated.

**Note:** 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.

**USERID=userid/password**
indicates the userid (SQLDBA) and password assigned for SQL/DS CONNECT authority. 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 SQL/DS tables for the DL/I Documentation Aid. 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.

---

---

# 4.1.3 Preparing Job Control Statements

Subtopics:



# 4.1.3.1 Writing Output to SYSLNK

If you specified OUT=LINK in any BUILD statement, the object modules will be written to SYSLNK. In this case, use the following job control statements to execute the ACBGEN utility and to catalog and link-edit the object modules.

Table 76. JCL for ACBGEN (output to SYSLNK)

| | | |
|---|---|---|
| // OPTION | CATAL | |
| [// ASSIGN | SYSLNK,cuu] | This statement defines the SYSLNK file that is to contain the object module output. This statement may be omitted if the standard physical unit is used for SYSLNK. |
| // EXEC | DLZUACB0,SIZE=xxxk | This statement identifies the program name of application control blocks creation and maintenance utility module and causes it to execute. If the Documentation Aid facility (USERID=) is specified on the BUILD statement, execution will occur in the multiple partition mode. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |
| Put your BUILD statement(s) here. | | |
| /*<br>// EXEC<br>/& | LNKEDT | This statement causes the object module on SYSLNK to be cataloged (OPTION CATAL) and link-edited. |

Following are the job control statements for execution in single partition mode if the Documentation Aid facility (USERID=) is specified on the BUILD statement:

Table 77. JCL for ACBGEN with SQL (output to SYSLNK)

| | | |
|---|---|---|
| // OPTION | CATAL | |
| [// ASSGN | SYSLNK,cuu] | This statement defines the SYSLNK file that is to contain the object module output. This statement may be omitted if the standard physical unit is used for SYSLNK. |
| // EXEC | PROC=xxxxxxxx | This statement executes the procedure that contains the identification statements for your SQL/DS database. |
| // EXEC | PGM=ARISQLDS,<br>SIZE=AUTO,<br>PARM='SYSMODE=S,<br>PROGNAME=DLZUACB0' | This statement invokes SQL/DS and passes to SQL/DS the program name of the application control blocks creation and maintenance utility module. Once the module receives control, it accesses SQL/DS in the same way as if executed in multi-partition mode. The SIZE=AUTO is |

| | | |
|---|---|---|
| | | required. |
| Put your BUILD statement(s) here. | | |
| /*<br>// EXEC<br>/& | LNKEDT | This statement causes the object module on SYSLNK to be cataloged (OPTION CATAL) and link-edited. |

# 4.1.3.2 Writing Output to SYSPCH

If you did not specify OUT=LINK in any BUILD statement, the object modules will be written to SYSPCH. In this case, use the following job control statements to execute the ACBGEN utility. A job stream including the necessary JOB and EXEC LNKEDT statements will also be included on SYSPCH to catalog and link-edit the object module(s). Note that there are no LIBDEF statements in this job stream.

Table 78. JCL for ACBGEN (output to SYSPCH)

| | | |
|---|---|---|
| [// ASSGN | SYSPCH,cuu] | This statement defines the SYSPCH file that is to contain the object module output. This statement may be omitted if the standard physical unit is used for SYSPCH. |
| // EXEC | DLZUACB0,<br>SIZE=xxxx | This statement identifies the program name of the application control blocks creation and maintenance utility module and causes it to execute. If the Documentation Aid facility (USERID=) is specified on the BUILD statement, execution will occur in the multiple partition mode. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |
| Put your BUILD statement(s) here. | | |
| /*<br>/& | | |
| [// CLOSE | SYSPCH,cuu] | This statement causes SYSPCH to be closed. It may be omitted if SYSPCH is assigned to a unit record device. |
| [// ASSGN | SYSIN,cuu] | This statement temporarily assigns the logical system reader to a physical unit device. The physical unit assigned to SYSIN must be cuu. The referenced job stream is read and the object module(s) cataloged and link-edited. This statement must not be included if SYSIN is assigned to a unit record device (that is, card reader). It is the console operator's responsibility to reassign SYSIN after the job has completed execution. |

The following are the job control statements for execution in single partition mode if the Documentation Aid facility (USERID=) is specified on the BUILD statement:

Table 79. JCL for ACBGEN with SQL (output to SYSPCH)

| | | |
|---|---|---|
| [// ASSGN | SYSPCH,cuu] | This statement defines the SYSPCH file that is to contain the object module output. This statement may be omitted if the standard physical unit is used for |

| | | |
|---|---|---|
| | | SYSPCH. |
| // EXEC | PROC=xxxxxxxx | This statement executes the procedure that contains the identification statements for your SQL/DS database. |
| // EXEC | PGM=ARISQLDS,<br>SIZE=AUTO,<br>PARM='SYSMODE=S,<br>PROGNAME=DLZUACB0' | This statement invokes SQL/DS and passes to SQL/DS the program name of the application control blocks creation and maintenance utility module. Once the module receives control, it accesses SQL/DS in the same way as if executed in multi-partition mode. The SIZE=AUTO is required. |
| Put your BUILD statement(s) here. | | |
| /*<br>/& | | |
| [// CLOSE | SYSPCH,cuu] | This statement causes SYSPCH to be closed. It may be omitted if SYSPCH is assigned to a unit record device. |
| [// ASSGN | SYSIN,cuu] | This statement temporarily assigns the logical system reader to a physical unit device. The physical unit assigned to SYSIN must be cuu. The reference job stream is read and link-edited. This statement must not be included if SYSIN is assigned to a unit record device (that is, card reader). It is the console operator's responsibility to reassign SYSIN after the job has completed execution. |

# 4.1.4 ACBGEN Examples

***Example 1:***

```
// JOB BLOCKBLD
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// LIBDEF PHASE,CATALOG='DL/I user lib'
// LIBDEF OBJ,CATALOG='DL/I user lib'
// OPTION CATAL
// EXEC   DLZUACB0,SIZE=50K
   BUILD  PSB=PSB1,OUT=LINK,DMB=YES
/*
// EXEC LNKEDT
/&
```

In this example, control blocks for PSB1 are created. Output is to SYSLNK. The DMBs referenced by PSB1 (DBD1@@@D and DBD2@@@D) already exist. However, the physical data format has been changed for these data bases and the user has generated new DBDs for DBD1 and DBD2. (Remember, the DMB name dynamically generated by extending the DBD name to 7 characters with at-signs (@) and appending the alphabetic character D.) The job consists of two separate program executions:

1. The DL/I application control blocks creation and maintenance utility program. A PSB named PSB1@@@P is created for PSB1. DMBs named DBD1@@@D and DBD2@@@D are created from DBD1 and DBD2. Two special PSBs named DBD1@@@U and DBD2@@@U are created for use by the recovery and reorganization utility programs.

2. The system linkage editor, which catalogs and link-edits the five created object modules into the assigned libraries.

***Example 2:***

```
// JOB     BUILDBLK
// LIBDEF  *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN   SYSPCH,180
// EXEC    DLZUACB0,SIZE=50K
   BUILD   PSB=(PSB2,PSB3,PSB4)
/*
/&
// CLOSE   SYSPCH,180
// ASSGN   SYSIN,180
```

**Note:** SYSIN must be reassigned from the system console prior to executing the next job.

In example 2, control blocks for PSB2, PSB3, and PSB4 are created. Output is on SYSPCH. If any DMBs referenced by one of these PSBs do not exist in the assigned libraries, a DMB and special utility PSB are also created. The control blocks which include JOB and EXEC LNKEDT statements are written to SYSPCH which is assigned to tape for this example. When the ASSGN SYSIN,X"180" statement is sensed, the output job stream is read as input and the object modules are cataloged and link-edited.

***Example 3:***

```
// JOB BLOCKBLD
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib,SQL prod lib')
// LIBDEF PHASE,CATALOG='DL/I user lib'
// LIBDEF OBJ,CATALOG='DL/I user lib'
// OPTION CATAL
// EXEC   DLZUACB0,SIZE=50K
   BUILD  PSB=PSB1,OUT=LINK,DMB=NO,USERID=SQLDBA/'sqldbapw'
/*
// EXEC LNKEDT
/&
```

This example is identical to example 1 except the DL/I Documentation Aid facility is specified through the USERID= parameter. Suppression of DMB generation has also been requested (DMB=NO).

The USERID= parameter establishes SQL/DS CONNECT authority and indicates that DBD and PSB data information is to be created and inserted into the DL/I SQL/DS tables.

---

# 4.1.5 DL/I Documentation Aid (DA) Facility

The DL/I Documentation Aid facility is intended to provide an ease-of-use facility in documenting DL/I definitions that can be accessed directly by ISQL. This permits a user to monitor the structure of the data base and its definitions.

The DL/I Documentation Aid feature is available to DL/I users who have SQL/DS and ISQL installed on their VSE system. When the Documentation Aid facility is invoked, data base description (DBD) and program specification block (PSB) data information is automatically created for the DL/I data base. This information is inserted into a special group of SQL/DS tables.

Subtopics:



# 4.1.5.1 Installing the DL/I Documentation Aid Facility

The following three job streams and two Documentation Aid access modules are provided in the 'DL/I prod lib' for installing the DL/I Documentation Aid facility:

° DLZDATAB.A

- Acquires public DBSPACE named DLIDBDDB and creates the following DL/I DA SQL/DS tables for storing the Data Base Description (DBD) data:

| | |
|---|---|
| **DBDBASICDATA** | Contains the information supplied on the DL/I DBD and DATASET statements. |
| **DBDACCESSDATA** | Contains the information supplied on the DL/I ACCESS, LCHILD, and XDFLD statements for the primary randomized access, primary indexed access, and secondary indexed access for HD organization data bases. |
| **DBDSEGMENTDATA** | Contains the information supplied on the DL/I SEGM statements. |
| **DBDFIELDDATA** | Contains the information supplied on the DL/I FIELD statements. |

- It also acquires public DBSPACE named DLIPSBDB and creates the following DL/I DA SQL/DS tables for storing the Program Specification Blocks (PSB) data:

| | |
|---|---|
| **PSBBASICDATA** | Contains the information supplied on the DL/I PSBGEN statement. |
| **PSBPCBDATA** | Contains the information supplied on the DL/I PCB statement. |
| **PSBSEGMENTDATA** | Contains the information supplied on the DL/I SENSEG statements. |
| **PSBFIELDDATA** | Contains the information supplied on the DL/I SENFLD and VIRFLD statements. |

° DLZDANDX.A

- Creates the DL/I Documentation Aid SQL/DS table indexes.

° DLZDARTN.A

- Dataloads the DL/I Documentation Aid ISQL Routines into the Routine table.

° DLZDLBP.A

- The DL/I PSB Documentation Aid module.

° DLZDLBD.A

- The DL/I DBD Documentation Aid module.

Once the above job streams are executed and the modules are preprocessed, the Applications Control Blocks Creation and Maintenance Utility can be executed to collect the DL/I data attributes in the DL/I DA SQL/DS tables.

---



---

# 4.1.5.2 Retrieving and Modifying the DL/I Documentation Aid Job Streams

Subtopics:



## 4.1.5.2.1 Retrieving the Job Streams

To retrieve the job streams, the following punch job should be used:

```
// JOB PUNCH DL/I DOCUMENTATION AID JOB STREAMS
// EXEC   LIBR
 ACCESS SUBLIB='DL/I prod lib'
 PUNCH DLZDATAB.A FORMAT=NOHEADER
 PUNCH DLZDANDX.A FORMAT=NOHEADER
 PUNCH DLZDARTN.A FORMAT=NOHEADER
/*
/&
```

Sample job stream to punch the DL/I Documentation Aid Job Stream from the DL/I production library.

## 4.1.5.2.2 Modifying the Job Streams for Your Environment

The following procedures should be followed to modify the job streams to meet your local needs:

- All job control statements are set to invoke the jobs in SQL/DS single-partition mode. The appropriate changes must be made if SQL/DS multi-partition mode is used.

- The EXEC PROC= statement should be updated to include your Procedure Name for the SQL/DS data base identification statements.

- The installation DBA userid of "SQLDBA" and DBA password of "SQLDBAPW" are used in the following statements:

  - EXEC job control statements;

  - SQL/DS CONNECT statements;

  - Qualified table-names in the ISQL Query routines.

  - The userid of "SQLDBA" is required for the DLZDATAB.A and DLZDANDX.A job streams that create the access modules and for the DLZDATAB.A and DLZDANDX.A job streams that create the tables and indexes.

- Care should be taken that the sequence numbers (in columns 73-80) are not destroyed in the DATALOAD jobstream DLZDARTN.A.

- Add a /& job control statement at the end.

**For more information...**

about determining if the DBSPACE specified meets the requirements of your local environment, see the section titled "Example of Storage Requirements for DL/I Documentation Aid" in the *DL/I DOS/VS Data Base Administration*.

**Note:** If "FORMAT=NOHEADER" was not specified in the librarian PUNCH statements, then CATALOG and BKEND statements are added to the punched members. The appropriate changes should then be made.

# 4.1.5.3 Preprocessing the Documentation Aid Modules

A job stream similar to the following example must be run to preprocess the the DL/I PSB and DBD Documentation Aid modules.

```
// JOB PREP DLZDLBP AND DLZDLBD
// LIBDEF *,SEARCH='(DL/I prod lib,SQL prod lib)'
// EXEC PROC='sqllabel'
// EXEC PGM=ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,PROGNAME=ARIPRPA/PREPNAM*
               E=DLZDLBPP,USERID=SQLDBA/'sqldbapw''
READ MEMBER DLZDLBP
/*
// EXEC PGM=ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,PROGNAME=ARIPRPA/PREPNAM*
               E=DLZDBLDP,USERID=SQLDBA/'sqldbapw''
READ MEMBER DLZDLBD
/*
/&
```

Subtopics:

- 4.1.5.3.1 Modifying the Job Stream for Your Environment

 *© Copyright IBM Corp. 1981, 1989*

## 4.1.5.3.1 Modifying the Job Stream for Your Environment

The following procedures should be followed to modify the job stream to meet your local needs:

° The job control statements are set to invoke the job in SQL/DS single partition mode. Make the appropriate changes if SQL/DS multi-partition mode is used.

° The EXEC PROC= statement statement should be updated to include your procedure name for the SQL/DS data base identification statement.

° The installation DBA userid of 'SQLDBA' and its password of 'SQLDBAPW' are used in the EXEC job control statement.

- The userid 'SQLDBA' is required

- the password should be changed to the current password of SQLDBA

° Because the modules' object codes and phases are already cataloged in the DL/I libraries at DL/I installation time, output to SYSPCH and SYSLST can be suppressed using an // UPSI job control statement (SQL/DS, Release 1) or the PRINT/NOPRINT and PUNCH/NOPUNCH options (Release 2).

**For more information...**

on how to adapt the DL/I documentation aid facilities to your local SQL/DS environment see *DL/I DOS/VS Release Guide*.

# 4.1.5.4 SQL/DS Documentation Aid Tables

The following SQL/DS tables contain information on the data base description (DBD) and describe the physical characteristics of the DL/I data base:

**DBDBASICDATA** Names the data base, its organization, and defines each data file that makes up the data base. Only one DBD and one DATASET statement per DBD generation.

**DBDACCESSDATA** Defines either the primary randomized access, primary indexed access, or secondary indexed access for HD organization DL/I data bases.

**DBDSEGMENTDATA** Defines all occurrences of a segment type that are placed in the data base being defined. It defines a segment type, the segment's position in a data base hierarchy, the physical characteristics of the segment, and how the segment is related to other segments. Maximum of 255 SEGMs per DBD generation.

**DBDLCHILDDATA** Defines a logical relationship between two segment types. Maximum of 255 LCHILDs per DBD generation.

**DBDFIELDDATA** Defines the fields within a segment. Maximum of 255 FIELDs per segment and maximum of 1020 FIELDs per DBD generation.

The following SQL/DS tables contain information on the program specification block (PSB) and describe the program and its use of logical data structures.

**PSBBASICDATA** Names the Program Specification Block and the language of the application program using this PSB. Only one PSBGEN statement for a PSB generation.

**PSBPCBDATA** Defines the Program Communication Block within the PSB. There must be at least one PCB for each physical or logical data base defined in the DBDBASICDATA table. There is a maximum of 255 PCBs for a PSB generation.

**PSBSEGMENTDATA** Identifies the segments within the data base to which the PCB is sensitive. There is a maximum of 255 SENFLDs for a PCB.

**PSBFIELDDATA** Identifies the fields in a physical segment which are included in this logical view of the segment. It also defines virtual fields that do not exist in the physical segment, but are used in the PCB's view of the segment. There is a maximum of 255 SENFLDs or VIRFLDs for a segment and 4095 for a PSB generation.

A detailed description of the DBD and PSB SQL/DS tables is provided in the following sections.

*DBD BASIC DATA*

**SQL/DS Table Name**: DBDBASICDATA

**Explanation**: This SQL/DS table contains the information that describes the data base, its organization, and the data files that make up the data base.

This table is based on the DL/I DBD and DATASET macro definitions.

| Name | Type | Explanation & Notes |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the DBD being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| DBD.NAME | CHAR | (KEY)<br>The name of the data base being defined. Must be 1 - 7 characters long, first must be alphabetic and no at-sign (@). |
| DBACCESS | CHAR | Access method for this data base. Must be one of the following: HIDAM, HDAM, INDEX, LOGICAL, HISAM, SHISAM, HSAM, or SHSAM. HD defaults to HDAM or HIDAM. Note: If the original specification was HSAM and only a root segment was specified, the access method is changed to SHSAM at DBDGEN time and will also be displayed as SHSAM. |
| CIANCHOR | INTEGER | For HDAM: Anchor points per VSAM CI. Must be a value of 1 - 255, default value is 1. |
| PRIMARYCI | INTEGER | For HDAM: CIs in Root Addressable Area. Must be a value of 0 - 16777215. Value of 0 signifies no upper limit check for CIs. OVERLAP defaults to 0. |
| RILIMIT | INTEGER | For HDAM: Largest record put in the Root Addressable Area at one time. Must be a value of 1 - 16777215. |
| IMSCOMP | CHAR | Data base format to be compatible with IMS. Specify YES or NO, NO is default. |
| DDF1.NAME | CHAR | For HDAM/HIDAM: Defines the symbolic filename for the VSAM ESDS. Must be 1 - 7 characters long and the first alphabetic.<br>For HISAM/SHISHAM/INDEX: Defines the symbolic filename for the KSDS. Must be 1 - 7 characters long and the first alphabetic.<br>For HSAM/SHSAM: Defines the input symbolic filename. Must be 1 - 7 characters long and the first alphabetic. |

| | | |
|---|---|---|
| DEVICE | CHAR | The physical device on which the data base is stored. The following are valid device types: FBA, 3380, 3375, 3350, 3340, 3330, 2314, or TAPE. |
| DDF2.NAME | CHAR | For HSAM/SHSAM: Defines the output symbolic filename. Must be 1 - 7 characters long and the first alphabetic. |
| DEVADDR1 | CHAR | For HSAM/SHSAM: Specifies the symbolic tape unit for an input file. Must be in the format: SYSnnn. |
| DEVADDR2 | CHAR | For HSAM/SHSAM: Specifies the symbolic tape unit for an input file. Must be in the format: SYSnnn. |
| OVFLW.NAME | CHAR | For HISAM: Defines the symbolic filename for the VSAM ESDS. Must be 1 - 7 characters long and the first alphabetic. |
| BLOCK1 | INTEGER | For HDAM/HIDAM: CI size - multiple of 512 up to 8192; multiple of 2048 up to 30720.<br>For HISAM/SHISAM/INDEX: Maximum KSDS records per CI. |
| BLOCK2 | INTEGER | For HISAM: Maximum ESDS records per CI. |
| RECORD1 | INTEGER | For HISAM/SHISAM/INDEX: KSDS record length.<br>For HSAM/SHSAM: Input logical record length. Must be a multiple of 2 up to 32766. |
| RECORD2 | INTEGER | For HISAM: ESDS record length.<br>For HSAM/SHSAM: Output logical record length. Must be a multiple of 2 up to 32766. |
| SCAN | INTEGER | For HDAM/HIDAM: The number of cylinders or blocks to be scanned in both directions when searching for available space. Cylinders: 0 - 255, default: 3. FBA Blocks: 0 - 32767, default approximately to 3 cylinders. |
| FREEBLOCK | INTEGER | For HDAM/HIDAM: Number of control intervals to be left free during load. Value must be 0 - 100, excluding 1, default is 0. |
| FREEPERCENT | INTEGER | For HDAM/HIDAM: Percent of each control interval to be left free during load. Value must be 0 - 99, default is 0. |

*DBD ACCESS DATA*

**SQL/DS Table Name**: DBDACCESSDATA

**Explanation**: This SQL/DS table contains the information that describes the access to HD organization DL/I data bases (randomized, primary or secondary).

This table is based on the DL/I ACCESS and XDFLD macro definitions.

| **Name** | **Type** | **Explanation & Notes** |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |

| | | |
|---|---|---|
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the DBD being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| DBD.NAME | CHAR | (KEY)<br>The name of the data base being defined. Must be 1 - 7 characters long, the first must be alphabetic and no at-sign (@). |
| SEGM.NAME | CHAR | (KEY)<br>Root or associated segment name. Must be 1 - 8 characters long and the first must be alphabetic. |
| INDEXDB.NAME | CHAR | (KEY)<br>DBD name for the index data base. Must be 1 - 7 characters long, the first alphabetic, and no at-sign (@). Null if primary randomized access is defined. |
| INDEXSEGM.NAME | CHAR | The name of the index pointer or index target segment. Must be 1 - 8 characters long and the first must be alphabetic. |
| INDEXFLD.NAME | CHAR | The name of the secondary index or sequence field. Must be 1 - 8 characters long and the first must be alphabetic. |
| POINTER | CHAR | Identifies the indexed or indexing segment. Must be "INDX" for the indexed segment and "SNGL" for the indexing segment. |
| SEQVALUE | VARCHAR | Value of sequence field is unique or duplicate. Must be 1-9 characters long. Must be one of the following: UNIQUE or U; DUPLICATE or D. UNIQUE is the default. |
| RANDMOD.NAME | CHAR | Randomizing module name. Must be 1 - 8 characters long, and the first must be alphabetic. Required for randomized access. |
| SRCHFLD1.NAME | CHAR | The name of the sequence field in the root segment for primary index or the name of a field in the index source segment to be used as the first search field in a secondary index. Must be 1 - 8 characters long. |
| SRCHFLD2.NAME | CHAR | The name of a field in the index source segment to be used as the second search field in a secondary index. Must be 1 - 8 characters long. |
| SRCHFLD3.NAME | CHAR | The name of a field in the index source segment to be used as the third search field in a secondary index. Must be 1 - 8 characters long. |
| SRCHFLD4.NAME | CHAR | The name of a field in the index source segment to be used as the fourth search field in a secondary index. Must be 1 - 8 characters long. |
| SRCHFLD5.NAME | CHAR | The name of a field in the index source segment to be used as the fifth search field in a secondary index. Must be 1 - 8 characters long. |
| SUBSEQ1.NAME | CHAR | Specifies the name of the first subsequence data field within the source segment. Must be 1 - 8 characters long. |
| SUBSEQ2.NAME | CHAR | Specifies the name of the second subsequence data field within the source segment. Must be 1 - 8 characters long. |
| SUBSEQ3.NAME | CHAR | Specifies the name of the third subsequence data field within the source segment. Must be 1 - 8 characters long. |
| SUBSEQ4.NAME | CHAR | Specifies the name of the fourth subsequence data field within the source segment. Must be 1 |

| | | |
|---|---|---|
| | | - 8 characters long. |
| SUBSEQ5.NAME | CHAR | Specifies the name of the fifth subsequence data field within the source segment. Must be 1 - 8 characters long. |
| DDATA1.NAME | CHAR | Specifies the name of the first index source segment field that is regarded as duplicated data in the index data base. Must be 1 - 8 characters long. |
| DDATA2.NAME | CHAR | Specifies the name of the second index source segment field that is regarded as duplicated data in the index data base. Must be 1 - 8 characters long. |
| DDATA3.NAME | CHAR | Specifies the name of the third index source segment field that is regarded as duplicated data in the index data base. Must be 1 - 8 characters long. |
| DDATA4.NAME | CHAR | Specifies the name of the fourth index source segment field that is regarded as duplicated data in the index data base. Must be 1 - 8 characters long. |
| DDATA5.NAME | CHAR | Specifies the name of the fifth index source segment field that is regarded as duplicated data in the index data base. Must be 1 - 8 characters long. |
| ISS.NAME | CHAR | The name of the index source segment. Must be 1 - 8 characters long and the first must be alphabetic. |
| SUPVALUE | CHAR | Suppress the creation of a secondary index entry if value specified is equal to the data in SEQFLDs. It can be a 1-byte self-defining value (that is: X"10," C"Z," B"10001001") or the words BLANK or ZERO. Note: This column value will always be displayed as a hexadecimal value. |
| SUPRTN.NAME | CHAR | Name of the routine to suppress creation of an index entry. Must be 1 - 8 characters long and the first must be alphabetic. |

**Note:** At least one ACCESS statement is required for HD organization and it must follow the DATASET statement. Any additional ACCESS statements must follow the required one.

*DBD SEGMENT DATA*

**SQL/DS Table Name**: DBDSEGMENTDATA

**Explanation**: This SQL/DS table contains the information that describes all occurrences of a segment type in the data base.

This table is based on the DL/I SEGM macro definitions.

| **Name** | **Type** | **Explanation & Notes** |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format, "HHMMSSFF," as calculated for this record. This time will be |

| | | |
|---|---|---|
| | | used for all other records associated with the DBD being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| DBD.NAME | CHAR | (KEY)<br>The name of the data base being defined. Must be 1-7 characters long, the first must be alphabetic and no at-sign (@). |
| SEGM.NAME | CHAR | (KEY)<br>Defines the segment name. Must be 1-8 characters long and the first alphabetic. |
| PARENT.NAME | CHAR | Defines the physical parent for this segment. Must be 1-8 characters long and the first alphabetic. If this is the root segment, must be null. |
| PCPOINTER | CHAR | If other than a root segment, specifies the pointers to be reserved in the physical parent segment. Valid entries: SNGL if one pointer to be reserved, DBLE if two pointers to be reserved. Default is SNGL. |
| LPSEGM.NAME | CHAR | Defines the logical parent segment. Must be 1 - 8 characters long and the first alphabetic. |
| IMSCOMPV | CHAR | The character "V" for upward compatibility to IMS. |
| LPDB.NAME | CHAR | Defines the logical data base name. Must be 1 - 7 characters long and the first alphabetic and no at-sign (@). |
| MAXBYTES | INTEGER | For fixed length segments: Length of the data portion of this segment. For variable length segment: Maximum length of segment. Maximum length = 4068. |
| MINBYTES | INTEGER | For HDAM/HIDAM: Minimum length of a variable length segment. Minimum length = 4. |
| COMPRTN.NAME | CHAR | For HDAM/HIDAM: Compression routine name. Must be 1 - 8 characters long and the first alphabetic. |
| IMSCOMPD | CHAR | For HDAM/HIDAM: The character "D" for upward compatibility with IMS. |
| INITTERM | CHAR | For HDAM/HIDAM: Value "INIT" if initialization and termination processing control required by the segment compression routine. |
| PHYSICALPTR | CHAR | For HDAM/HIDAM: Pointers to be reserved for the physical segments, logical parent segments, and virtual logical child segments. Valid values: TWIN (T), TWINBWD (TB), NOTWIN (NT), or PAIRED. Default: TWIN (T). |
| LOGICALPTR | CHAR | For HDAM/HIDAM: Pointers to be reserved for the logical twin segments. Valid values: LTWIN (LT) or LTWINBWD (LTB). |
| INSTRULE | CHAR | For HDAM/HIDAM/HISAM/SHISAM: Insert rules if part of a logical relationship. Valid values: P, L, or V. If omitted, L is the default. |
| DELRULE | CHAR | For HDAM/HIDAM/HISAM/SHISAM: Delete rules if part of a logical relationship. Valid values: P, L, or V. If omitted, L is the default. |
| REPLRULE | CHAR | For HDAM/HIDAM/HISAM/SHISAM: Replace rules if part of a logical relationship. Valid values: P, L, or V. If omitted, L is the default. |
| FLHRULES | CHAR | For HDAM/HIDAM/HISAM/SHISAM: Defines where new occurrences are to be inserted. Valid values: FIRST, LAST, or HERE. Default is LAST. |

| | | |
|---|---|---|
| SRCSEGM1.NAME | CHAR | For HDAM/HIDAM/LOGICAL: The name of the source segment in the physical data base which represents this virtual-logical segment. Must be 1-8 characters long and the first alphabetic. |
| IMSCOMPD1 | CHAR | For HDAM/HIDAM/LOGICAL: The character "D" for upward compatibility to IMS. |
| SRCDBD1.NAME | CHAR | For HDAM/HIDAM/LOGICAL: The name of the physical data base containing the source segment. Must be 1-7 characters long and the first alphabetic. |
| SRCSEGM2.NAME | CHAR | For LOGICAL: The name of the destination parent segment. Must be 1-8 characters long and the first alphabetic. |
| IMSCOMPD2 | CHAR | For LOGICAL: The character "D" for upward compatibility with IMS. |
| SRCDBD2.NAME | CHAR | For LOGICAL: The name of the data base containing the destination parent segment. Must be 1-7 characters long and the first must be alphabetic. |
| SEGCODE | INTEGER | The hierarchical sequence assigned to segments from the top to bottom, left to right order in a data base. The sequence is from 1, the root segment, up to 255. |
| SEGLEVEL | INTEGER | The depth in the hierarchical structure at which a segment is located. The levels are numbered from 1 to 15. |

**Note:** At least one SEGM statement is required and must follow the ACCESS statement if organization is HD, or the DATASET statement for any other organization.

For SHSAM/SHISAM/INDEX organizations, maximum of 1 SEGM per data base.
For any other organization, maximum of 225 SEGMs per data base.

*DBD LCHILD DATA*

**SQL/DS Table Name**: DBDLCHILDDATA

**Explanation**: This SQL/DS table contains the information that describes the relationship between two segments (either an index relationship or a logical relationship between two segment types).

This table is based on the DL/I LCHILD macro definition, which is used only by HDAM,HIDAM or INDEX access methods.

| **Name** | **Type** | **Explanation & Notes** |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF" as calculated for this record. This time will be used for all other records associated with the DBD being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |

| | | |
|---|---|---|
| DBD.NAME | CHAR | (KEY)<br>The name of the data base being defined. Must be 1-7 characters long and the first must be alphabetic and no at-sign (@). |
| SEGM.NAME | CHAR | (KEY)<br>Defines the segment name. Must be 1-8 characters long and the first alphabetic. |
| LSEGM.NAME | CHAR | (KEY)<br>For HDAM/HIDAM: Defines the name of the segment. Must be 1-8 characters long and the first alphabetic.<br>INDEX: Defines the name of the root or index target segment. Must be 1-8 characters long and the first alphabetic. |
| LDBD.NAME | CHAR | (KEY)<br>For HDAM/HIDAM: Defines the name of the data base. Must be 1-7 characters long and the first alphabetic.<br>INDEX: The name of the DBD of the index target segment. |
| POINTER | CHAR | For HDAM/HIDAM: Specifies the segment pointers. Valid value are NONE, SNGL or DBLE. Default: NONE. |
| PAIRSEGM.NAME | CHAR | For HDAM/HIDAM: Defines the name of the virtual logical child. Must be 1-8 characters long and the first alphabetic. |
| FLHRULES | CHAR | For HDAM/HIDAM: Defines where new occurrences of the segment are to be inserted. Valid values: FIRST, LAST, or HERE. |

**Note:** LOGICAL RELATIONSHIP - LCHILD establishes the logical relationship between segments. It follows the SEGM statement or its FIELD statements of the logical parent.

*DBD FIELD DATA*

**SQL/DS Table Name**: DBDFIELDDATA

**Explanation**: This SQL/DS table contains the information that describes the fields within a segment that may be referred to by an application program in a DL/I call segment search argument.

This table is based on the DL/I FIELD macro definitions.

| **Name** | **Type** | **Explanation & Notes** |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the DBD being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| DBD.NAME | CHAR | (KEY)<br>The name of the data base being defined. Must be 1-7 characters long, the first must be |

| | | |
|---|---|---|
| | | alphabetic and no at-sign (@). |
| SEGM.NAME | CHAR | (KEY)<br>Defines the segment name. Must be 1-8 characters long and the first must be alphabetic. |
| FLD.NAME | CHAR | (KEY)<br>Defines the name of the field. Must be 1-8 characters long. |
| SEQFIELD | CHAR | Specifies if the field is a sequence (key) field. Valid value: SEQ. |
| SEQTYPE | CHAR | Specifies the type of sequence (key) field. Valid values: U or M. Default value is U. |
| BYTES | INTEGER | Specifies the length of field. Must be in the range of 1-256 or 1-236 for HISAM/SHISAM/HIDAM indexed root sequence fields. |
| POSITION.NAME | CHAR | Specifies the name of the previously defined field in the segment that will be used for the starting position of this field. |
| POSITION | INTEGER | Specifies the starting position of this field within the segment. Must be in the range of 1-32767. First byte of segment is 1. |
| DATATYPE | CHAR | Specifies the type of data contained in the field. Valid values are: C, X, P, Z, E, D, or L. H and F default to X. |

**Note:** FIELD defines the fields within the segment. It follows its related SEGM statement. Maximum of 255 per SEGM or 1020 per DBD generation.

*PSB BASIC DATA*

**SQL/DS Table Name**: PSBBASICDATA

**Explanation**: This SQL/DS table contains the information that describes the characteristics of the application program.

This table is based on the DL/I PSBGEN macro definitions.

| **Name** | **Type** | **Explanation & Notes** |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the DBD being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| PSB.NAME | CHAR | (KEY)<br>Defines the name for the Program Specification Block. Must be 1 - 7 characters long, the first alphabetic, and no at-sign (@). |
| LANGUAGE | CHAR | Specifies the compiler language in which the application program is written. Valid values: |

| | | |
|---|---|---|
| | | COBOL, PL/I, ASSEM and RPG. Note: The only values to be displayed for this column are PL/I or COBOL, since PSBGEN bit setting identifies the program language as PL/I or non-PL/I. |

**Note:** This is equivalent to the PSBGEN statement. It follows the last set of PCB/SENSEG/SENFLD/VIRFLD statements.

*PSB* *PCB* *DATA*

**SQL/DS Table Name**: PSBPCBDATA

**Explanation**: This SQL/DS table contains the information that describes the interfaces to a DL/I data base for an application program.

This table is based on the DL/I PCB macro definitions.

| Name | Type | Explanation & Notes |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the PSB being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| PSB.NAME | CHAR | (KEY)<br>Defines the name of the Program Specification Block associated with this PCB. Must be 1-7 characters long and the first alphabetic. The at-sign (@) must not be used. |
| PCBNUMBER | INTEGER | (KEY)<br>Specifies the offset in the PSB for this PCB entry. |
| DBTYPE | CHAR | The characters "DB" for upward compatibility with IMS. |
| DBD.NAME | CHAR | Specifies the physical or logical data base name. Must be 1-7 characters long and the first alphabetic. The at-sign (@) must not be used. |
| PROCOPT | CHAR | Specifies the processing options for this sensitive segment. Valid values: G, I, R, D, A, GO, N, T, P, E, L, or LS. |
| KEYLEN | INTEGER | Specifies the longest concatenated key for a hierarchical path. Must be in range 1-32767. |
| POSITIONING | CHAR | Specifies the positioning to be maintained by DL/I. Valid values: SINGLE (S), or MULTIPLE (M). Default: SINGLE (S). |
| PROCSEQ | CHAR | Specifies the name of the secondary index that is used to process the data base. Must be 1-7 characters long and the first alphabetic. The at-sign (@) must not be used. |

**Note:** One PCB for each physical and logical hierarchical data structure.

A maximum of 255 PCBs per PSB generation.

*PSB SEGMENT DATA*

**SQL/DS Table Name**: PSBSEGMENTDATA

**Explanation**: This SQL/DS table contains the information that describes the logical data structures and the segments to which the application program is sensitive.

This table is based on the DL/I SENSEG macro definitions.

| Name | Type | Explanation & Notes |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the PSB being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| PSB.NAME | CHAR | (KEY)<br>Defines the name of the Program Specification Block associated with this PCB. Must be 1-7 characters long and the first alphabetic. The at-sign (@) must not be used. |
| PCBNUMBER | INTEGER | (KEY)<br>Specifies the offset in the PSB for this PCB entry. Must be in range 1-255. |
| SEGNUMBER | INTEGER | Specifies the offset in the PCB for this segment entry. Must be in range 1-255. |
| SEG.NAME | CHAR | (KEY)<br>Specifies the name of a sensitive segment. Must be 1-8 characters long and the first must be alphabetic. |
| PARENT.NAME | CHAR | Specifies the name of the parent of this segment. Segment name must be 1-8 characters long and the first must be alphabetic. If root segment, must be null. |
| PROCOPT | CHAR | Specifies the processing options to override those in the PCB statement. Valid values: G, I, R, D, A, P, or E. |

**Note:** Defines the hierarchically related set of segments. It follows its related PCB. A maximum of 255 SENSEG statements allowed per PCB.

*PSB FIELD DATA*

**SQL/DS Table Name**: PSBFIELDDATA

**Explanation**: This SQL/DS table contains the information that describes the fields in the physical segment to which the application program is sensitive and to build a new view of the segment for its exclusive use.

It also contains the information that describes the fields in the application program's view of a segment that do not exist in the physical view.

This table is based on the DL/I SENFLD and VIRFLD macro definitions.

| Name | Type | Explanation & Notes |
|---|---|---|
| CREATEDATE | CHAR | Creation date of this record, in the format "YYYYMMDD." |
| CREATEUSER | CHAR | SQL/DS CONNECT authority userid of the creator of the record. Must be SQLDBA. |
| CHANGEDATE | CHAR | Date the record was changed, in the format "YYYYMMDD." |
| CHANGETIME | CHAR | The time of day, in the format "HHMMSSFF," as calculated for this record. This time will be used for all other records associated with the PSB being processed. |
| CHANGEUSER | CHAR | SQL/DS CONNECT authority userid of the modifier of the record. Must be SQLDBA. |
| PSB.NAME | CHAR | (KEY)<br>Defines the name of the Program Specification Block associated with this PCB. Must be 1-7 characters long and the first alphabetic. The at-sign (@) must not be used. |
| PCBNUMBER | INTEGER | (KEY)<br>Specifies the offset in the PSB for this PCB entry. Must be in range 1-255. |
| SEGNUMBER | INTEGER | Specifies the offset in the PCB for this segment entry. Must be in range 1-255. |
| SEG.NAME | CHAR | (KEY)<br>Specifies the name of a sensitive segment. Must be 1-8 characters long and the first must be alphabetic. |
| FLDNUMBER | INTEGER | Specifies the offset in the segment for this segment entry. Must be in range 1-255. |
| FLD.NAME | CHAR | (KEY)<br>Specifies the name of the sensitive field. Must be 1-8 characters long and the first alphabetic. |
| BYTES | INTEGER | Specifies the length of the field. Must be in range 1-256. |
| POSITION.NAME | CHAR | Specifies the name of the previously defined field in the segment that will be used for the starting position of this field. Must be 1-8 characters long and first alphabetic. |
| POSITION | INTEGER | Specifies the starting position of the field within the segment. Must be in range 1-32767. |
| DATATYPE | CHAR | Specifies the type of data this field is to contain. Valid values: C, X, P, Z, E, D, or L. H and F default to X. |
| EXITRTN.NAME | CHAR | Specifies the name of the field exit routine. Must be 1-8 characters long and first must be alphabetic. |
| FLDTYPE | CHAR | Specifies the field type. Must be "S" for sensitive field or "V" for virtual field. |
| REPLACE | CHAR | Specifies whether or not the field can be modified. Valid values: YES or NO. Default is YES. |
| INTEGERVALUE | INTEGER | Specifies the initial value for this field. |

| | | |
|---|---|---|
| | | Data type must be X, H, or F. |
| DECIMALVALUE | DECIMAL | Specifies the initial value for this field. Data type must be P or Z. |
| FLOATVALUE | FLOAT | Specifies the initial value for this field. Data type must be E, D, or L. Data types E and L are converted to data type D. |
| CHARVALUE | VARCHAR | Specifies the initial value for this field. Data type must be C. |

**Note:** Defines the sensitive or virtual fields in an application program's view of a segment. Follows its related SENSEG statement. A maximum of 255 fields are allowed per segment and a maximum of 4095 fields are allowed per PSB generation.

---

---

# 4.1.5.5 Accessing Data from the Documentation Aid Tables

The DL/I Documentation Aid includes sixteen ISQL sample query routines for accessing data from the SQL/DS tables. For each query routine, there is a corresponding report routine. When an SQL/DS user executes one of the query routines, the respective table data will be displayed on the terminal screen. After the display is complete, the user can enter END to generate a printed report. If a printed report is not desired, the user should enter CANCEL to terminate routine processing.

Subtopics:



---

## 4.1.5.5.1 ISQL Sample Query and Report Routines

The following list identifies the routine name, required parameters, if any, and function for each of the sample query routines:

| Routine Name | Parameters | Function |
|---|---|---|
| DBDNAMES | None | Displays all the DL/I data base names processed during the DL/I ACBGEN. |
| DBDBASIC | dbdname | Displays the DL/I data base basic attributes, based on a data base name, as defined during the DL/I DBDGEN via the DBD and DATASET statements. |
| DBDACESS | dbdname | Displays the DL/I data base access attributes, based on a data base name, for Primary and Secondary Indexes, as defined during the DL/I DBDGEN via the DBD, ACCESS, LCHILD, and/or XDFLD statements. |
| DBDSEGM | dbdname | Displays all the DL/I data base segment names and their attributes, based on a data base name, as defined during the DBDGEN via the SEGM statement. |
| DBDLCHLD | dbdname | Displays all the DL/I data base logical relationships, based on a data base name, as defined during the DBDGEN via the LCHILD statement. |
| DBDFIELD | dbdname | Displays all the data base field names and their attributes, based on a data base name, as defined during the DBDGEN via the FIELD statement. |

| | | | |
|---|---|---|---|
| | DBDHIER | dbdname | Displays the hierarchical view of a DL/I data base based on a data base name. |
| | PSBNAMES | None | Displays all the DL/I PSB names processed during the DL/I ACBGEN. |
| | PSBPCB | psbname | Displays all the DL/I PCB attributes based on a PSB name, as defined during the PSBGEN via the PCB and PSBGEN statements. |
| | PSBSEG | psbname<br>pcbnumber | Displays all the DL/I sensitive segment names and their attributes, based on a PSB name and a PCB number, as defined during the PSBGEN via the SENSEG statement. |
| | PSBFIELD | psbname<br>pcbnumber | Displays all the DL/I sensitive field names and their attributes, based on a PSB name and a PCB number, as defined during the PSBGEN via the SENFLD and/or VIRFLD statements. |
| | PSBSEGV | psbname<br>pcbnumber<br>segment-name | Displays the logical view of a DL/I segment, based on a PSB name, PCB number, and a segment name, as defined during the PSBGEN or the DBDGEN. If a logical view is not defined at PSBGEN time, the physical view as defined during the DBDGEN will be displayed. |
| | WHEREDBD | dbdname | Displays the PSB name(s) and PCB number where the DBD name is referenced. |
| | WHERESEG | segment-name | Displays the PSB name(s) and/or DBD name(s) where the segment name is referenced. |
| | WHEREFLD | field-name | Displays the segment name(s), PSB name(s) and/or the DBD name(s) where the field name is referenced. |
| | DLISTATS | None | Displays the user creation and change statistics for |

```
|
  |                          |             | all DL/I DBD and PSB data.
|
|__________________________|_____________|_____________________________________________________|
```

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 4.1.5.5.2 Executing a Query Routine

Each query routine can be executed through the ISQL RUN command. The format for the RUN command is:

```
RUN routinename (parameter1 parameter2 parameter3)
```

For example if an SQL/DS user wanted to run the query routine named DBDNAMES to display all the DL/I data base names processed during the ACBGEN, the user would enter

```
RUN DBDNAMES
```

and press the ENTER key.

The information requested by this routine is displayed on the terminal screen. If a printed report is desired, the user can type END; if not, the user should type CANCEL.

To display all the DL/I sensitive segment names and their attributes for a particular PSB and PCB as defined during the PSBGEN, an SQL/DS user should enter

```
RUN PSBSEG (psbname pcbnumber)
```

where psbname is the name of the desired Program Specification Block and pcbnumber is the Program Communication Block number.

The query routines provided are considered the most useful, but are not intended to be comprehensive for any special information needs. Also, the user can prepare different query routines.

**For more information...**

on using SQL/DS and ISQL, see the *SQL/DS Terminal User's Reference* manual.

# 4.1.6 Examples of Output From the ISQL DA Run Routines

The following figures show the results of some ISQL RUN commands. All routines supplied by the DL/I Documentation Aid automatically print the results of the routine by entering the ISQL display command END. If a printed output is not wanted, the ISQL routine should be ended by entering CANCEL rather than END.

Note that the display of the NULL value has been changed via the ISQL command **SET NULL ''**.

```
     11/14/88                                DL/I DATA BASE BASIC ATTRIBUTES (PART 1)
PAGE 1

     DBD.NAME  DBACCESS  CIANCHOR  PRIMARYCI  RILIMIT  IMSCOMP  DDF1.NAME  DEVICE  DDF2.NAME
DEVADDR1  DEVADDR2  OVFLW.NAME
     --------  --------  --------  ---------  -------  -------  ---------  ------  ---------  -----
---  --------  ----------
     STDCDBP   HDAM             3        100      600  NO       STDCDBC    3380
```

```
     11/14/88                                  DL/I DATA BASE BASIC ATTRIBUTES (PART 2)
PAGE 1

     DBD.NAME  BLOCK1  BLOCK2  RECORD1  RECORD2  SCAN  FREEBLOCK  FREEPERCENT
     --------  ------  ------  -------  -------  ----  ---------  -----------
     STDCDBP     2048                                2          0            0
```

Figure 38. ISQL Command: RUN DBDBASIC (STDCDBP)

```
     11/14/88          DL/I PRIMARY RANDOMIZED, PRIMARY INDEXED, AND/OR SECONDARY INDEXED ACCESS
ATTRIBUTES (PART 1)      PAGE 1

     DBD.NAME  SEGM.NAME  INDEXDB.NAME  INDEXSEGM.NAME  INDEXFLD.NAME  POINTER  SEQVALUE
RANDMOD.NAME  SRCH NAMES(S)
     --------  ---------  ------------  --------------  -------------  -------  ---------  --------
----  --------------
     STDCDBP   STSCCST    STDCX1P       STICMNA         STXCMNA        INDX     DUPLICATE
STUCCNM

                          STDCX2P       STICRDN         STXCRDN        INDX     UNIQUE
STQCODN

                                                        STQCCNO                 UNIQUE     DLZHDC10
```

---

```
     11/14/88          DL/I PRIMARY RANDOMIZED, PRIMARY INDEXED, AND/OR SECONDARY INDEXED ACCESS
ATTRIBUTES (PART 2)      PAGE 1

     DBD.NAME  SEGM.NAME  INDEXDB.NAME  SUBSEQ NAME(S)  DDATA NAME(S)  ISS.NAME  SUPVALUE
SUPRTN.NAME
     --------  ---------  ------------  --------------  -------------  --------  --------  --------
---
     STDCDBP   STSCCST    STDCX1P       /CKSCCST                       STSCCST

                          STDCX2P                       /CKPCORD       STPCORD
```

---

Figure 39. ISQL Command: RUN DBDACESS (STDCDBP)

---

---

```
     11/14/88                                DL/I SEGMENTS AND THEIR ATTRIBUTES (PART 1)
PAGE 1

     DBD.NAME  SEGM.NAME  PARENT.NAME  PCPOINTER  LPSEGM.NAME  IMSCOMPV  LPDB.NAME  MAXBYTES
MINBYTES  COMPRTN.NAME  IMSCOMPD
     --------  ---------  -----------  ---------  -----------  --------  ---------  --------  -----
---  ------------  --------
```

```
     STDCDBP   STSCCST                SNGL                                          106
               STSCLOC    STSCCST     SNGL                                          106
               STPCORD    STSCLOC     SNGL                                           55
               STCCITM    STPCORD     SNGL       STPIITM      V         STDIDBP       38
               STSCSTA    STSCCST     SNGL                                           24
               STSCHIS    STSCCST     SNGL                                          130
53   DLZSAMCP
```

---

```
     11/14/88                                DL/I SEGMENTS AND THEIR ATTRIBUTES (PART 2)
PAGE 1

     DBD.NAME  SEGM.NAME  INITTERM  PHYSICALPTR  LOGICALPTR  INSTRULE  DELRULE  REPLRULE  FLHRULES
     --------  ---------  --------  -----------  ----------  --------  -------  --------  --------
     STDCDBP   STSCCST              TWIN                     L         L        L         LAST
               STSCLOC              TWINBWD                  L         L        L         LAST
               STPCORD              TWINBWD                  P         P        V         LAST
               STCCITM              TWINBWD      LTWINBWD    P         P        V         LAST
               STSCSTA              TWIN                     L         L        L         FIRST
               STSCHIS              TWINBWD                  L         L        L         LAST
```

---

```
     11/14/88                                DL/I SEGMENTS AND THEIR ATTRIBUTES (PART 3)
PAGE 1

     DBD.NAME  SEGM.NAME  SRCSEGM1.NAME  IMSCOMPD1  SRCDBD1.NAME  SRCSEGM2.NAME  IMSCOMPD2
SRCDBD2.NAME
     --------  ---------  -------------  ---------  ------------  -------------  ---------  -------
-----
     STDCDBP   STSCCST
               STSCLOC
               STPCORD
               STCCITM
               STSCSTA
               STSCHIS
```

---

Figure 40. ISQL Command: RUN DBDSEGM (STDCDBP)

---

```
     11/14/88                                DL/I FIELDS AND THEIR ATTRIBUTES
```

PAGE 1

| DBD.NAME | SEGM.NAME | FLD.NAME | SEQFIELD | SEQTYPE | BYTES | POSITION.NAME | POSITION | DATATYPE |
|---|---|---|---|---|---|---|---|---|
| STDCDBP | STSCCST | /CKSCCST | | | 6 | | 1 | |
| | | STQCCNO | SEQ | U | 6 | | 1 | C |
| | | STFCCA1 | | | 25 | STUCCNM | 7 | C |
| | | STUCCNM | | | 25 | | 7 | C |
| | | STFCCA2 | | | 25 | | 57 | C |
| | | STFCCA3 | | | 25 | | 82 | C |
| | STSCLOC | STQCLNO | SEQ | U | 6 | | 1 | C |
| | | STFCLNM | | | 25 | | 7 | C |
| | | STFCLA1 | | | 25 | | 32 | C |
| | | STFCLA2 | | | 25 | | 57 | C |
| | | STFCLA3 | | | 25 | | 82 | C |
| | STPCORD | STQCODT | | | 6 | STQCODN | 1 | C |
| | | STQCODY | | | 2 | STQCODN | 1 | C |
| | | /CKPCORD | | | 12 | | 1 | |
| | | STQCODN | SEQ | U | 12 | | 1 | C |
| | | STQCODM | | | 2 | | 3 | C |
| | | STQCODD | | | 2 | | 5 | C |
| | | STQCONO | | | 6 | | 7 | C |
| | | STFCORF | | | 25 | | 13 | C |
| | | STFCOIC | | | 6 | | 38 | C |
| | | STFCOAM | | | 12 | | 44 | C |
| | STCCITM | STKCIIN | | | 6 | | 1 | C |
| | | STQCILI | SEQ | U | 2 | | 7 | C |
| | | STFCIQO | | | 6 | | 9 | C |
| | | STFCIQS | | | 6 | | 15 | C |
| | | STFCIQB | | | 6 | | 21 | C |
| | | STFCIAM | | | 12 | | 27 | C |
| | STSCSTA | STFCSCL | | | 12 | | 1 | C |
| | | STFCSBL | | | 12 | | 13 | C |
| | STSCHIS | STGCSL | | | 2 | | 1 | X |
| | | STQCHDT | | | 6 | STQCHDN | 3 | C |
| | | STQCHDY | | | 2 | STQCHDN | 3 | C |
| | | STQCHDN | SEQ | U | 12 | | 3 | C |
| | | STQCHDM | | | 2 | | 5 | C |
| | | STQCHDD | | | 2 | | 7 | C |
| | | STQCHNO | | | 6 | | 9 | C |
| | | STFCHRF | | | 25 | | 15 | C |
| | | STFCHIC | | | 2 | | 40 | C |
| | | STFCHAM | | | 12 | | 42 | C |
| | | STFCLOS | | | 77 | | 54 | C |

---

Figure 41. ISQL Command: RUN DBDFIELD (STDCDBP)

---

11/14/88 DL/I DATA BASE HIERARCHICAL VIEW
PAGE 1

DATA BASE SEGMENT(S) PARENT LEVEL

```
---------  ----------  -----------  -----
STDCDBP    STSCCST                      1

           STSCLOC     STSCCST          2

           STSCSTA     STSCCST          2

           STSCHIS     STSCCST          2

           STPCORD     STSCLOC          3

           STCCITM     STPCORD          4
```

---

Figure 42. ISQL Command: RUN DBDHIER (STDCDBP)

---

```
11/14/88                          DL/I PROGRAM COMMUNICATION BLOCKS AND THEIR ATTRIBUTES
PAGE 1

PSB.NAME  PCBNUMBER  LANGUAGE  DBTYPE  DBD.NAME  PROCOPT  KEYLEN  POSITIONING  PROCSEQ
--------  ---------  --------  ------  --------  -------  ------  -----------  -------
STBICAP           1  COBOL     DB      STDIDBP   AP           50  SINGLE
                  2  COBOL     DB      STDCDBP   AP           50  SINGLE
```

---

Figure 43. ISQL Command: RUN PSBPCB (STBICAP)

---

```
11/14/88                                DL/I SENSITIVE SEGMENTS AND THEIR ATTRIBUTES
PAGE 1

PSB.NAME  PCBNUMBER  SEG.NAME  PARENT.NAME  PROCOPT
--------  ---------  --------  -----------  -------
STBICAP           2  STSCCST                AP
                     STSCLOC   STSCCST      AP
                     STPCORD   STSCLOC      AP
                     STCCITM   STPCORD      AP
                     STSCSTA   STSCCST      AP
                     STSCHIS   STSCCST      AP
```

Figure 44. ISQL Command: RUN PSBSEG (STBICAP 2)

```
11/14/88                                    DL/I LOGICAL VIEW OF A SEGMENT
PAGE 1

PSBNAME   PCBNUMBER  SEGMENT    FIELD(S)  BYTES   START  TYPE
--------  ---------  ---------  --------  -----  ------  ----
STBICAP           2  STSCCST    /CKSCCST      6       1
                                STQCCNO       6       1  C
                                STFCCA1      25       7  C
                                STUCCNM      25       7  C
                                STFCCA2      25      57  C
                                STFCCA3      25      82  C
```

Figure 45. ISQL Command: RUN PSBSEGV (STBICAP 2 STSCCST)

```
11/14/88                              DL/I DATA BASE USAGE REPORT
PAGE 1

DBD.NAME  PSB.NAME  PCBNUMBER
--------  --------  ---------
STDCDBP   STBICAP           2
          STBICLD           2
```

Figure 46. ISQL Command: RUN WHEREDBD (STDCDBP)

```
11/14/88                              DL/I SEGMENT USAGE REPORT
PAGE 1
```

```
     SEGM.NAME  DBD/PSB NAME(S)
     ---------  ---------------
     STSCCST    STBCUSR
                STBCUSU
                STBICAP
                STBICLD
                STBICLG
                STDCDBL
                STDCDBP
                STDIDBL
```

---

Figure 47. ISQL Command: RUN WHERESEG (STSCCST)

---

```
     _____________________________________________________________________
     11/14/88                          DL/I FIELD USAGE REPORT
PAGE 1

     FIELD     SEGMENT(S)  DBD/PSB NAME(S)
     --------  ----------  ---------------
     STQCCNO   STSCCST     STDCDBP
```

---

Figure 48. ISQL Command: RUN WHEREFLD (STQCCNO)

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 5.0 Part 5. Defining your Data Sets to VSAM

This part explains how to define your data sets to VSAM. This means that space is allocated for the data set, the name is assigned, and other data set information is provided. You must do this before your data base is loaded with actual data base records.

Because certain values are obtained from your DBDGEN Assembly listing, this part assumes you have successfully completed DBD generation as described in Chapter 8, "Doing a DBD Generation."

Subtopics:



# 5.1 Chapter 12. Defining VSAM Data Sets

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

- *VSAM Catalog*: A master catalog must be defined first and then, optionally, any number of VSAM user catalogs. A VSAM catalog is a central information point for all VSAM data sets and the direct-access storage volumes on which they are stored. The VSAM catalog provides VSAM with the information to allocate space for data sets, verify authorization to gain access to them, compile usage statistics on them and relate relative byte addresses (RBAs) to physical locations.

- *VSAM Data Spaces*: This is DASD space assigned to VSAM, from which VSAM allocates space for VSAM data sets. A record of this data space is maintained in a VSAM catalog. VSAM does its own DASD space management (for example, allocating space for VSAM data sets). Each VSAM data space can occupy part or all of a DASD volume.

- *VSAM Data Sets*: When a VSAM data set is defined, it is allocated space in a VSAM data space. A record of the data set and the space that it occupies is maintained in a VSAM catalog. All VSAM data sets must be cataloged.

The sample application supplied with DL/I includes the Access Method Services job needed to define the VSAM data sets. It is assumed that you already have defined your VSAM catalog(s) and VSAM data spaces. That is, you have used the Access Method Services DEFINE command (DEFINE MASTERCATALOG, DEFINE USERCATALOG, DEFINE SPACE) to establish your VSAM system. This chapter covers the use of the Access Method Services DEFINE CLUSTER command. See *Using VSE/VSAM Commands and Macros* for additional information.

**Note:** 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 job streams (DLZSAMAC/DLZSAMJS). Two sample job streams are provided for online (DLZSAMJS) and ACCESS (DLZSAMAC). The ACCESS job stream illustrates the ACCESS statement, but does not support the online sample application.

**For more information...**

about sample application job streams, refer to *DL/I DOS/VS Guide for New Users*.

Subtopics:

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 5.1.1 A Sample Data Set Definition

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 job shown in Figure 49 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 job stream DLZSAMAC.

```
// 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))
   /*
   /&
```

Figure 49. Defining the Inventory, Customer, and Index Data Bases to VSAM

Subtopics:

- 5.1.1.1 Using the DBDGEN Output Listing

---



---

# 5.1.1.1 Using the DBDGEN Output Listing

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: (see also Figure 4 in topic 2.1.4)

```
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........................  3380
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........................  3380
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 will assure that you have defined your VSAM data sets correctly.

# 6.0 Part 6. Describing DL/I Tables for an Online System

This section explains how to define and generate two DL/I tables that are used when processing DL/I in an online environment.

The first table described is called an Application Control Table (ACT). You must create an ACT if you wish to process DL/I in an online environment. It is used to associate online application programs with one or more DL/I data bases.

The second table described is called a Storage Layout Control (SLC) table. It may be optionally defined for use in an online system. This table lets you specify the sequence in which DL/I modules are loaded from the 'DL/I prod lib' during DL/I initialization.

Chapter 13, "Defining Application and Storage Layout Control Tables" describes how to define these tables and Chapter 14, "Doing ACT and SLC Generations" describes how to generate them.

**Creating an ACT Interactively**

The ACT definition and generation procedures described here can also be done interactively on a 3270-type terminal by using the Interactive Macro Facility (IMF). See *DL/I DOS/VS Interactive Resource Definition and Utilities* for more information.

Subtopics:

# 6.1 Chapter 13. Defining Application and Storage Layout Control Tables

Subtopics:



# 6.1.1 Defining an Application Control Table

This section explains how to prepare DLZACT control statements to define an Application Control Table (ACT). After you complete your definition, the ACT must be generated as explained in Chapter 14, "Doing ACT and SLC Generations."

For brevity, the ACT definition and generation procedure is commonly called ACTGEN.

Subtopics:



# 6.1.1.1 Input Structure and Rules

Figure 50 illustrates the rules for structuring an input stream for ACT assembly. The figure shows the types of DLZACT control statements required, the number of each type that may be specified, and their specific order of sequence.

***To Define an Application Control Table . . .***

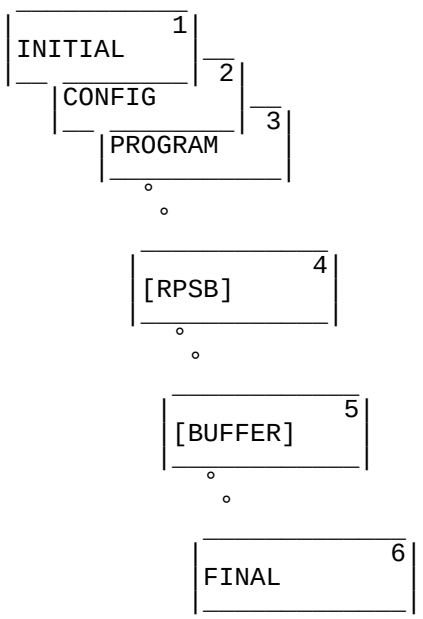



Figure 50. Batch Input Stream for ACTs

1. Code one INITIAL type statement. It must be the first statement in the ACT assembly.

2. Code one CONFIG type statement. Put it after the INITIAL statement.

3. Code one PROGRAM type statement for each application program that schedules a PSB. This includes mirror programs, such as DLZBPC00 (MPS batch mirror program) and DFHMIR (ISC mirror program). Place the PROGRAM statements after the CONFIG statement. A maximum of 4095 PROGRAM statements may be specified.

4. RPSB type statements are optional. Code one for each PSB located on a remote system. Put the RPSB statements after the last PROGRAM statement.

5. BUFFER type statements are optional. They can be used to specify information about DL/I buffer subpools and VSAM buffer usage. Put the

```
        BUFFER statements after the last RPSB statement.

    6.  Code one FINAL type statement to define the end of the ACT generation
        to the assembler.
```

---

---

# 6.1.2 Control Statements Format

This section explains how to code DLZACT control statements to define an ACT. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introduction.

Subtopics:



# 6.1.2.1 DLZACT TYPE=INITIAL

Table 80. DLZACT Statement Syntax for TYPE=INITIAL

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZACT | TYPE= | **I N I T I A L** | TYPE=INITIAL must be specified. |
| | ,SUFFIX= | _ _ | Optional 1- or 2-character alphameric suffix. |

*Coding Example*

```
DLZACT                                                  X
      TYPE=INITIAL,                                     X
      SUFFIX=DL
```

**TYPE=INITIAL**
Identifies this statement as the INITIAL type of DLZACT.

**SUFFIX=**
Specifies an optional 1- or 2-character alphameric suffix for the ACT assembly. The suffix, if specified, is appended to the standard module name (DLZNUC). This becomes the name of the module in the linkage editor output library. If this operand is omitted, a suffix is not provided. Do not use NO for a suffix.

# 6.1.2.2 DLZACT TYPE=CONFIG

Table 81. DLZACT Statement Syntax for TYPE=CONFIG

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DLZACT | TYPE= | **C O N F I G** | TYPE=CONFIG must be specified. |
| | ,MAXTASK= | _ _ _ | 1 to 255.(1) |
| | ,CMAXTSK= | _ _ _ | 1 to 255. Cannot exceed MAXTASK value.(2) |
| | ,BFRPOOL= | _ _ _ | 0 to 255.(3) |
| | ,PASS= | _ _ _ _ _ _ _ _ | Password; 1-8 alphameric characters.(4) |
| | ,SLC= | _ _ _ _ _ _ _ _ | Phase name of Storage Layout Control (SLC) table.(5) |
| | ,PI= | _ _ _ | YES - use program isolation.(6)<br>NO - do not use program isolation. |
| | ,REMOTE= | _ _ _ | YES - make PSBs accessible from remote system.(7)<br>NO - do not make PSBs accessible from remote system. |

**Notes:**

1. The MAXTASK operand is optional. If omitted, the default is 10.
2. The CMAXTSK operand is optional. If omitted, the default is the MAXTASK value.
3. The BFRPOOL operand is optional. If omitted (or if BFRPOOL=0 is specified), DL/I will request a value during system initialization.
4. The PASS operand is optional. If omitted, the default is DLZPASS1.
5. The SLC operand is optional. See operand description for details.
6. The PI operand is optional. If omitted, the default is YES.
7. The REMOTE operand is optional. If omitted, the default is NO.

*Coding Example*

```
DLZACT                                                  X
      TYPE=CONFIG,                                      X
      MAXTASK=10,                                       X
      CMAXTSK=5,                                        X
      PASS=PASSWORD,                                    X
      SLC=DLZSLC01,                                     X
```

```
                BFRPOOL=3,                                          X
                PI=YES,                                             X
                REMOTE=YES
```

**TYPE=CONFIG**
Identifies this statement as the CONFIG type of DLZACT.

**MAXTASK=**
Specifies the maximum number of DL/I tasks that may be processed concurrently. You may specify from 1 to 255 tasks. If you omit this operand, the default is 10.

**Note:** The DL/I MAXTASK parameter cannot be changed dynamically during CICS/DOS/VS execution as can the CICS/DOS/VS MXT and AMXT parameters. It can only be changed by assembling a new ACT. Since only dispatchable tasks can issue PCB calls, and the number of dispatchable tasks is limited by AMXT, there is no advantage in setting the DL/I MAXTASK parameter higher than the CICS/DOS/VS AMXT parameter. (Caution: this may possibly cause a deadlock.)

**CMAXTSK=**
Specifies the current maximum number of concurrent DL/I tasks allowed. You may specify from 1 to 255 tasks. Do not specify a number that exceeds your MAXTASK value. If this operand is omitted, the default is the number specified for MAXTASK.

The CMAXTSK parameter allows the maximum number of DL/I tasks within the system to be changed dynamically. A special DL/I system call (CMXT) provides the ability to adjust the current maximum task value.

Normally CMAXTSK should be set equal to MAXTASK. During periods of heavy demand on your CICS/DOS/VS system by non-DL/I tasks, you can lower the CMAXTSK value to restrict the number of concurrent DL/I tasks. This allows a greater proportion of non-DL/I tasks to concurrently execute in your CICS/DOS/VS system.

**BFRPOOL=**
Specifies the number of buffer subpools to be acquired and formatted during the initialization of the online DL/I system. You may specify a value from 0 to 255. If the value is omitted or zero, a request for the value will be 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 (CI) size, which may be either

- 512-8192 bytes in steps of 512 bytes, or
- 8192-30720 bytes in steps of 2048 bytes.

**Note:** Specification of CI sizes greater than 4K bytes is only possible for HD, HDAM, and HIDAM(ESDS) data base organizations.

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 size of the VSAM control intervals. A data base is assigned a subpool which contains buffers that are equal to or greater in size than the size of the data base control interval.

**PASS=**
Specifies the 1 to 8 alphameric character password associated with the special functions of DL/I system calls. If you omit this operand, the default is DLZPASS1. For more information about DL/I

system calls, see *DL/I DOS/VS Data Base Administration*.

**SLC=**
Specifies the phase name of the Storage Layout Control (SLC) table to be used during DL/I initialization. For more information about SLC tables see "Defining a Storage Layout Control Table" in this chapter.

**PI={YES}**
**{NO}**
Specify YES if you use program isolation. Specify NO if you do not. If you omit this operand, YES is the default.

**Note:** PI=NO must not be specified when an RPSB with local component has been specified in the LNAME operand of DLZACT TYPE=RPSB.

**REMOTE=**
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/DOS/VS mirror program DFHMIR. The REMOTE=YES parameter accomplishes this by generating a DLZACT TYPE=PROGRAM,PGMNAME=DFHMIR statement for the CICS/DOS/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.

---

# 6.1.2.3 DLZACT TYPE=PROGRAM

Table 82. DLZACT Statement Syntax for TYPE=PROGRAM

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZACT | TYPE= | **P R O G R A M** | TYPE=PROGRAM must be specified. |
| | ,PGMNAME= proc. | ° ° ° ° ° ° ° ° | Name of application program. |
| | ,PSBNAME= | (_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>,_ _ _ _ _ _ _ _<br>.<br>.<br>.<br>,_ _ _ _ _ _ _ _) | PSB name(s) associated with this application program. |
| | ,CONT= | Y E S | Specify YES to continue the list of PSB names. |

*Coding Example*

```
DLZACT                                                  *
      TYPE=PROGRAM,                                     *
      PGMNAME=ONLINE1B,                                 *
      PSBNAME=(PSBACT2,PSBACT1)
```

**TYPE=PROGRAM**
Identifies this statement as the PROGRAM type of DLZACT.

**PGMNAME=**
Specifies the name of the application program that is authorized to schedule a DL/I PSB.

**PSBNAME=**
Specifies the PSB names associated with this application program.

The PSBNAME name list (name,name,...) can have a maximum of 255 characters, including the parentheses (a VSE Assembler restriction). If more names are required, the name list can be continued on subsequent statements as shown below.

```
DLZACT    TYPE=PROGRAM,
          PGMNAME=name,
          PSBNAME=(name,name,...),CONT=YES
DLZACT    PSBNAME=(name,name,...),CONT=YES
  .
  .
  .
DLZACT    PSBNAME=(name,name,...)
```

**CONT=YES**
Indicates that this list of PSB names continues on the next line with another DLZACT statement.

Any keywords other than PSBNAME and CONT are ignored in PSBNAME continuation statements. No other types of statements may be within a string of PSBNAME continuation statements.

*For MPS batch,* a DLZACT TYPE=PROGRAM statement naming the MPS batch partition controller program (PGMNAME=DLZBPC00) must be included in your ACT definition. Because this program issues DL/I calls on behalf of the MPS batch partition, all PSB names that can be referenced by the MPS batch program must also be included in the statement's PSBNAME operand.

*For Intersystem Communication,* a DLZACT TYPE=PROGRAM statement naming the CICS/DOS/VS mirror program (PGMNAME=DFHMIR) must be included in your ACT definition. Because this program issues DL/I calls on behalf of another (remote) system, all PSB names that can be referenced by a remote system must be specified in the statement's PSBNAME operand (see DLZACT TYPE=CONFIG REMOTE operand).

---

---

# 6.1.2.4 DLZACT TYPE=RPSB

Table 83. DLZACT Statement Syntax for TYPE=RPSB

| Statement | Keyword | Parameter | Description (See also Notes below) |
|---|---|---|---|
| DLZACT | TYPE= | **R P S B** | TYPE=RPSB must be specified. |
| | ,PSB= proc. | ° ° ° ° ° ° ° | PSB name. |
| | ,LNAME= | _ _ _ _ _ _ _ | Name of PSB in local system.(1) |
| | ,RNAME= | _ _ _ _ _ _ _ _ | Name of PSB in remote system.(2) |
| | ,SYSID= proc. | ° ° ° ° | CICS/DOS/VS system identifier of remote PSB and associated data bases. |
| | ,LANG= | P L / I | Specify PL/I if all MPS batch programs using the DL/I call interface to access this RPSB are written in PL/I. |

**Notes:**

1. The LNAME operand is used for "extended RPSB support." See operand description for details.
2. The RNAME operand is required for "extended RPSB support." See operand description for details.

*Coding Example*

```
DLZACT                                                  *
      TYPE=RPSB,                                        *
      PSB=PSBACT2,                                      *
      LNAME=LOCPSB,                                     *
      RNAME=REMPSB,                                     *
      SYSID=SYS2
```

**TYPE=RPSB**
Identifies this statement as the RPSB type of DLZACT.

**PSB=**
Specifies a unique name for this PSB. This is the PSB name the application program will use to access remote data. The PSB name specified here must also be one of the names previously defined in the PSBNAME list of a PROGRAM type DLZACT statement in this ACT assembly.

**LNAME=**
Specifies the name of the local PSB component.

This operand is optional. If specified, this local PSB is concatenated with the remote PSB. This produces a single RPSB with a local PSB component. Application programs using this RPSB would then be able to access DL/I data bases located on both systems (that is, the local system and the remote system) simultaneously. This capability is known as "extended RPSB support."

If you do not want the RPSB defined with a local PSB component, skip this operand.

**RNAME=**
Specifies the name of the PSB in the remote system.

This is a required operand if you are defining the RPSB with a local PSB component (and are using the LNAME operand to name the local PSB component).

If, however, you are defining the RPSB without a local PSB component (LNAME was not specified), then this operand is optional. If you omit it, the PSB name specified in the PSB operand of this statement is used as a default.

**Notes:**

1. If the remote system is CICS/DOS/VS with DL/I, the RNAME parameter must be 1-7 characters long. If the remote system is CICS/OS/VS with IMS, however, the RNAME parameter can be 1-8 characters long.

2. The order of PCBs in an extended RPSB is determined by the local and remote PSB names. The PCBs in the PSB whose name comes first in standard EBCDIC collating sequence will be presented first in the PCB address list.

3. If LNAME is specified, the RNAME operand must not be the same as the PSB operand.

**SYSID=**
Specifies the 4-character identifier of the CICS/DOS/VS system to which this remote PSB and associated data bases are assigned. This identifier corresponds to the SYSIDNT operand of the CICS/DOS/VS DFHTCT TYPE=ISLINK statement.

**Note:** The remote PSB and its related DMB(s) and data base(s) must all reside on the same system.

**LANG=**
Specifies that all MPS batch application programs which use the DL/I call interface to access this RPSB will be written in PL/I. "PL/I" may be specified as PL/I, PL/1, PLI, or PL1.

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 6.1.2.5 DLZACT TYPE=BUFFER

Table 84. DLZACT Statement Syntax for TYPE=BUFFER

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZACT | TYPE= | **B U F F E R** | TYPE=BUFFER must be specified. |
| | ,HDBFR= | (_ _<br><br>,_ _ _ _ _ _ _<br>,_ _ _ _ _ _ _<br>.<br>.<br>.<br>,_ _ _ _ _ _ _) | Number of buffers (from 2 to 32) to be allocated to this subpool. If omitted, 32 is assumed.<br><br>Name(s) of DBDs to be allocated to this subpool. |
| | ,HSBFR= | (_ _ | Number of index buffers for the KSDS. If omitted, 3 is assumed. |
| | | ,_ _ | Number of data buffers for the KSDS. If omitted, 2 is assumed. |
| | | ,_ _ | Number of data buffers for the ESDS (HISAM only). If omitted, 2 is assumed. |
| | | ,&box.&box.&box.&box.&bo | .DBDxnamexof the SHISAM, HISAM, or INDEX DBD. |

*Coding Example*

```
DLZACT                                                  *
      TYPE=BUFFER,                                      *
      HDBFR=(30,HDAM1,DHIDAM2,DINDEX2),                 *
      HSBFR=(3,2,,INDEX)
```

**TYPE=BUFFER**
Identifies this statement as the BUFFER type of DLZACT.

**HDBFR=**
Describes one DL/I subpool as follows:

*The first parameter* for HDBFR 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.

*The second parameter* for HDBFR specifies the name(s) of DBDs that are

to be allocated to this subpool. If no DBD names 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, because a logical DBD has one or more physical DBDs, all physical DBDs should be specified that are to be allocated to a specific subpool. Do not specify an INDEX or HISAM DBD name in an HDBFR statement unless there is a PSB with load or insert sensitivity for the DBD. Use HSBFR instead, for these DBDs.

**HSBFR=**

Defines VSAM buffer allocation for SHISAM, HISAM, and INDEX data bases as follows:

*The first parameter* for HSBFR specifies the number of index buffers for a KSDS. If omitted, 3 is assumed.

*The second parameter* of HSBFR specifies the number of data buffers for a KSDS. If omitted, 2 is assumed.

*The third parameter* of HSBFR applies to HISAM data bases only. It specifies the number of data buffers for the ESDS. If omitted 2 is assumed.

*The last parameter* of HSBFR specifies the DBD name of the SHISAM, HISAM, or INDEX DBD referenced by the application program.

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 6.1.2.6 DLZACT TYPE=FINAL

Table 85. DLZACT Statement Syntax for TYPE=FINAL

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZACT | TYPE= | **F I N A L** | TYPE=FINAL must be specified. |

*Coding Example*

```
DLZACT
      TYPE=FINAL
```

**TYPE=FINAL**
Identifies this statement as the FINAL type of DLZACT.

 *© Copyright IBM Corp. 1981, 1989*

# 6.1.3 Control Statements Summary

Table 86 summarizes the DLZACT control statements used to define an ACT. For an explanation about the conventions used to prepare this illustration, see "Coding Conventions" in the Introduction.

Table 86. Summary of Control Statements Used to Code ACTs

| [Label] | Statement | Operand |
|---|---|---|
| | DLZACT | TYPE=INITIAL<br>[,SUFFIX=xx] |
| | DLZACT | TYPE=CONFIG<br>[,MAXTASK={*10* }]<br>{nnn}<br>[,CMAXTSK={*maxtaskvalue*}]<br>{nnn }<br>[,BFRPOOL=nnn]<br>[,PASS={*DLZPASS1*}]<br>{password}<br>[,SLC=phasename]<br>[,PI={*YES*}]<br>{NO }<br>[,REMOTE={*NO* }]<br>{YES} |
| | DLZACT | TYPE=PROGRAM<br>,PGMNAME=name<br>,PSBNAME=(name,name,...)[,CONT=YES] |
| | DLZACT | TYPE=RPSB<br>,PSB=psbname<br>[,LNAME=localname]<br>[,RNAME=remotename]<br>,SYSID=systemid<br>[,LANG=PL/I] |
| | DLZACT | TYPE=BUFFER<br>{,HDBFR=(bufno[,dbdname1,dbdname2,...]) }<br>{,HSBFR=(indno,ksdsbuf[,esdsbuf],dbdname)} |
| | DLZACT | TYPE=FINAL |

Subtopics:

- 6.1.3.1 Example of an ACT Definition

# 6.1.3.1 Example of an ACT Definition

```
1.        DLZACT                                                *
                TYPE=INITIAL,                                   *
                SUFFIX=DL
2.        DLZACT                                                *
                TYPE=CONFIG,                                    *
                MAXTASK=10,                                     *
                CMAXTSK=5,                                      *
                PASS=PASSWORD,                                  *
                SLC=DLZSLC01,                                   *
                BFRPOOL=3,                                      *
                PI=YES,                                         *
                REMOTE=YES
3.        DLZACT                                                *
                TYPE=PROGRAM,                                   *
                PGMNAME=ONLINE1A,                               *
                PSBNAME=PSBACT1
4.        DLZACT                                                *
                TYPE=PROGRAM,                                   *
                PGMNAME=ONLINE1B,                               *
                PSBNAME=(PSBACT2,PSBACT1)
5.        DLZACT                                                *
                TYPE=RPSB,                                      *
                PSB=PSBACT2,                                    *
                LNAME=LOCPSB,                                   *
                RNAME=REMPSB,                                   *
                SYSID=SYS2
6.        DLZACT                                                *
                TYPE=BUFFER,                                    *
                HDBFR=(3,DHIDAM1),                              *
                HSBFR=(3,2,,INDEX)
          DLZACT                                                *
                TYPE=BUFFER,                                    *
                HDBFR=(30,DHDAM1,DHIDAM2,DINDEX2),              *
                HSBFR=(3,2,,INDEX)
7.        DLZACT                                                *
                TYPE=FINAL
```

Figure 51. Example of an ACT Definition

**Notes:**

1. The generated CSECT name is DLZNUCDL.
2. The absolute maximum number of tasks permitted to process concurrently is 10. The maximum number of concurrent tasks after startup is 5. Three buffer subpools will be acquired and initialized during DL/I initialization. The password required to access DL/I system calls (CMXT,STRT,STOP) is 'PASSWORD'. The storage layout control table with the phase name DLZSLC01 will be used. Program isolation will be used. All PSBs defined in this nucleus can be scheduled on behalf of the DL/I application program running in a remote DL/I system.
3. The "ONLINE1A" program may process with PSB "PSBACT1."
4. The program "ONLINE1B" may process with either PSB "PSBACT1" or "PSBACT2."
5. PSBACT2 and its DMBs are located on SYS2, a different system than programs ONLINE1A and ONLINE1B are run on. The name of PSBACT2 is REMACT2 on the remote system.

```
6.  The first DL/I subpool contains three buffers and is assigned to the
    data base DHIDAM1.  The second subpool has 30 buffers and is for data
    bases DHDAM1, DHIDAM2, and work areas for DINDEX2.  The third subpool,
    since it is not explicitly described, has 32 buffers and is used for
    all remaining data bases (including work areas for all other indexes).
7.  Required to begin the generation of the nucleus.
```

---



---

# 6.1.4 Defining a Storage Layout Control Table

This section explains how to prepare DLZSLC control statements to define a Storage Layout Control (SLC) table. After you complete your definition, the SLC table must be generated as explained in Chapter 14, "Doing ACT and SLC Generations."

Subtopics:



# 6.1.4.1 Input Structure and Rules

Figure 52 illustrates the rules for structuring an input stream for SLC assembly. The figure shows the types of DLZSLC control statements required, the number of each type that may be specified, and their specific order of sequence.

***To Define a Storage Layout Control Table . . .***

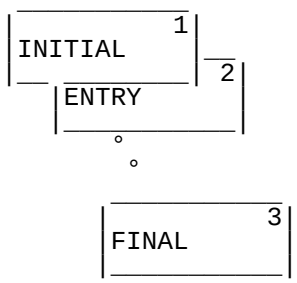



Figure 52. Batch Input Stream for SLCs

1. Code one INITIAL type statement. It must be the first statement in the SLC assembly.

2. Code one ENTRY type statement for each action module. Code one for the buffer pool also. Keep the statements in the same sequence in which the modules and buffer pool are to be loaded during DL/I initialization. A maximum of 12 ENTRY type DLZSLC statements may be defined. Put them after the INITIAL type statement.

3. Code one FINAL type statement. Put it after the last ENTRY statement.

# 6.1.5 Control Statements Format

This section explains how to code DLZSLC control statements to define SLC tables. For an explanation about the conventions used to describe these statements, see "Coding Conventions" in the Introducton.

Subtopics:



# 6.1.5.1 DLZSLC TYPE=INITIAL

Table 87. DLZSLC Statement Syntax for TYPE=INITIAL

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZSLC | TYPE= | **I N I T I A L** | TYPE=INITIAL must be specified. |
| | ,NAME= | _ _ _ _ _ _ _ _ | Phase name of this SLC table. If omitted, the default is DLZSLC. |

**TYPE=INITIAL**
Identifies this statement as the INITIAL type of DLZSLC.

**NAME=**
Specifies the phase name of this SLC table. If omitted, the default name is DLZSLC. The SLC table name specified here must also be specified in the SLC= operand of a DLZACT TYPE=CONFIG statement. See "Defining an Application Control Table" in this chapter.

 *© Copyright IBM Corp. 1981, 1989*

# 6.1.5.2 DLZSLC TYPE=ENTRY

Table 88. DLZSLC Statement Syntax for TYPE=ENTRY

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZSLC | TYPE= | **E N T R Y** | TYPE=ENTRY must be specified. |
| | ,MODULE= proc. | ° ° ° ° ° ° ° ° | Program name of action module. See operand description for list of valid names. |
| | ,LOAD= | _ _ _ _ | LOW or HIGH. |
| | ,ALIGN= | _ _ _ | NO or YES. |

**TYPE=ENTRY**
Identifies this statement as an ENTRY type of DLZSLC.

**MODULE=**
Identifies the program name of the action module/buffer. The following names may be specified:

| Program Name | Title |
|---|---|
| DLZDBH00 | Buffer Handler |
| DLZDLR00 | Retrieve |
| DLZDLA00 | Call Analyzer |
| DLZDLD00 | Delete/Replace |
| DLZDDLE0 | Load/Insert |
| DLZDHDS0 | Space Management |
| DLZDXMT0 | Index Maintenance |
| DLZDLOC0 | Open/Close |
| DLZRDBL0 | Data Base Logger |
| DLZQUEF0 | PI Queueing Facility |
| DLZQUEFW | PI Queueing Facility Work Area |
| DLZCPY10 | FLS Copy |
| BUFFER | HD Buffer Pool |

Do not specify any name more than once. Those modules in this list that are not specified in your SLC table will be loaded according to the default sequence described next.

**MODULE=ALL**
may be specified if all action modules are to be loaded with the same LOAD and ALIGN options and in the same sequence provided as a default. The default loading sequence into low storage is as follows:

DLZDBH00
DLZDLR00
DLZDLA00
DLZRDBL0
DLZDLD00
DLZDDLE0

```
DLZDHDS0
DLZDXMT0
DLZDLOC0
DLZQUEF0
DLZQUEFW
DLZCPY10
```

If MODULE=ALL is specified, do not use additional ENTRY type statements to specify any module names. You may, however, still specify MODULE=BUFFER. MODULE=BUFFER is coded for the HD buffer pool.

**LOAD=**

Defines whether the module/buffer is to be loaded into low or high storage. Frequently used modules and the HD buffer pool should be placed in low address space. The least referenced modules should be placed in high address space.

**ALIGN=**

Defines whether the module is to be loaded on a page boundary or not. Not on a page boundary is the default.

**Note:** Always specify ALIGN=YES for MODULE=BUFFER.

---

# 6.1.5.3 DLZSLC TYPE=FINAL

Table 89. DLZSLC Statement Syntax for TYPE=FINAL

| Statement | Keyword | Parameter | Description |
|---|---|---|---|
| DLZSLC | TYPE= | **F I N A L** | TYPE=FINAL must be specified. |

**TYPE=FINAL**
Identifies this statement as a FINAL type of DLZSLC.

# 6.1.6 Control Statements Summary

Table 90 summarizes the DLZSLC control statements used to define a SLC table.

Table 90. Summary of Control Statements Used to Code SLCs

| [Label] | Statement | Operand |
|---|---|---|
| | DLZSLC | TYPE=INITIAL<br>[,NAME={*DLZSLC*  }]<br>{phasename} |
| | DLZSLC | TYPE=ENTRY<br>,MODULE={name}<br>{ALL }<br>,LOAD={*LOW* }<br>{HIGH}<br>,ALIGN={*NO* }<br>{YES} |
| | DLZSLC | TYPE=FINAL |

# 6.2 Chapter 14. Doing ACT and SLC Generations

This chapter assumes you have completed the definition of an ACT and/or SLC table, as explained in Chapter 13, "Defining Application and Storage Layout Control Tables," and are now ready to do the generation. Generations may be invoked any time after DL/I has been installed in your system. The generation requirements for ACT and SLC are described separately in the following text.

Subtopics:



# 6.2.1 Doing the ACT Generation

Subtopics:



## 6.2.1.1 Preparing the Job Control Statements

ACTGEN is run as a standard system job. Note that it has two steps. Step 1 is a macro assembly execution. This step reads and expands your DLZACT input statements and produces an object module. The DL/I macros used for step 1 reside in the 'DL/I prod lib'. In step 2, the generated online nucleus is cataloged and link-edited as a load module in the 'DL/I user lib'.

The job control statements used to invoke the ACTGEN procedure are shown below. The placement of your DLZACT input statements within the job control statements is also shown.

```
// JOB DLZNUC
// OPTION CATAL
// LIBDEF *,SEARCH='DL/I prod lib'
// LIBDEF PHASE,CATALOG='DL/I user lib'
* STEP 1
// EXEC ASSEMBLY
       DLZACT TYPE=INITIAL,SUFFIX=... |
       DLZACT TYPE=CONFIG,...         |
       DLZACT TYPE=PROGRAM,...        | Put your
       DLZACT TYPE=RPSB,...           | DLZACT statements
       DLZACT TYPE=BUFFER,...         | here
       DLZACT TYPE=FINAL              |
       END
/*
* STEP 2
       ENTRY DLZNUC
// EXEC LNKEDT
/*
/&
```

**Notes:**

- An Assembler END statement is required at the end of your DLZACT input stream.

- Note the linkage editor ENTRY statement identifying DLZNUC as the entry point.

 *© Copyright IBM Corp. 1981, 1989*

## 6.2.1.2 Analyzing the Output

A standard assembly listing is created each time the ACT generation procedure is executed.

Any errors found, such as invalid parameters or the omission or invalid sequence of input statements will cause the ACT generation procedure to terminate prematurely and diagnostic messages to be printed.

You must then correct the errors and rerun the job. The ACTGEN must be error free.

Diagnostic messages used to identify ACTGEN error conditions are described in the *DL/I Messages and Codes* manual.

 *© Copyright IBM Corp. 1981, 1989*

# 6.2.2 Doing the SLC Generation

SLC table generation is executed as a standard system job. The procedure is similar to that described for ACTGEN. Two job steps are used. Step 1 is a DLZSLC macro assembly execution. It reads and expands your input statements and produces an object module. Diagnostic messages are printed if any errors are detected during assembly. Step 2 uses the object module to produce the SLC table which is cataloged and link-edited as a load module into the 'DL/I user lib'.

The job control statements used to invoke SLC table generation are shown below. The placement of your DLZSLC control statements within the job control statements is also shown.

```
// JOB SLCGEN
// OPTION CATAL
// LIBDEF *,SEARCH='DL/I prod lib'
// LIBDEF PHASE,CATALOG='DL/I user lib'
* STEP 1
// EXEC ASSEMBLY
      DLZSLC TYPE=INITIAL
      DLZSLC TYPE=ENTRY

            .     |
            .     |   Put your DLZSLC
            .     |   statement here.

      DLZSLC TYPE=FINAL
      END
/*
* STEP 2
// EXEC LNKEDT
/&
```

**Note:** An Assembler END statement is required at the end of your DLZSLC input stream.

 *© Copyright IBM Corp. 1981, 1989*

# 7.0 Part 7. Running DL/I Programs in Your System

This part explains how to execute DL/I application and utility programs in your system. Separate requirements are given for batch, online, and multiple partition support (MPS) environment operation.

**Generating DL/I Utility Job Streams Interactively**

The DL/I utility job streams described here can also be generated interactively on a 3270-type terminal using the Interactive Utility Generation (IUG) Facility. See *DL/I DOS/VS Interactive Resource Definition and Utilities* for more information.

Subtopics:

- 7.1 Chapter 15. Executing DL/I Programs

# 7.1 Chapter 15. Executing DL/I Programs

This chapter describes job control statements and DL/I parameter information used to execute DL/I application and utility programs. Separate requirements are given for batch, online, and multiple partition support (MPS) environment operation.

Subtopics:



# 7.1.1 Batch Requirements

Some DL/I utility programs execute as standard VSE application programs. These stand-alone programs are obtained from a 'DL/I prod lib' for execution by specifying the program name in a // EXEC statement of a job control stream. DL/I utility programs in this category include:

- HISAM Reorganization Unload
- HISAM Reorganization Reload
- Data Base Prefix Resolution Utility
- Part 1 of Partial Data Base Reorganization
- Data Base Change Accumulation
- Data Base Data Set Image Copy
- Log Print
- Trace Print.

For all other DL/I utilities (except the ISQL Extract Define Utility which runs with SQL/DS) and for all user-written application programs, the DL/I initialization module (DLZRRC00) is the program name specified in the // EXEC statement for execution by VSE. The actual user-written program name, or the utility name, is then specified in a parameter statement that is passed to DL/I on SYSIPT or directly from SYSLOG. User requirements for this parameter statement are described later in this chapter. The DL/I utilities in this program category include:

- HD Reorganization Unload
- HD Reorganization Reload
- Data Base Prereorganization
- Data Base Scan
- Data Base Prefix Update
- Part 2 of Partial Data Base Reorganization
- Data Base Data Set Recovery

- Data Base Backout.

Job streams for all DL/I utilities, whether they run under the DL/I initialization module or are stand-alone programs, may be generated interactively through the Interactive Utility Generation (IUG) facility. For more information on generating utility job streams interactively, see *DL/I DOS/VS Interactive Resource Definition*.

Subtopics:



---

# 7.1.1.1 DL/I Initialization Job Control Statement Requirements

The job control statements required for DL/I utility program execution are shown in Part 9, "Data Base Reorganization Utilities" and Part 10, "Data Base Recovery Utilities" in this manual.

The job control statements required for user-written batch application program are shown below.

| | | |
|---|---|---|
| // UPSI | x0000xxx | The x character represents the DL/1 functions you select. See "UPSI Byte Settings for Batch DL/1" described later in this section for the meaning of the bit settings. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(Disk)<br>[// DLBL | SYS011,cuu]<br>LOGOUT<br><br>DSKLOG1]<br>DSKLOG2] | These statements define the output log file(s). Their use is optional. If no output log file is desired, these statements may be omitted. Bit 6 of UPSI must then be set to 1 and the LOG parameter cannot be specified in the DL/I parameter statement.<br><br>An output log file can be tape or disk.<br><br>For a tape log file, the symbolic file name is LOGOUT.<br><br>For disk log files, the symbolic filenames are DSKLOG1 and DSKLOG2. These DSKLOG files must be VSAM data sets. If one disk file is used, it must be DSKLOG1. If two disk files are used, separate DLBL statements must be used for DSKLOG1 and DSKLOG2. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(Disk) | SYSnnn,cuu<br>filename<br><br>filename | These statements define a data base file and must be present for each file referenced by every data base referenced by the application program's PSB<br><br>The filenames used in these statements must be the same as the filenames specified for the DD1, DD2, or OVFLW parameter of DATASET statements, or the REF parameter of ACCESS statements, in the data base DBD. |
| // EXEC | DLZRRC00,<br>SIZE=xxxK | This statement identifies the program name of the DL/I Batch Initialization Module and causes it to execute. Refer to *DL/I DOS/VS Data Base Administration* for storage size requirements. |
| /*<br>// | | |

```
|__________________|__________________|_____________________________________|
```

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Subtopics:

- 7.1.1.1.1 UPSI Byte Settings for Batch DL/I

---

 *© Copyright IBM Corp. 1981, 1989*

---

### 7.1.1.1.1 UPSI Byte Settings for Batch DL/I

Several functions can be controlled by the UPSI byte setup. Through its use, DL/I initialization (DLZRRC00) determines whether DL/I will log modifications to the data base, receive control if a program terminates abnormally, and whether DL/I parameter information should be read from the SYSIPT or SYSLOG device.

The UPSI byte settings for batch DL/I are shown here. Be aware that UPSI byte settings in the online or MPS environment have different meanings than those shown here for batch. They are described elsewhere in this chapter.

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 program.

Bit 5 = 0   Storage dump on set exit (STXIT) abnormal task termination.
      = 1   No storage dump on set exit STXIT abnormal task termination.

Bit 6 = 0   All data base modifications written to the DL/I system log.
      = 1   Dl/I system log function inactive.

Bit 7 = 0   Set exit (STXIT) linkage to DL/I for abnormal task termination.
      = 1   STXIT inactive.
```

## 7.1.1.2 DL/I Parameter Information Requirements

DL/I application programs 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, RC, TRACE, ASLOG, and LOG. For the complete format of these optional parameters, see the statement format used by "DL/I Application Programs."

*Conditional JCL*: No special action is required to invoke conditional JCL support for utilities. DL/I will automatically determine whether the program running is a DL/I utility or a user application program.

If it is a DL/I utility, conditional JCL support will automatically be activated. If it is a user application program, conditional JCL support must be explicitly requested with RC=YES.

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
           [,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
             {1  }
```

° **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}]
{DLR}                  {1  }
{PLS}
{PLC}
  [,HDBFR=({bufno}[,dbdname1,dbdname2,...])][,...]
           {32   }
  [,HSBFR=({indno},{ksdsbuf},[{esdsbuf}],dbdname3)][,...]
           {3    } {2      }  {2      }
  [,RC={NO}]
       {YES}
  [,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 continued in a continuation statement.

**DLI**
Function code required for batch DL/I programs or MPS batch DL/I programs that do not use the MPS restart facility.

**DLR**
Function code for MPS batch DL/I programs using the MPS restart facility. If DLR is specified as for a non-MPS batch DL/I program, it is treated the same as DL/I.

**Note:** The only parameters recognized for MPS batch DL/I programs (with DLI/DLR function codes) are progname, psbname, and RC=.

**PLS/PLC**
Function codes for batch DL/I programs which use the Partial Data

Base Load facility:

**PLS** (Partial Load Step) is similar to DL/I, but must be specified within Partial Data Base Load for the first and following steps - except the last one.
**PLC** (Partial Load Complete) is similar to DL/I, but must be specified within Partial Data Base Load only for the last step.

**For more information...**

see Chapter 17, "Partial Data Base Load" in topic 8.2.

**progname**
Specifies a one to eight alphameric character name of the application program or utility to be executed.

**dbdname**
Specifies a one to seven alphameric character name of the DBD for the data base to be accessed by the utility program, if required. This parameter is not required for the utilities:

DLZURPR0 - Data base preorganization
DLZURGS0 - Data base scan
DLZURGP0 - Data base prefix update.

**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 from 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 (CI) size, which may be either

- 512-8192 bytes in steps of 512 bytes, or
- 8192-30720 bytes in steps of 2048 bytes.

**Note:** Specification of CI sizes greater than 4K bytes is possible only for HD, HDAM, and HIDAM(ESDS) data base organizations.

The buffer size value is determined at DL/I system initialization and is based on the value specified in **buf**, the number of data bases, and size of the VSAM control intervals. A data base is assigned a subpool which contains buffers that are equal to or greater in size than the size of the data base control interval.

**For More Information...**

about DL/I buffer pool, see "Buffer Pool Assignment" in *DL/I DOS/VS Data Base Administration*.

**HDBFR**
Describes one DL/I subpool.

- 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,... specify 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, because a logical DBD has one or more physical DBDs, all physical DBDs should be specified that are to be allocated to a specific subpool. Do not specify an INDEX or HISAM DBD name in an HDBFR statement unless the DBD has load or insert sensitivity. Use HSBFR instead, for these DBDs.

**HSBFR**

Defines VSAM buffer allocation for HISAM, SHISAM, and INDEX data bases.

- indno specifies the number of index buffers for a KSDS; if omitted, 3 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.

- ksdsbuf specifies the number of data buffers for a KSDS; if omitted, 2 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.

- esdsbuf specifies the number of data buffers for the ESDS (applies to HISAM only); if omitted, 2 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.

- dbdname3 is the name of the HISAM, SHISAM, or INDEX DBD referenced by the application program.

**TRACE**

Indicates that tracing is to be active during this execution. See the *DL/I DOS/VS Diagnostic Guide* for details on tracing.

**ASLOG=YES**

Specifies that asynchronous logging is to be used.

Due to the "write ahead" function of logging, the speed of the log device may become the critical factor, so delaying system execution. The user may, therefore, decide not to use synchronous logging. He may do so by specifying ASLOG=YES in the DL/I parameter statement (batch jobs only). He must, however, be aware that in this case logging information may get lost if there is a power failure or unsuccessful STXIT processing. Data base backout may then not be possible, depending on the type of processing that was being done by the abnormally terminating program.

**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 disks extents. If one disk extent becomes full, the extent is closed and the other extent is used. DSKLOG1 is used first, then DSKLOG2. If DSKLOG2 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 DL/I logging is required. If anything other than the above parameters is specified, an error message is issued and the job is cancelled.

In the online and/or MPS environments, DL/I disk logging is not supported. If program isolation is active, the user must select CICS/DOS/VS journalling services for writing log information.

**RC=**

**NO** Suppress conditional JCL support (the default).
**YES** Invoke conditional JCL support.

If an invalid request for RC= is coded, message DLZ111I will be issued and RC=NO will be assumed.

See "Return Code Information" below.

Subtopics:



---

---

## 7.1.1.2.1 Return Code Information

*Return codes provided by DL/I*: The DL/I initialization programs (DLZRRC00 and DLZMPI00) and the DL/I utilities will always provide return codes. No special user action is necessary. For the various error severities the following return codes are assigned:

| RC | Severity | Description |
|---|---|---|
| 0 | Information message | Successful completion. |
| 4 | Warning message | A problem was encountered; all functions were completed but minor specifics were bypassed. |
| 8 | Warning message | The functions were completed, but major specifics were bypassed. |
| 12 | Error message | Not all requested functions were completed. |

Due to initial settings of VSE JCL, return codes greater than or equal to 16 are not provided, to prevent the job streams from canceling. On the other hand, there will still be error situations that cause DL/I or its utilities to cancel.

*Return codes issued by user application programs*: The DL/I initialization programs DLZRRC00 and DLZMPI00 pass a return code provided by user application programs back to VSE/SP. This must be requested with RC=YES.

If conditional JCL support is requested for application programs (RC=YES), DL/I interprets the contents of register 15 as return code at the application program's return point to DL/I.

For the various programming languages supported by DL/I, return codes can be set as follows:

**PL/I** PL/I provides the built-in function PLIRETC, which can be used to issue return codes.

**COBOL** COBOL only supports conditional JCL for MVS, using the predefined register RETURN-CODE. There is no possibility to pass return codes to VSE; this applies also to DL/I. However, it is possible to write ASSEMBLER subroutines for COBOL which handle return code setting.

**RPGII** Same as for COBOL

**ASSEMBLER** Return codes can be passed to DL/I via register 15 at return point.

*Special Considerations*

- Multiple errors

  In case of multiple error situations, DL/I always passes the *highest* return code back to VSE.

- Status codes

  *No* return codes are associated with the various DL/I status codes. DL/I utilities will handle those status codes, but for user application programs it is the user's responsibility to react to the status codes.

- MPS considerations

  For performance considerations, no return codes are passed from the online to the batch partition. To react, for example, on completion of online Dynamic Transaction Backout (DTB), the batch partition(s) would have to "long" wait. This conflicts with the intention of optimizing performance. Usually, this restriction only affects severe error situations, as the online task and the batch partition are terminating asynchronously.

---

# 7.1.2 Online Requirements

DL/I online processing permits the scheduling of multiple application programs to DL/I and allows access by more than one user to the same data base. CICS/DOS/VS is specially adapted to interface with the DL/I online processor. By combining the CICS/DOS/VS and DL/I system capabilities, you may expand DL/I applications from the batch processing environment to the CICS/DOS/VS teleprocessing environment.

DL/I online initializing consists of a CICS/DOS/VS initialization overlay phase called by the CICS/DOS/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, the user may elect to make use of the combined CICS/DOS/VS-DL/I journal facility.

When using this feature the DL/I log records are written directly to the CICS/DOS/VS system journal file by CICS/DOS/VS journal requests. If the DL/I logger is used, SYS011 must be assigned to the output log file device.

Subtopics:

- 7.1.2.1 UPSI Byte Settings for Online DL/I

 *© Copyright IBM Corp. 1981, 1989*

## 7.1.2.1 UPSI Byte Settings for Online DL/I

```
Bits 0 - 2   Reserved for CICS/DOS/VS.

Bits 3 - 5   Available subject to information described
             in CICS/DOS/VS Installation and Operations Guide, SC33-0070.

Bit 6 = 0    DL/I log function active.
      = 1    DL/I log function inactive.

Bit 7 = 0    DL/I system log function on CICS/DOS/VS system log.
      = 1    DL/I system log function on DL/I system log device
             (SYS011-LOGOUT).
```

# 7.1.3 MPS Requirements

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/DOS/VS parition 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/DOS/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 catagory are:

- Utilities
- Programs loading a data base
- Programs using SHSAM or HSAM

In addition, any batch DL/I programs that modify the contents of the DL/I control blocks cannot run under MPS because the DL/I control blocks no longer exist in the batch partition.

Subtopics:



# 7.1.3.1 MPS Initialization Job Control Language Requirements

The job control statements required for MPS batch application program execution are shown below. Note that no ASSGN, DLBL, EXTENT, 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 statements for the CICS/DOS/VS partition.

| | | |
|---|---|---|
| // UPSI | x0000x00 | The x character represents the DL/I functions you select. See "UPSI Byte Setting for MPS" for the meaning of the bit settings. |
| // ASSGN<br>[// TLBL<br>or<br>[// DLBL<br>[// EXTENT | SYSnnn,cuu<br>filename ]<br><br>filename ]<br>extent data] | If the MPS Restart facility is used, these statements must be included to define the VSE CHKPT file. It may be either DASD or tape with standard labels. |
| // EXEC | DLZMPI00,<br>SIZE=xxxK | This statement identifies the program name of the MPS Initialization Module and causes it to execute. Refer to *DL/I DOS/VS Data Base Administration* for storage size requirements. Do not specify SIZE=AUTO. |
| // RSTRT | SYSnnn,chkptid | If an MPS batch job is to be restarted using the MPS Restart facility, the VSE RSTRT statement must be used instead of the EXEC statement when the job is resubmitted for execution. The checkpoint ID (CHKPTID) specified on this statement is obtained from messages issued by DL/I either at the time of failure or at system bring-up time. |

**Notes:**

1. The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.
2. If the checkpoint ID specified on the RSTRT statement is incorrect, an error message will be issued. The error message will indicate the correct checkpoint ID to be used, and will cancel the restart job, allowing it to be submitted again with the correct checkpoint ID.
3. On a restart, parameter input is ignored by DL/I because the parameters were already read and saved when the job first started. However, if the parameter input 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.

Subtopics:

---



---

## 7.1.3.1.1 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/DOS/VS
             partition under MPS operation.  STXIT linkage to
             DL/I for abnormal task termination, normally controlled
             by UPSI bit 7, as always active under MPS operation.
```

# 7.1.3.2 MPS Parameter Information Requirements

The following information, beginning in column 1, must be entered from either SYSIPT or SYSLOG:

```
{DLI},progname,psbname[,RC={YES}]
{DLR}                     {NO}
```

**DLI|DLR**
Specify DLR if the MPS Restart facility is to be used; otherwise, DLI is required. When using the MPS Restart facility, you must specify DLR when the program is first executed, not just when it is restarted.

**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.

**RC=**

**NO** Suppress conditional JCL support (the default).
**YES** Invoke conditional JCL support.

For more information on this parameter see "Return Code Information" in topic 7.1.1.2.1.

Any other parameters on the DL/I parameter statement are ignored. The parameter statement information is printed on SYSLST.

For more information about the MPS Restart facility, refer to *DL/I DOS/VS Data Base Administration*.

Subtopics:

## 7.1.3.2.1 Examples

***Example 1:*** Shown below are the execution job control statements for an MPS batch program, INVUPDT, with a PSB of INVMSTR. For the MPS environment, data base logging is controlled in the CICS/DOS/VS partition.

The UPSI statement is optional and when set to all zeros, as shown, it may be omitted.

```
// JOB     UPDATE
// UPSI    00000000
// EXEC    DLZMPI00,SIZE=. . .
DLI,INVUPDT,INVMSTR,RC=YES
             .
             .
             DATA CARDS IF REQUIRED
             .
             .
/*
/&
```

***Example 2:*** Shown below are the execution job control statements for the same MPS batch program using MPS Restart. Included in this example are statements which assign a tape to contain checkpoint records written by VSE CHKPT.

```
// JOB     UPDATE
// MTC     REW,280
// ASSGN   SYS100,280
// EXEC    DLZMPI00,SIZE=. . .
DLR,INVUPDT,INVMSTR
             .
             .
             DATA CARDS IF REQUIRED
             .
             .
/*
/&
```

***Example 3:*** The job control statements in the following example will restart the job in example 2 from checkpoint 0010. Note that the jobname must be the same on the restart job as it was on the job that failed. Also, the PSB must be the original one (required for compare, but you should know that parameter input and UPSI byte setting are ignored by DL/I since they were already read and saved when the job was first started).

```
// JOB     UPDATE
// MTC     REW,280
// ASSGN   SYS100,280
// RSTRT   SYS100,0010
DLR,INVUPDT,INVMSTR
             .
             .
             DATA CARDS IF REQUIRED
             .
             .
/*
/&
```

 *© Copyright IBM Corp. 1981, 1989*

# 7.1.3.3 Dynamically Selecting MPS or Non-MPS

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/DOS/VS is active and MPS 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 was in operation or not and fetch the corresponding phase. The program that can be used is DLZCTRL, which will fetch either DLZRRC00 (if MPS is not active) or DLZMPI00 (if MPS is already started). Specifying DLZCTRL on the EXEC statement performs dynamic scheduling of MPS or non-MPS execution. In that case you should use the same job control statements as for batch applications. No other changes in the job control statements would be required.

DLZCTRL uses the XPCC FUNC=IDENT macro to test for the presence of the Start Partition XPCCB of MPS (APPL=SYSDLIO1, TOAPPL=SYSDLIB1):

- If this XPCC communication link in MPS online is currently set up (R15 not 0 and return code IJBXRETC reflects IDENT already done), then MPS is active and the program fetches the phase DLZMPI00.

- If MPS is not active, the program fetches 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 sample program DLZCTRL (shown below) is included (as type .A) in the 'DL/I prod lib'.

```
DLZCTRL  CSECT
         BALR  R12,0                   ESTABLISH BASE REGISTER
         USING *,R12
         OPEN  PRINTER,CONSOLE
         SUBSID NOTIFY,NAME=DLIID      MAKE DL/I SUBSYSTEM KNOWN
         LTR   R15,R15
         BZ    SUBOK                   NO ERRORS DETECTED: CONTINUE
         MVC   IOAREA(L'SUBEMSG),SUBEMSG
         LA    R7,L'SUBEMSG
         BAL   R3,OUTPUT               PRINT SUBSID ERROR MESSAGE
         CANCEL
SUBOK    LA    R8,MPSXPCC
         USING IJBXPCCB,R8
         XPCC  XPCCB=(R8),FUNC=IDENT   FIND OUT IF MPS IS ACTIVE
         LTR   R15,R15
         BZ    NOMPS                   MPS IS NOT ACTIVE
         TM    IJBXRETC,IJBXDAPP+IJBXAPSP
         BZ    NOMPS                   MPS IS NOT ACTIVE
         LA    R2,=C'DLZMPI00'         MPS IS ACTIVE: USE DLZMPI00
         MVC   IOAREA(L'MPSMSG),MPSMSG
         LA    R7,L'MPSMSG
         BAL   R3,OUTPUT
         B     PGMEX
NOMPS    LA    R2,=C'DLZRRC00'         MPS NOT ACTIVE: USE DLZRRC00
         MVC   IOAREA(L'NOMPSMSG),NOMPSMSG
```

```
            LA    R7,L'NOMPSMSG
            BAL   R3,OUTPUT
PGMEX       XPCC XPCCB=(R8),FUNC=TERMIN   EXECUTE MPS OR BATCH
            SUBSID REMOVE,NAME=DLIID
            FETCH (R2)                    FETCH APPROPRIATE PHASE
OUTPUT      PUT   PRINTER                 PRINT MESSAGES
            PUT   CONSOLE                 DISPLAY MESSAGES
            CLOSE PRINTER,CONSOLE
            BR    R3
MPSMSG      DC    C'MPS ACTIVE - WILL EXECUTE DLZMPI00'
NOMPSMSG    DC    C'MPS NOT ACTIVE - WILL EXECUTE DLZRRC00'
SUBEMSG     DC    C'SUBSID NOTIFY=DLI ERROR - TERMINATE'
DLIID       DS    0CL8                    DL/I SUBSYSTEM ID
            DC    CL7'DLI    '
            DC    X'00'
IOAREAC     DS    0CL81
            DC    X'F1'
IOAREA      DC    CL80' '
MPSXPCC     XPCCB APPL=SYSDLIO1,TOAPPL=SYSDLIB1,BUFFER=(B1,B1LN)
B1          DS    CL60
B1LN        EQU   *-B1
R2          EQU   2
R3          EQU   3
R7          EQU   7
R8          EQU   8
R12         EQU   12
R15         EQU   15
PRINTER     DTFDI DEVADDR=SYSLST,IOAREA1=IOAREAC,RECSIZE=81
CONSOLE     DTFCN DEVADDR=SYSLOG,IOAREA1=IOAREA,BLKSIZE=80,                X
                  RECSIZE=(7),RECFORM=UNDEF,MODNAME=DLZCONSL
            MAPXPCCB
            END
```

Note that since the data base I/O operations are being carried out in the CICS/DOS/VS partition for the batch MPS job, the job accounting information for the batch partition will not reflect any data phase 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/DOS/VS partition, the data base I/Os and associated CPU time for the data base call processing will be charged to the DL/I batch partition controller task, CSDC.

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 7.1.4 CICS/DOS/VS - DL/I Tables - Requirements

Before the DL/I online system can be executed in a CICS/DOS/VS environment, certain CICS/DOS/VS components must be redefined to allow the addition of DL/I. Additionally, specific entries in various CICS/DOS/VS and DL/I tables are required for multiple partition support (MPS), program isolation (PI), the run and buffer statistics function, and for the use of the CICS/DOS/VS monitoring facility (CMF).

Examples of the statements and parameters required in each CICS/DOS/VS and DL/I table to support these functions follow.

The numbers preceding the statements refer to the comments following the table example. The numbers are then continued for each table described to provide easy reference between tables.

**For More Information . . .**

about the CICS/DOS/VS tables described in this section, see *CICS/DOS/VS Resource Definition Guide*.

Subtopics:



## 7.1.4.1 System Initialization Table (SIT)

Code the DL1= operand. DL1=YES causes module DLZNUC to be loaded by the CICS/DOS/VS system initialization and the proper SIMODS sequence to be generated. DL1=xx causes the same action to take place with the exception that CICS/DOS/VS loads DLZNUC appended with the 2-character suffix supplied in the operand. If DL1=NO is coded, or if the parameter is omitted, DL/I is not loaded.

Code the PLTSD= operand with the suffix of the shutdown PLT and, optionally, the PLTPI= operand with the suffix of the startup PLT. Both PLT names are then coded in the program processing table (PPT).

Code MONITOR=PER to activate the performance class of CICS/DOS/VS Monitoring Facilities (CMF) for use by DL/I.

For full recovery/restart support, code START=AUTO. This provides for warm starts and emergency restarts of the CICS/DOS/VS system and is required when using the MPS Restart Facility. Other CICS/DOS/VS facilities, such as keypointing, are required to perform warm starts or emergency restarts. For more information, see *CICS/DOS/VS Resource Definition Guide*.

Here is a SIT coding example:

```
1. DFHSIT TYPE=CSECT,DL1=01,FCT=01,PCT=01,PPT=01,          X
          JCT=01,TST=01,ALT=01,DCT=01,MCT=01,PLTPI=SU,     X
          PLTSD=SD,                                        X
          MONITOR=PER,START=AUTO,KPP=YES,TSP=YES,DBP=YES
```

**Notes:**

1. Defines the SIT with a suffix of 01 for the CICS/DOS/VS FCT, PCT, PPT, JCT, ALT, DCT, and MCT; a suffix of SU for the startup PLT; SD for the shutdown PLT; and a suffix of 01 for the DL/I ACT. MONITOR=PER activates the performance class of CICS/DOS/VS Monitoring Facilitites.

   DBP=YES, START=AUTO, KPP=YES, and TSP=YES are coded to provide recovery/restart support. They are required if the MPS Restart Facility is used. Note that there must be entries in the PPT for the programs specified in the SIT.

 *© Copyright IBM Corp. 1981, 1989*

## 7.1.4.2 File Control Table (FCT)

Specify an entry in the FCT for each DL/I data base referenced either by an online DL/I transaction or by a batch DL/I MPS program. Code the DFHFCT macro as follows for each FCT entry to be added:

```
DFHFCT  TYPE=DATASET,
        DATASET=dbdname,
        ACCMETH=DL/I,
        OPEN={INITIAL }
             {DEFERRED}
```

Note that if OPEN=DEFERRED is coded, the data base must be opened using the DL/I system call STRT before the data base can be used.

Here is a FCT coding example:

```
2. DFHFCT TYPE=INITIAL,SUFFIX=01
            .
            .
            .
3. DFHFCT TYPE=DATASET,DATASET=DB1,ACCMETH=DL/I,OPEN=...
   DFHFCT TYPE=FINAL
```

**Notes:**

2. Defines the FCT with a suffix of 01 to match the SIT specification in (1).
3. Example of a DL/I data base FCT entry. Because these entries are only referenced during open processing, they should be placed at the end of the FCT.

 *© Copyright IBM Corp. 1981, 1989*

## 7.1.4.3 Program Control Table (PCT)

Specify an entry in the PCT for each transaction that may be processed by the system. Include entries for the five DL/I MPS transactions (CSDA, CSDB, CSDC, CSDD, and CSDP). Because these transactions are probably of low activity compared to most of the transactions in your system, their entries should be placed toward the end of the PCT. They should also be specified as CLASS=LONG. Transactions CSDB and CSDC should be marked for dynamic transaction backout (DTB=YES). This is a requirement if the MPS Restart Facility is used.

If program isolation (PI) is used, all DL/I transactions that modify data bases must be marked for dynamic transaction backout (DTB=YES).

A PCT entry is also required for the Run and Buffer Statistics function transaction (CSDE).

Here is a PCT coding example:

```
4.  DFHPCT TYPE=INITIAL,SUFFIX=01
           .
           .
           .
5.  DFHPCT TYPE=ENTRY,PROGRAM=DL1PROG,DTB=YES,TRANSID=ABCD...

6.  DFHPCT TYPE=ENTRY,PROGRAM=DLZMSTR0,TRANSID=CSDA,CLASS=LONG
7.  DFHPCT TYPE=ENTRY,PROGRAM=DLZMPC00,TRANSID=CSDB,
           TWASIZE=488,CLASS=LONG,DTB=YES
8.  DFHPCT TYPE=ENTRY,PROGRAM=DLZBPC00,TRANSID=CSDC,
           TWASIZE=256,CLASS=LONG,DTB=YES
9.  DFHPCT TYPE=ENTRY,PROGRAM=DLZMSTP0,TRANSID=CSDD,CLASS=LONG
10. DFHPCT TYPE=ENTRY,PROGRAM=DLZSTTL,TRANSID=CSDE
11. DFHPCT TYPE=ENTRY,PROGRAM=DLZMPUR0,TRANSID=CSDP,CLASS=LONG
    DFHPCT TYPE=FINAL
```

**Notes:**

4. Defines the PCT with a suffix of 01 to match the SIT specification in (1).
5. Defines a DL/I transaction (non-batch MPS) with dynamic transaction backout (DTB) specified as required for PI support.
6. Defines the MPS start transaction.
7. Defines the MPS master partition controller transaction. Note that DTB support is specified for this transaction. This is required if the MPS Restart facility is used.
8. Defines the MPS batch partition controller (BPC) transaction. Note that DTB support is specified for this transaction. Its use will cause the DL/I updates made by a batch MPS program to be backed out if the batch program abnormally terminates. This is required if the MPS Restart Facility is used.
9. Defines the MPS stop transaction.
10. Defines the DL/I Run and Buffer Statistics function transaction.
11. Defines the MPS Restart Purge Temporary Storage transaction.

*© Copyright IBM Corp. 1981, 1989*

---

---

## 7.1.4.4 Processing Program Table (PPT)

Make an entry in the PPT for each transaction defined in the program control table. This includes program entries for the five MPS transactions (DLZMSTR0, DLZMPC00, DLZBPC00, DLZMSTP0, and DLZMPUR0) and also for the run statistics transaction (DLZSTTL). Because of their low usage, specify DLZMSTR0, DLZMSTP0, and DLZMPUR0 as RES=PGOUT. DLZMPC00 and DLZBPC00 should be specified as RES=YES.

Always define an entry in the PPT for the DL/I system termination program (DLZSTP00). Also if you want a DL/I formatted dump to be printed in case of a CICS/DOS/VS system ABEND, make an entry for the DL/I formatted dump program (DLZFSDP0). Because DLZSTP00 and DLZFSDP0 are only used once per CICS/DOS/VS session, specify them as RES=PGOUT for best CICS/DOS/VS performance.

If program isolation or the MPS Restart Facility are used, include a PPT entry for the CICS/DOS/VS dynamic transaction backout program (DFHDBP2$). This may be specified by using TYPE=GROUP,FN=BACKOUT.

If the Execution Diagnostic Facility (EDF) is used, the PPT must contain an entry for module DLZHLPI to process HLPI commands.

Here is a PPT coding example:

```
12. DFHPPT TYPE=INITIAL,SUFFIX=01
13. DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTR0,RES=PGOUT
14. DFHPPT TYPE=ENTRY,PROGRAM=DLZMPC00,RES=YES
15. DFHPPT TYPE=ENTRY,PROGRAM=DLZBPC00,RES=YES
16. DFHPPT TYPE=ENTRY,PROGRAM=DLZMSTP0,RES=PGOUT
17. DFHPPT TYPE=ENTRY,PROGRAM=DLZMPUR0,RES=PGOUT
18. DFHPPT TYPE=ENTRY,PROGRAM=DLZHLPI
19. DFHPPT TYPE=ENTRY,PROGRAM=DLZSTP00,RES=PGOUT
20. DFHPPT TYPE=ENTRY,PROGRAM=DL1PROG
21. DFHPPT TYPE=ENTRY,PROGRAM=DLZSTTL
22. DFHPPT TYPE=ENTRY,PROGRAM=DFHPLTSU
23. DFHPPT TYPE=ENTRY,PROGRAM=DFHPLTSD
24. DFHPPT TYPE=ENTRY,PROGRAM=DLZFSDP0,RES=PGOUT
25. DFHPPT TYPE=GROUP,FN=(AKP,BACKOUT,RECOVERY)
              .
              .
              .
    DFHPPT TYPE=FINAL
```

**Notes:**

12. Defines the PPT with a suffix of 01 to match the SIT specification in (1).
13. Defines the MPS start program referred to by (6).
14. Defines the MPS master partition controller program referred to by (7).
15. Defines the MPS batch partition controller program referred to by (8). Note that this program is also defined in the DL/I ACT (57).
16. Defines the MPS stop program referred to by (9).
17. Defines the MPS Restart Purge Temporary Storage program referred to by

```
(11).
18. Required by the CICS/DOS/VS execution diagnostic facility to process
HLPI commands.
19. Defines the DL/I system termination program.
20. Example of a DL/I application program PPT entry.  Note that this
program is also defined in the CICS/DOS/VS ALT (40) and in the DL/I ACT
(56).
21. Defines the statistics program referred to by (10).
22. Specifies the Startup PLT entry referenced in (1) and defined in (31).
23. Specifies the Shutdown PLT entry referenced in (1) and defined in
(33).
24. Defines the DL/I formatted system dump program.
25. Defines groups of CICS/DOS/VS facilities including keypointing and
dynamic backout which are required for full recovery/restart support.
These are necessary if the MPS Restart Facility is used.
```

---



---

## 7.1.4.5 Journal Control Table (JCT)

Make an entry in the JCT to define the CICS/DOS/VS system journal. The entry should include JOUROPT=(CRUCIAL,INPUT) for full recovery/restart support. This is required if the MPS Restart Facility is used. If PI is used, the DL/I log must be assigned to the CICS/DOS/VS system journal.

Journals used to record the monitoring facilities output data must also be defined in the JCT as user journals. Do this by specifying a JFILEID value between 02 and 99. You must also specify FORMAT=SMF so that the SMF block format is used instead of the CICS/DOS/VS block format.

Here is a JCT coding example:

```
26. DFHJCT TYPE=INITIAL,SUFFIX=01
27. DFHJCT TYPE=ENTRY,JFILEID=SYSTEM,JOUROPT=(CRUCIAL,INPUT),...
28. DFHJCT TYPE=ENTRY,JFILEID=02,FORMAT=SMF,...
              .
              .
              .
    DFHJCT TYPE=FINAL
```

**Notes:**

26. Defines the JCT with a suffix of 01 to match the SIT specification in (1).
27. Defines the CICS/DOS/VS system journal.
28. Defines the user journal used to record the monitoring facilities output data.

 *© Copyright IBM Corp. 1981, 1989*

## 7.1.4.6 Temporary Storage Table (TST)

If the MPS Restart Facility is used, 'DLZTSQ00' must be specified as a recoverable DATAID in the Temporary Storage Table.

Here is a TST coding example:

```
29. DFHTST TYPE=INITIAL,SUFFIX=01
30. DFHTST TYPE=RECOVERY,DATAID=DLZTSQ00
             .
             .
             .
    DFHTST TYPE=FINAL
```

**Notes:**

29. Defines the TST with a suffix of 01 to match the SIT specification in (1).
30. Defines the recoverable temporary storage queue used by the MPS Restart facility to maintain checkpoint IDs.

Sufficient space must exist in the Temporary Storage data set (DFHTEMP) for the temporary storage queue (TSQ) used by MPS Restart. Each entry in the TSQ is 40 bytes in length. The maximum number of entries is equal to the maximum number of MPS batch jobs which execute concurrently plus the number of MPS batch jobs which failed and are waiting to be restarted. Note that temporary storage space is also required by dynamic transaction backout (DTB) for each active batch partition controller (BPC) in the online partition. If sufficient space is not defined, BPCs and other tasks may go into a wait state. Care should be taken to avoid this situation by defining enough space.

 *© Copyright IBM Corp. 1981, 1989*

---

## 7.1.4.7 Program List Table (PLT)

Define an entry in the startup PLT for MPS start program DLZMSTR0 if you want it executed automatically as part of CICS/DOS/VS initialization processing. The startup PLT suffix must be defined in the SIT.

Define an entry in the shutdown PLT for MPS stop program DLZMSTP0 if you want it stopped automatically during CICS/DOS/VS shutdown processing. This entry must appear before the PLT DFHDELIM entry.

Define an entry in the shutdown PLT for DL/I system termination program DLZSTP00. This entry is always required. It must appear after the PLT DFHDELIM entry.

Define an entry in the shutdown PLT for statistics program DLZSTTL if you want it invoked automatically during CICS/DOS/VS shutdown. If this entry is specified, it must appear after the DFHDELIM entry.

Remember to define the shutdown PLT suffix in the SIT.

Here is a PLT coding example:

```
31. DFHPLT TYPE=INITIAL,SUFFIX=SU
32. DFHPLT TYPE=ENTRY,PROGRAM=DLZMSTR0
             .
             .
             .
    DFHPLT TYPE=FINAL

33. DFHPLT TYPE=INITIAL,SUFFIX=SD
34. DFHPLT TYPE=ENTRY,PROGRAM=DLZMSTP0
             .
             .
             .
35. DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
36. DFHPLT TYPE=ENTRY,PROGRAM=DLZSTP00
37. DFHPLT TYPE=ENTRY,PROGRAM=DLZSTTL
             .
             .
             .
    DFHPLT TYPE=FINAL
```

**Notes:**

31. Defines the startup PLT with a suffix of SU to match the SIT specification in (1).
32. The MPS start program is listed in the startup PLT so that MPS operation will be initiated automatically during CICS/DOS/VS initialization.
33. Defines the shutdown PLT with a suffix of SD to match the SIT specification in (1).
34. Defines the MPS stop transaction in the shutdown PLT so that MPS operation will be stopped during the first stage of CICS/DOS/VS shutdown processing.

```
35. DFHDELIM is a special CICS/DOS/VS delimiter entry for shutdown PLTs.
Programs listed after this entry are executed during the second shutdown
processing stage.
36. This required entry defines the DL/I system termination program.
37. This PLT entry causes statistics program DLZSTTL to be invoked
automatically during CICS/DOS/VS shutdown.
```

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 7.1.4.8 Application Load Table (ALT)

CICS/DOS/VS loads its application modules and tables based upon the order and specification in the ALT. Modules with little activity are placed in high address space while frequently used modules are placed in low address space. Note that some specifications made in the PPT can also be made in the ALT.

Here is an ALT coding example:

```
38. DFHALT TYPE=INITIAL,SUFFIX=01
            .
            .
            .
39. DFHALT TYPE=ENTRY,PROGRAM=DLZMPC00,ADRSPCE=LOW,
           PAGEOUT=NO,ALIGN=...
40. DFHALT TYPE=ENTRY,PROGRAM=DL1PROG,ADRSPCE=LOW,...
41. DFHALT TYPE=ENTRY,ALIGN=YES,ADRSPCE=HIGH,
           PROGRAM=(DLZMSTR0,DLZMSTP0,DLZSTP00,DLZFSDP0),
           PAGEOUT=YES
            .
            .
            .
    DFHALT TYPE=FINAL
```

**Notes:**

38. Defines the ALT with a suffix of 01 to match the SIT specification in (1).
39. 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/DOS/VS application programs. The CICS/DOS/VS statistics can be used to determine this for your system.
40. Makes this DL/I application 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/DOS/VS application programs. The CICS/DOS/VS statistics can be used to determine this for your system.
41. 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/DOS/VS is requested to page them out (PAGEOUT=YES). They are aligned to allow a page out that will not carry code for other programs with it.

## 7.1.4.9 Monitoring Control Table (MCT)

Entries must be made in the MCT if DL/I is to use the CICS/DOS/VS Monitoring Facility (CMF) to collect performance data. These entries define clocks and counters used to record the monitored events. An entry must also be made to define the CICS/DOS/VS user journal to which the collected performance data is to be sent.

Here is an MCT coding example that activates all of the DL/I clocks and counters.

```
42. DFHMCT TYPE=INITIAL,SUFFIX=01
43. DFHMCT TYPE=EMP,ID=(PP,1),CLASS=PER,PER=SCLOCK(1)
    DFHMCT TYPE=EMP,ID=(PP,2),CLASS=PER,PER=PCLOCK(1)
44. DFHMCT TYPE=EMP,ID=(PP,3),CLASS=PER,PER=SCLOCK(2)
    DFHMCT TYPE=EMP,ID=(PP,4),CLASS=PER,PER=PCLOCK(2)
45. DFHMCT TYPE=EMP,ID=(PP,5),CLASS=PER,PER=SCLOCK(3)
    DFHMCT TYPE=EMP,ID=(PP,6),CLASS=PER,PER=PCLOCK(3)
46. DFHMCT TYPE=EMP,ID=(PP,7),CLASS=PER,PER=SCLOCK(4)
    DFHMCT TYPE=EMP,ID=(PP,8),CLASS=PER,PER=PCLOCK(4)
47. DFHMCT TYPE=EMP,ID=(PP,9),CLASS=PER,PER=SCLOCK(5)
    DFHMCT TYPE=EMP,ID=(PP,10),CLASS=PER,PER=PCLOCK(5)
48. DFHMCT TYPE=EMP,ID=(PP,11),CLASS=PER,PER=SCLOCK(6)
    DFHMCT TYPE=EMP,ID=(PP,12),CLASS=PER,PER=PCLOCK(6)
49. DFHMCT TYPE=EMP,ID=(PP,13),CLASS=PER,PER=SCLOCK(7)
    DFHMCT TYPE=EMP,ID=(PP,14),CLASS=PER,PER=PCLOCK(7)
50. DFHMCT TYPE=EMP,ID=(PP,15),CLASS=PER,PER=(MLTCNT(1,22))
51. DFHMCT TYPE=RECORD,CLASS=PER,DATASET=2,MAXBUF=2040,
           FREQ=100
    DFHMCT TYPE=FINAL
```

**Notes:**

42. Defines the MCT with a suffix of 01 to match the SIT.
43. Defines a clock and counter to monitor 'Task Scheduling Delay'.
44. Defines a clock and counter to monitor 'PSB Scheduling Delay'.
45. Defines a clock and counter to monitor 'Program Isolation
46. Defines a clock and counter to monitor 'Program Isolation Task Suspension Delay'.
47. Defines a clock and counter to monitor 'VSAM I/O Wait Time'.
48. Defines a clock and counter to monitor 'Time Spent in DL/I to Process DL/I Calls'.
49. Defines a clock and counter to monitor 'Time Spent Waiting for a Response to a Remote Call'.
50. Defines twenty-two counters used to record 'Number of DL/I Calls by Function'.

**For more information...**

about the clocks and counters in notes 43 through 50 above, refer to *DL/I DOS/VS Data Base Administration*.

51. Defines the CICS/DOS/VS user journal to which the collected data is sent. The user journal must be defined in the JCT as in (28).

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 7.1.4.10 Destination Control Table (DCT)

The run and buffer statistics function captures online (including MPS) statistics and writes them to the CICS/DOS/VS destination "CSSL." The CSSL must be defined as an extra-partition transient data set in the DCT. Data written to the CSSL is automatically printed during CICS/DOS/VS shut down.

Here is a DCT coding example:

```
52. DFHDCT TYPE=INITIAL,SUFFIX=01
             .
             .
             .
53. DFHDCT TYPE=EXTRA,DESTID=CSSL,RESIDNT=NO,
           DSCNAME=PRINTER
             .
             .
             .
    DFHDCT TYPE=FINAL
```

**Notes:**

52. Define the DCT with a suffix of 01 to match the SIT specification in (1).
53. Defines the destination CSSL.

 *© Copyright IBM Corp. 1981, 1989*

## 7.1.4.11 Application Control Table (ACT)

The ACT is a DL/I table used to associate online application programs with one or more DL/I data bases. Chapter 13, "Defining Application and Storage Layout Control Tables" explains how to create an ACT.

Here is an ACT coding example showing its relationship to other CICS/DOS/VS tables:

```
54. DLZACT TYPE=INITIAL,SUFFIX=01
55. DLZACT TYPE=CONFIG,PI=YES,SLC=DLZSLC01,...
56. DLZACT TYPE=PROGRAM,PGMNAME=DL1PROG,PSBNAME=(...,)
57. DLZACT TYPE=PROGRAM,PGMNAME=DLZBPC00,PSBNAME=(...,)
             .
             .
             .
58. DLZACT TYPE=BUFFER,...
    DLZACT TYPE=FINAL
```

**Notes:**

54. Defines the DL/I ACT with a suffix of 01 to match the SIT specification in (1).
55. Defines the DL/I configuration. Note that in this example program isolation is used. Also note that the storage layout control table with the phase name DLZSLC01 will be used.
56. An example of a DL/I online application program entry. Note that this program is also defined in the CICS/DOS/VS PPT (20).
57. Defines all PSBs which 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 infrequently accessed relative to online DL/I application programs, it should be placed toward the end of the ACT. This program is also defined in the CICS/DOS/VS PPT (15). This is the only one of the four DL/I MPS programs that should be defined in the DL/I ACT.
58. Specifies the DL/I buffer pool characteristics.

## 7.1.4.12 Storage Layout Control (SLC)

The SLC is a DL/I table used to control the sequence in which DL/I action modules are to be loaded during initialization. Chapter 13, "Defining Application and Storage Layout Control Tables" explains how to create an SLC.

Here is an SLC coding example:

```
59. DLZSLC TYPE=INITIAL,SLC=DLZSLC01
60. DLZSLC TYPE=ENTRY,MODULE=ALL
    DLZSLC TYPE=FINAL
```

**Notes:**

59. Defines the phase name of the SLC to match the ACT specification in (55).
60. MODULE=ALL is coded in this example to specify that all DL/I action modules are to be loaded with the same LOAD and ALIGN options and in the same sequence provided as a default in the initializer.

# 8.0 Part 8. Loading Data Bases

This section explains how to load a data base initially, after a DBD, PSB, and ACB have been generated and space has been allocated. Loading is the process of storing data in a data base for the first time.

Chapter 16, "Initial Data Base Load" in topic 8.1 explains how to load a data base.

Chapter 17, "Partial Data Base Load" in topic 8.2 also explains data base loading, as in the previous chapter, but in more than one load step. This may be a good choice if your data base contains a large amount of data.

**Loading a Data Base Interactively**

The initial data base load procedure described in Chapter 16, Initial Data Base Load can also be performed interactively on a 3270-type terminal by using the Interactive Utility Generation facility (IUG). See *DL/I DOS/VS Interactive Resource Definition* for more information.

Subtopics:



# 8.1 Chapter 16. Initial Data Base Load

Subtopics:



## 8.1.1 Basic Loading

Initial data base loading is done by an application program written by someone in your installation. This program must run in the *batch* environment. Load programs cannot run in the online or MPS batch environments.

The program inserts the segments in hierarchical sequence in the space reserved for the data base data set. The load program uses only one type of DL/I request. This is either the DL/I High Level Programming Interface LOAD command, or the DL/I CALL interface ISRT statement. No other DL/I requests are allowed in this program.

**For more information...**

about the use of these requests in load programs see *DL/I DOS/VS Application Programming: High Level Programming Interface* or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*.

The processing option PROCOPT, specified in the PCB(s) for this program must be L or LS. If LS is specified, or if the organization is HD indexed or HISAM, or HSAM (simple HSAM) with keys, the data base must be loaded sequentially. PCBs associated with load programs must refer to physical DBDs, not logical DBDs.

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

When loading an HD indexed or HIDAM data base, DL/I also loads the primary index data base automatically. As each root segment is stored, DL/I generates the index segment for it and stores the index segment in the INDEX data base. The INDEX data base is a KSDS. Index segments created both during and after initial loading are placed in this data set. The data stored in an HD indexed or HIDAM data base is stored in an ESDS. The HD space search algorithm is used by DL/I to locate the most suitable space available for inserting each segment.

For initial loading of an HD randomized or HDAM data base, you must specify PROCOPT=L in the PCB. The data base records do not need to be sorted into root key order, but the segments must be inserted in hierarchical order. For performance reasons it is advantageous if the data base records are sorted into physical sequence. Physical sequence is the ascending sequence of the block and root anchor point values as generated by the randomizing algorithms.

When loading a given data base record, your randomizing module generates a relative block (control interval) and anchor point number for the root segment of the record. These values are passed to DL/I, which attempts to store the root segment in the control interval specified.

**Note:** You must load at least one data base record in every data base that will be used online, before trying to use the online DL/I system.

As with any DL/I application program, DL/I will return a status code after the execution of each DL/I request in your load program. The codes that are returned, and how they should be handled, are somewhat different for the two DL/I interfaces.

**For more information...**

about handling status codes, see *DL/I DOS/VS Application Programming: High Level Programming Interface* if your load program uses HLPI, or *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces* if it uses the CALL interface.

---



---

# 8.1.2 Loading Requiring Prefix Resolution

The data base loading program discussed above is required in every initial loading situation. However, there are situations that the loading program cannot handle. These involve the resolving of pointers in the prefixes of logically related segments. During the initial loading of data bases having logical relationships or secondary indexes, the sequence in which logical parent segments are loaded is normally not the same as the sequence in which the logical child segments are loaded. This makes it impossible to generate all of the pointers just mentioned. To handle this situation, DL/I provides:

- A work file that is automatically created whenever you load a data base that contains logical child and/or logical parent segments. This work file contains the information needed to update the pointers.
- A set of utility programs that use the work file information to perform the pointer updating.

The three following sections describe the loading of data bases with logical relationships or with secondary indexes, and the utility programs that perform the pointer resolution.

Subtopics:



## 8.1.2.1 With Logical Relationships

If a data base to be loaded contains segments involved in logical relationships, the logical relationship resolution utility must be executed.

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 (or ISRT statement). 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 the data base being initially loaded contains logical relationships, job control statements must be provided for the load 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 only be on 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 on 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 issued as a warning and 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).

**For more information...**

see "Loading Data Bases with Logical Relationships" in *DL/I DOS/VS Guide for New Users*.

---

## 8.1.2.2 With Secondary Indexes

There are two ways of creating secondary indexes. One way is to create them automatically during initial loading. This is accomplished if all DLBL statements for the secondary index data bases are included in the job stream for the initial load. 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. The 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. The logical relationship resolution utilities then 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).

**Notes:**

1. If DLBL and ASSGN statements for secondary indexes are not included in the load job stream but exist in the standard labels, secondary indexes will be built automatically during loading.

2. 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), and loading continues.

3. It is not necessary to provide DLBL statements for secondary index files which are not referenced (that means no index source segments are loaded) during the load process). At least one record will be written into the work file for each defined secondary index to enable the prefix update utility to build the corresponding index file.

## 8.1.2.3 Utility Programs

DL/I provides a set of four utility programs to help you with the initial loading of data bases containing logical relationships. They are called logical relationship resolution utilities. These utilities are not concerned with the actual loading of data into data bases. That is the function of the loading program written by someone in your installation. The major function of the utilities is the resolution of pointers in the prefixes of logically related segments.

The four programs (described later in this manual) are the:

- Pre-reorganization utility (DLZURPR0)

  This program generates a control data set that controls the execution of the load program and the other logical relationship resolution utilities. It must be run first.

- Scan utility (DLZURGS0)

  This program searches designated data bases for segments that are involved in logical relationships. It generates one or more output records for each found segment. These output records are input to the prefix resolution utility.

- Prefix Resolution utility (DLZURG10)

  The main function of this program is to combine and sort (in physical data base sequence) all work files that are defined as input to it. These input files are generated by the prereorganization utility, the scan utility, the HD reorganization reload utility, or your load program. The prefix resolution utility also checks for missing logical parents.

- Prefix Update utility (DLZURGP0)

  This program applies the necessary changes to the prefixes of segments involved in logical relationships that could not be resolved previously. Its input is the file generated by the Prefix Resolution utility. After successful execution of this program, the data base(s) are ready for use.

**Loading a Data Base Interactively**

You can code the DL/I logical relationship utility job streams or, if you have a 3270-type terminal available, you can use the Interactive Utility Generation (IUG) facility to generate job streams for the utilities. For complete information on IUG and how to use it, see *DL/I DOS/VS Interactive Resource Definition*.

*© Copyright IBM Corp. 1981, 1989*

---

# 8.1.3 Load Process

Figure 53 shows the programs required for logical relationship resolution during initial load of an HD data base. Note that this does not describe the entire utility execution flow. Additional unload/reload steps may be required if any reorganization or changes are to be made to data bases involved in logical relationships with those to be initially loaded.

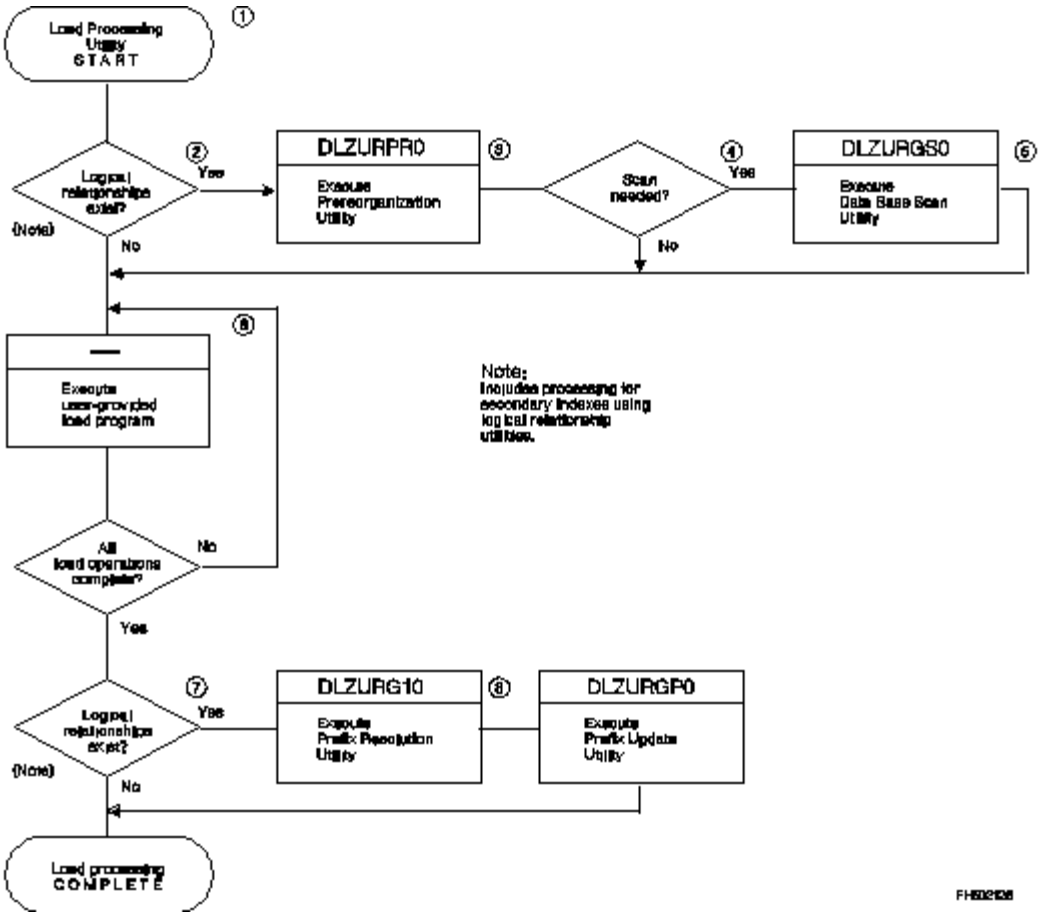



Figure 53. Initial Data Base Load Procedure. Numbers refer to Notes below.

**Notes:**

1. The 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, 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, 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 Chapter 24, "Data Base Prefix Resolution Utility (DLZURG10)" and Chapter 25, "Data Base Prefix Update Utility (DLZURGP0)."

2. The YES branch must be taken if any segment in any data base being operated upon is involved in a logical relationship. Taking the YES branch is also recommended when loading data bases with secondary index relationships; see the section "With Secondary Indexes" 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 loaded, 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 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. 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 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" later in this chapter.
8. The prefix resolution utility combines the output from the scan utility, the HD reorganization reload utility (if applicable), and the user initial data base load program to create an output file to be used by the prefix update utility. That utility then completes all logical relationships defined for the data bases that were operated upon.

---

# 8.1.4 Load Processing Example

The example shows initial loading of two data bases with logical relationships and secondary indexes.

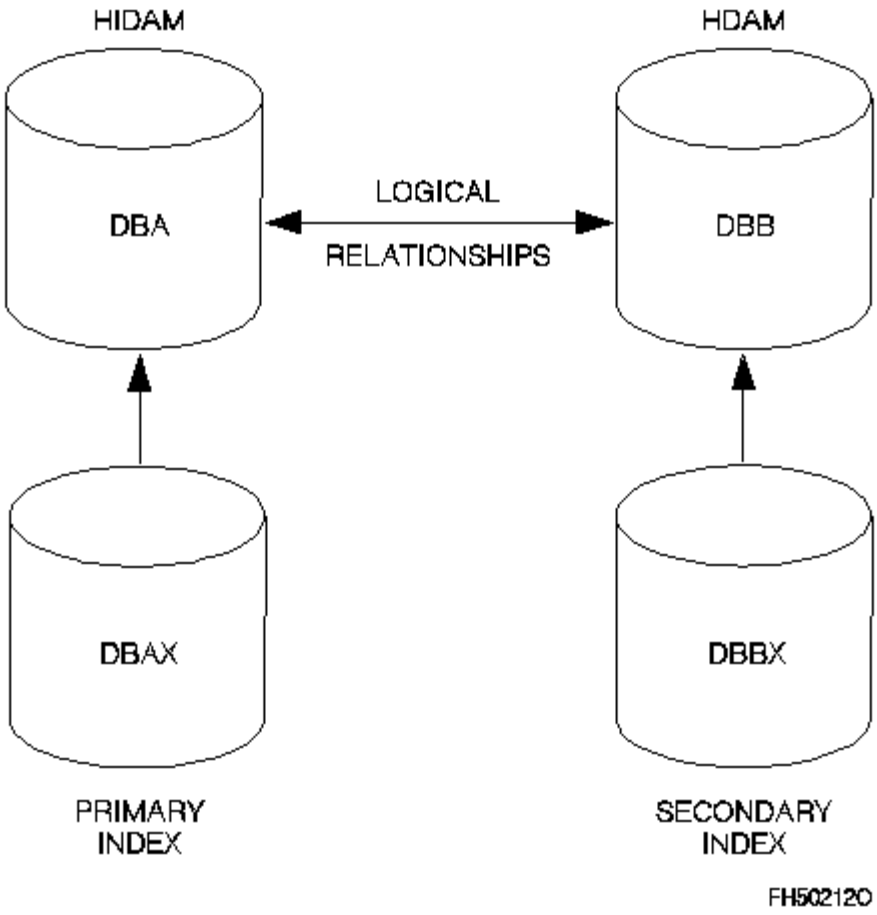



Figure 54. Typical Logically-related Data Bases for Initial Load Processing

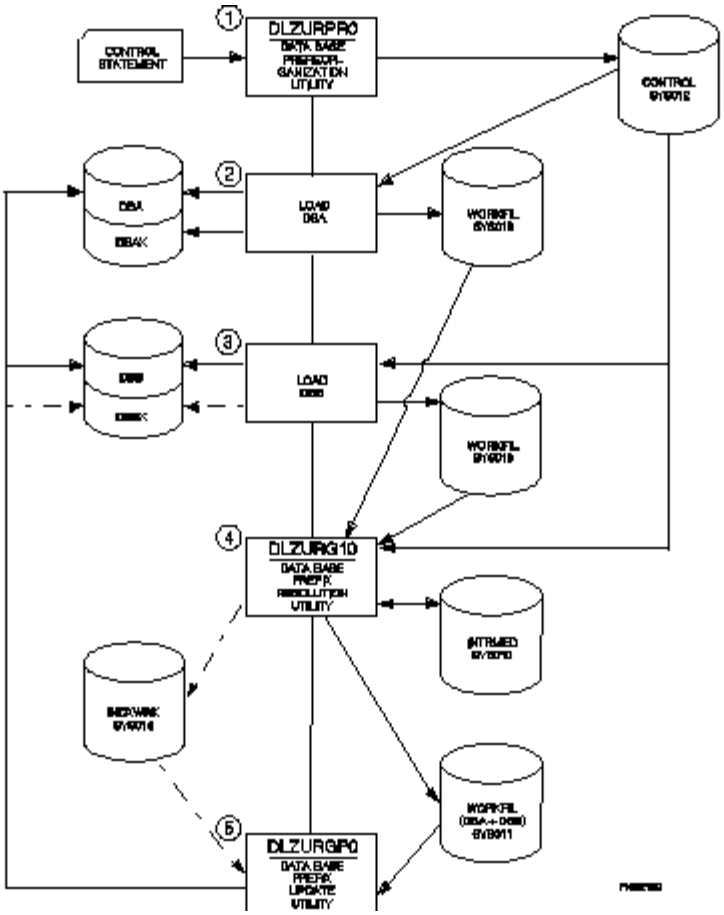



Figure 55. Initial Load of Two Data Bases with Logical Relationships and Secondary Indexes

---



---

# 8.2 Chapter 17. Partial Data Base Load

The "Partial Data Base Load" feature allows initial loading of a data base (in batch environment) with a large amount of data in several steps. That is, DL/I allows more than one load step with PROCOPT = L/LS.

The rules and requirements introduced in Chapter 16, "Initial Data Base Load" apply also to this chapter.

Subtopics:



## 8.2.1 Invocation

For the several partial load steps, the following parameter statements must be provided:

° For the first and additional load steps, except the last one:

PLS,programname,psbname,.....

The used PSB must be defined with PROCOPT = L/LS. (***PLS***: ***P***artial ***L***oad ***S***tep)

° To terminate the load process for a data base following parameter statement must be issued for the ***last*** load step:

PLC,programname,psbname,.....

The used PSB must be defined with PROCOPT = L/LS. (***PLC***: ***P***artial ***L***oad ***C***omplete)

**Note:** ***No further load steps are allowed***.

 *© Copyright IBM Corp. 1981, 1989*

## 8.2.2 General Rules for Partial Data Base Load

1. All load steps must be performed with DL/I 1.8.0 or later. DBDGEN, PSBGEN and ACBGEN are not affected.

2. The PSBs used for the various load steps must be defined with PROCOPT=L/LS, otherwise message DLZ023I is printed and status code = 'AI' is returned.

3. The data base organizations HSAM, SHSAM, HISAM and SHISAM are not supported. Message DLZ035I is issued and the job is canceled if these DB organizations are recognized along with partial data base load.

4. Each second and further load step must start by inserting a root segment. If insertion of another segment type is attempted, status code 'LD' is returned.

   - Data Base Organization = HIDAM

     The first root segment that will be inserted must have a key greater than all other keys that already exist in the data base; otherwise status code 'LB' or 'LC' is returned.

   - Data Base Organization = HDAM, PROCOPT=LS

     The first root segment that will be inserted must have a key greater than (or equal to, if multiple keys are allowed) all other keys already existing in the data base; otherwise status code 'LB' or 'LC' is returned.

   - Data Base Organization = HDAM, PROCOPT=L

     The sequential key of the root is not checked.

5. In all steps the segments to be inserted must be in the sequenced order that DL/I Initial Load requires, otherwise status codes 'LB', 'LC' or 'LE' are returned.

6. It is the user's responsibility to guarantee identical DBD and PSB definitions during the entire load process. Adding, changing and deleting of segment types and/or logical relationships cannot be recognized by DL/I and would cause unpredictable results.

7. To keep the state of a data base between the several load steps, the DL/I control record - included in the first CI of the data base ESDS file - is extended. This additional information will be removed after successful run of the prefix update utility, or after the last partial load step if prefix update is not required.

8. A DL/I DOS/VS data base is not compatible with IMS/VS (DB part) during

```
     the entire partial load process.  After completion of the load
     process, either by issuing parameter statement PLC,...., or by running
     the prefix update utility (if logical relationships exist or secondary
     indexes must be built using a work file) the DL/I DOS/VS data base can
     be processed by IMS/VS.


  9. If a partial load step fails, the data base must be restored to the
     state it was in after the last successful partial load step.
```

Subtopics:



---

---

## 8.2.2.1 With Secondary Indexes

During the various load steps DL/I must recognize the status of a secondary index. Therefore, an additional eight bytes of control information are used for each defined secondary index in the DL/I control record (which is in the first CI of the data base ESDS).

Depending on the CI size, the following restrictions are introduced with partial data base load:

| CI size in bytes | Number of allowed secondary indexes |
|---|---|
| 512 | 53 |
| 1024 | 117 |
| 1536 | 181 |
| 2045 | 245 |
| >2048 | 254 |

If one of these limits is violated, message DLZ036I will be issued and status code 'AI' returned.

If at least one secondary index is not built directly, an output work file (WORKFIL) must be built during each load step, serving as input for the prefix resolution utility. If opening of this work file fails, messages DLZ007I and DLZ864I are issued and status code 'AI' is returned.

The method to build a secondary index - chosen by the first load step - *must not be changed during the entire load process*, but different methods can be used for the various indexes. If DL/I recognizes either a change of the method to build a secondary index, or a new or a missing definition for a secondary index, either message DLZ020I (secondary index file open error) or message DLZ007I (work file open error) and/or message DLZ034I are issued and status code 'AI' is returned.

## 8.2.2.2 With Logical Relationships

```
If at least one logical relationship is defined, the input work file
CONTROL (built by Data Base Prereorganization Utility) and the output work
file (WORKFIL, serving as input for the prefix resolution utility) must be
assigned, even if only logical parents will be inserted during the entire
load process.  Otherwise messages DLZ007I and DLZ864I are issued and
status code 'AI' is returned.
```

 *© Copyright IBM Corp. 1981, 1989*

## 8.2.2.3 Workfile Requirements during Partial Data Base Load

If at least one secondary index is not built directly, or if logical relationships exist, the user must use different work files (WORKFIL) for each partial load step. All of these generated work files must be saved for execution of the prefix resolution utility.

As the prefix resolution utility allows a maximum of 99 input work files, the number of partial data base load steps is limited by the following:

- If logical relationships exist, this limit relates to the sum of the load runs for *all* concerned data bases.

- If only secondary indexes are to be built using a work file, the limit relates to the load runs for the *currently built* data base.

# 8.2.3 Using the Prefix Update Utility

```
Running the prefix update utility to build logical pointers and/or
secondary index data bases is only allowed if the partial load process of
all related data bases have been completed with a parameter statement
beginning with PLC, or if the data bases have been loaded with one step
using a parameter statement beginning with DLI.  Otherwise messages
DLZ992I and DLZ982I are issued and the batch DL/I run will be terminated
with JCL return code 12.
```

## 8.2.4 Backup of a Partially Loaded Data Base

To backup a data base between two partial load steps, the data set image copy utility must be used to take a dump of the data base files.

To restore a partially loaded data base the data set recovery utility must run with the dumped files as the only input.

For further information on use of these utilities refer to the utility descriptions later in this manual.

**Note:** Other DL/I utilities are not adequate to process partially loaded data bases.

## 8.2.5 Retrieve Access to a Partially Loaded Data Base

Between several load steps, only retrieval calls may be issued in a batch as well as in an online environment. If the load process is not completed (either by issuing parameter statement PLC,...., or by running the prefix update utility (if logical relationships exist or secondary indexes have to be built using a work file) status code 'AM' is returned for update calls.

PCBs with an alternative processing sequence may only be used if the referenced secondary index data base will be directly built within the load process. Status code 'AM' will be returned if this rule is violated.

It is not recommended to use PCBs that belong to logical DBDs to access data bases which are not (successfully) completely loaded. The results may be unpredictable.

If a logical child of a partially loaded data base (within a physical DBD) should be retrieved, DL/I cannot provide, in front of the logical child segment, the concatenated key which identifies its logical parent. Instead of this key, 'blanks' are returned in the I/O area.

Fields of logical child segments - either part of the concatenated key or located within the logical parent - must not be referenced in qualified SSAs by retrieval calls to partially loaded data bases. In this case, status code 'AK' will be issued by DL/I.

All logical related and all secondary index data bases (even if the secondary index will be built via work file) must be defined in the CICS/DOS/VS File Control Table (FCT) if a retrieve access is desired in an online environment during partial data base load.

## 8.2.6 Effects on Batch Processing

Previous releases of DL/I opened the data base after some syntactic semantic checks of a passed DL/I call.

Now DL/I will immediately open the data bases after a correct function and a valid PCB are recognized, because partial data base load requires information from the data base file for analysis of this first DL/I function call.

To ensure clean processing, this is done for partial data base load as well as for all other types of batch processing. Therefore, instead of status codes 'AC', 'AD', 'AH', 'AJ', 'AK', 'AM', 'LB', 'LC', 'LD', 'LE', 'DJ', 'V1' (...), status code 'AI' will be returned if any necessary open fails.

- Effects on Load Process

  DL/I opens the primary index data base directly after having opened the 'real' data base. As it is useless to continue loading of a data base without having the primary index available, status code 'AI' (instead of 'NI' in previous releases of DL/I) is returned if this open fails. Therefore, no segment will be inserted into the ESDS file of the data base.

- Effects on Prefix Update Utility

  The sequence of processing the two input files and the method to open the files for building logical relationship prefexes is changed (see "Rules for Prefix Update Utility" above), but no restriction will arise.

  It is not necessary to provide DLBL statements for secondary index files which are not referenced (that is, no index source segments are loaded) during the load process. This is true within the Initial Data Base Load as well as Partial Data Base Load. (At least one record will be written into the work file for each defined secondary index to enable the prefix update utility to build the corresponding index file).

# 8.2.7 Partial Data Base Load Process

The following figures illustrate the actions required by partial data base load.

**Notes:**

1. VSAM - 'DEFINE CLUSTER' and DLBL statements for the primary index file are required for DB organizations HIDAM or HD INDEXED.

2. Missing DLBL statements for secondary indexes cause building of the index files using the DL/I utilities prefix resolution and prefix update. The method - chosen by the first load step - must be kept through the whole load process.

3. Prefix building - that is, use of prefix resolution and prefix update utilities - is only required if either logical relationship is defined or secondary indexes will be built using work files.

4. The following figures include two recommendations for building secondary indexes indirectly:

   - The VSAM - 'DEFINE CLUSTER' is not required for secondary index files which will be built indirectly before the first load step. That will occur before the run of the prefix update utility. Maintenance is, however, improved if all definitions are collected in one place.

   - The run of the prereorganization utility to build the work file CONTROL before the first load step is not necessary if no logical relationship is defined for the data base to be loaded. The DLBL statement for CONTROL may be missing in the several load steps, but this file is required by the following prefix resolution utility. It is meaningful to build the work file CONTROL in the same step as for logical relationship, to get an identical process - especially if logical relationship may be added later.

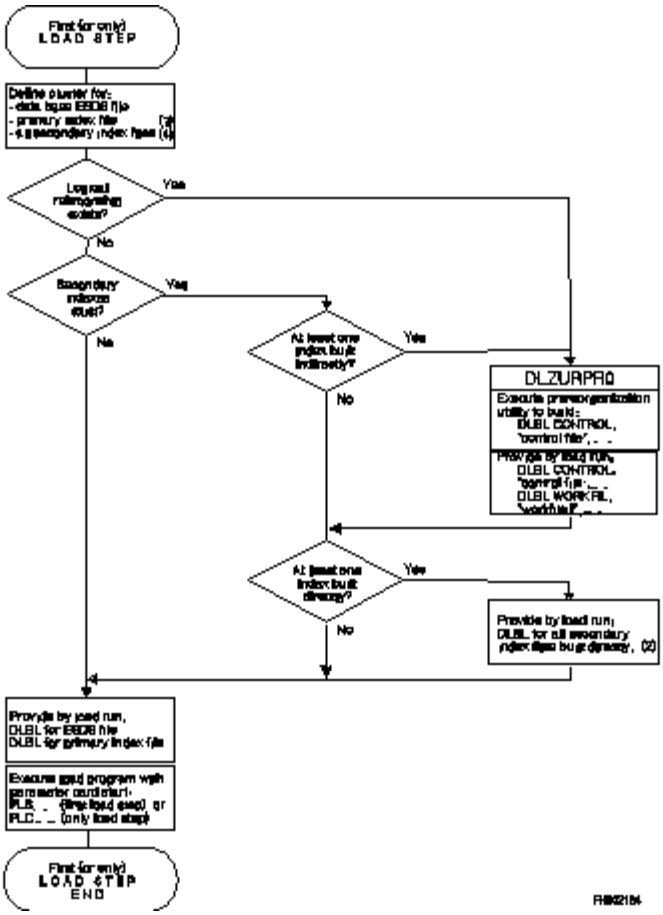



Figure 56. First (or only) Partial Data Load Step

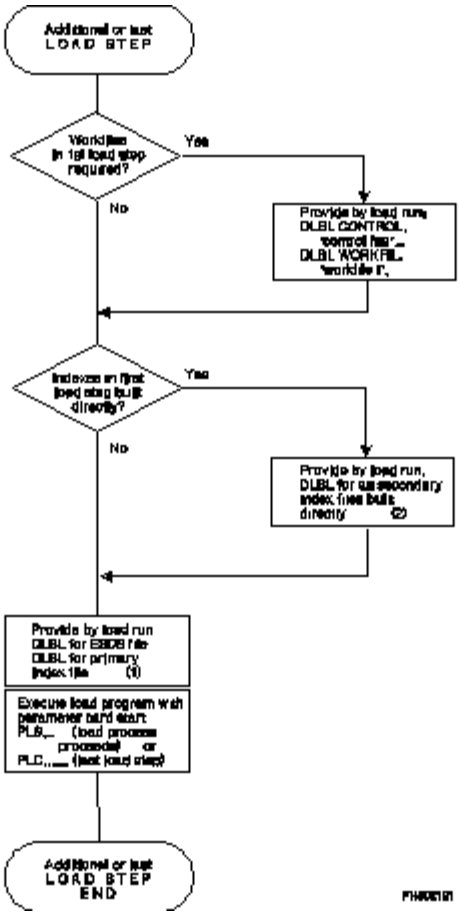
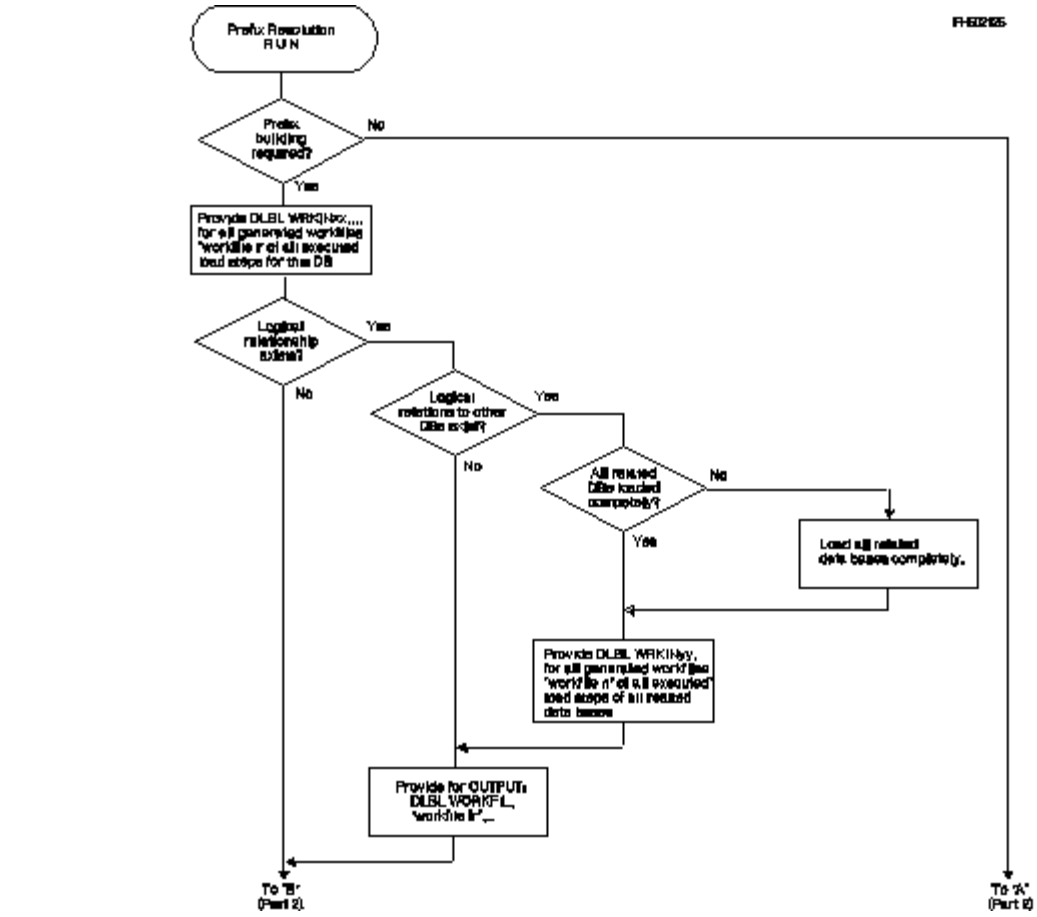



Figure 57. Additional or last Partial Data Load Step


Prefix Resolution
RUN
Prefix
building
required?
No
Yes
Provide DLBL WRKI-xx,...
for all generated workfiles
'workfile n' of all executed
load steps for this DB
Logical
relationship
exists?
Yes
No
Logical
relations to other
DBs exist?
Yes
No
All related
DBs loaded
completely?
No
Yes
Load all related
data bases completely.
Provide DLBL WRKI-yy,
for all generated workfiles
'workfile n' of all executed
load steps of all related
data bases
Provide for OUTPUT,
DLBL WORKFIL,
'workfile n',...
To 'B'
(Part 2)
To 'A'
(Part 2)

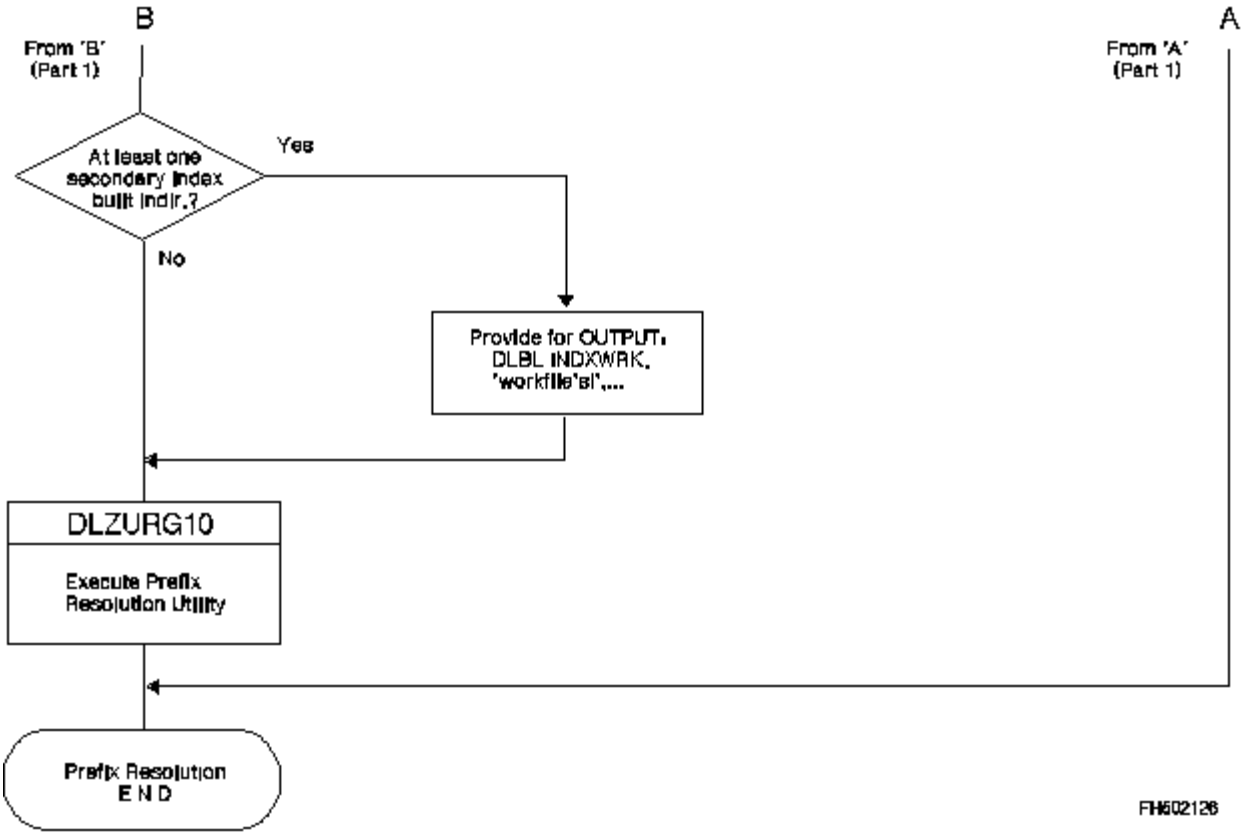

B
From 'B'
(Part 1)
A
From 'A'
(Part 1)
At least one
secondary index
built indir.?
Yes
No
Provide for OUTPUT,
DLBL INDXWRK,
'workfile'sl',...
DLZURG10
Execute Prefix
Resolution Utility
Prefix Resolution
E N D

Figure 58. Running the Prefix Resolution Utility

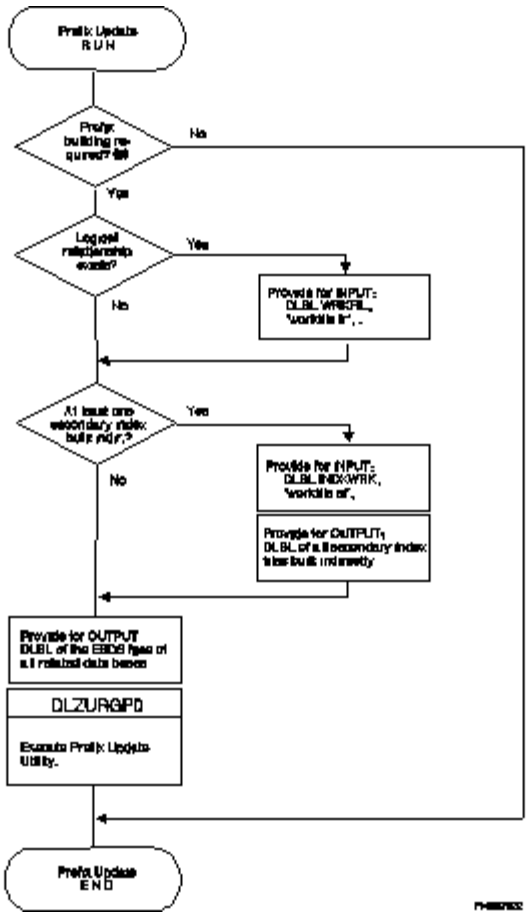



Figure 59. Running the Prefix Update Utility

---

# 9.0 Part 9. Data Base Reorganization Utilities

The following DL/I physical reorganization utilities and logical relationship resolution utilities are described in Chapter 18, "HD Reorganization Unload Utility (DLZURGU0)" through Chapter 26, "Partial Data Base Reorganization Utility (DLZPRCTn)." Each utility performs the indicated function.

Job streams for all utilities described in this section may be generated interactively on a 3270-type terminal by using the Interactive Utility Generation (IUG) Facility. See *DL/I DOS/VS Interactive Resource Definition and Utilities* for more information.

***Chapter 18, "HD Reorganization Unload Utility (DLZURGU0)" in topic 9.2***
This program unloads HD, HDAM, HIDAM, SHISAM, or HISAM data bases to a sequential format file.

***Chapter 19, "HD Reorganization Reload Utility (DLZURGL0)" in topic 9.3***
This program reloads HD, HDAM, HIDAM, SHISAM, or HISAM data bases from a sequential format file.

***Chapter 20, "HISAM Reorganization Unload Utility (DLZURUL0)" in topic 9.4***
This program unloads SHISAM, HISAM, or INDEX data bases to a sequential format file.

***Chapter 21, "HISAM Reorganization Reload Utility (DLZURRL0)" in topic 9.5***
This program reloads SHISAM, HISAM or INDEX data bases to a sequential format file.

***Chapter 22, "Data Base Prereorganization Utility (DLZURPR0)" in topic 9.6***
This program creates a control file that is used by the other logical relationship resolution utilities. It also indicates which data base and segments, if any, must be scanned by the data base scan utility.

***Chapter 23, "Data Base Scan Utility (DLZURGS0)" in topic 9.7***
This program scans any non-reorganized data bases that contain logical relationships that are affected by loading and/or reorganizing other data bases. Work files are generated for use by the data base prefix resolution utility.

***Chapter 24, "Data Base Prefix Resolution Utility (DLZURG10)" in topic 9.8***
This program combines and sorts all work files generated by the data base reload and data base scan utilities, or by internal DL/I programs during initial loading of a data base by a user-provided program. It generates one or more output work files that contain the prefix information needed to complete the loading and/or reorganization of data bases that contain logical and/or index relationships.

***Chapter 25, "Data Base Prefix Update Utility (DLZURGP0)" in topic 9.9***
This program uses the output file(s) generated by the data base prefix resolution program to update the prefix of each segment whose prefix information is affected by a data base load and/or reorganization.

***Chapter 26, "Partial Data Base Reorganization Utility (DLZPRCTn)" in topic 9.10***

```
    This program allows you to reorganize only that portion of data which is
    necessary.
```

Subtopics:



---

## 9.1 Restrictions

The following restrictions apply to the use of the data base logical relationship resolution utilities:

1. These utilities use the DOS/VS or DOS Sort/Merge program. Since the maximum sort field permitted by sort/merge is 256 characters, and DL/I requires 56 characters for control purposes, the following limits for any logical parent/logical child pair, must be observed:

   a. If the logical parent is being initially loaded, the length of its fully concatenated key must not exceed 200 characters.

   b. If the logical child is being initially loaded, the length of its sequence field plus the length of the logical parent's fully concatenated key (if the logical parent is also being initially loaded) must not exceed 200 characters.

   If the above limit check is not satisfied for either a logical parent or a logical child, the user can omit loading of the logical parent or logical child during initial data base loading. The logical parent or logical child can be inserted into the data base at a later time by an application program. Once a data base is loaded, the above limit checks do not apply.

   The data base Prereorganization utility performs the above limit checks for logical parent/logical child combinations. If the limit check fails for a logical parent/logical child combination, a warning message is issued.

2. The use of nonunique sequence fields for one or more segments in a data base allows the occurrence of nonunique fully concatenated keys for those segment types. If the user defines nonunique sequence fields for segments in a data base, and if multiple occurrences of a given segment type have the same fully concatenated key, certain ambiguities can arise during processing of the data base. In particular, for a segment type that has multiple occurrences of the same fully concatenated key and is also involved in a logical relationship, the following must be observed:

   a. During initial data base loading, the logical relationship resolution utilities relate all logical children that indicate a logical parent with a nonunique fully concatenated key to only one occurrence of the logical parent. Thus, one of the logical parents with a nonunique concatenated key is assumed to own all logical children pointing to that concatenated key, while other logical parents with that concatenated key are assumed to own no logical children.

   b. If a user has some mechanism other than a fully concatenated key for distinguishing logical parents with nonunique concatenated keys, the user may insert the logical children at any time, and not only during initial data base loading. In this manner, logical parents with nonunique concatenated keys can each own logical children. If logical parents with nonunique fully

concatenated keys do not occur, the logical relationship resolution utilities correctly maintain logical relationships during data base reorganization.

3. The user must ensure that if changes are to be made to the DBD between the time a data base is unloaded and the time it is reloaded, the following conditions must be observed:

   a. Names of existing segments must not be changed.

   b. The existing hierarchical structure must not be changed in any way that changes the fully concatenated key of any segment involved in a logical relationship.

4. It is invalid to initial load a data base containing a logical parent segment, while reorganizing the data base containing the logical child segment. This combination implies the existence of logical child segments without logical parent segments.

---

---

# 9.2 Chapter 18. HD Reorganization Unload Utility (DLZURGU0)

The HD reorganization utilities provide a means of reorganizing HD-organized data bases. If structural changes are to be made to a SHISAM or HISAM data base, the HD reorganization utilities must be used but, if no changes are to be made to SHISAM or HISAM data bases, the HISAM reorganization utilities should be used. The HD reorganization unload utility can be used with either HD randomized, HD indexed, HDAM, HIDAM, SHISAM, or HISAM data bases.

The general operation consists of issuing GET NEXT (GN) calls to access segments of the specified data base and writing a sequential output file containing each active segment, together with DL/I utility prefix information, arranged in hierarchical sequence. If all segments are deleted in the data base, no unload will be done and a later attempt to reload the empty data base will fail with message DLZ387I.

Several additional features are incorporated in this program.

1. By specifying an additional output unit, two copies of the data base can be created. In the event of a permanent I/O error on one output file, the other runs to completion, creating at least one usable copy.

2. A checkpoint facility is also incorporated. This facility takes checkpoints at intervals either specified by you or calculated by the utility (approximately every 5000 segments). Checkpoint records are numbered sequentially and the numbers are written to SYSLOG. The checkpoint is accomplished by writing a special checkpoint record onto the output file. If a restart is required, the message DLZ318I ENTER RESTART NUMBER nnnn is written to SYSLOG. The console operator enters the number of the last successful checkpoint record written. This number is provided in the checkpoint message written when each checkpoint is taken. The partially unloaded output file from the previously interrupted job is read as input. All records (including checkpoint records) are copied to the output file(s). When the correct checkpoint record is read, the proper position is established within the data base. The statistics table records are read into storage and the program then continues with normal processing.

3. A sequence check verifies the sequence of the data base. If a segment is found out of sequence, message DLZ400I SEQUENCE ERROR IN DATA BASE is issued to inform you of the problem. The sequence error should be corrected and the job resubmitted. A procedure that can be used to locate (in the dump) the segment that caused the sequence error is described in the *DL/I DOS/VS Diagnostic Guide* under the heading "DLZURGU0 - HD Reorganization Unload Utility."

4. The HD unload utility can also be used in combination with an application program PSB to produce a revised physical data base. This is done by selectively unloading only the sensitive data that represents the program's view of the data base. The selected data is then reloaded using a physical DBD which matches the PSB used for unloading. See "Doing a Selective Unload" in this chapter for more information.

Input to the HD reorganization unload utility consists of:

1. A DMB and PSB. Their names are obtained by DL/I by expanding the DBD name or PSB name specified in the parameter statement.

2. One of the following non-empty data bases to be unloaded:

   - HD randomized data base
   - HD indexed data base
   - HDAM data base
   - HIDAM data base
   - HISAM data base
   - SHISAM data base.

3. Primary and secondary index data bases related to the data base to be unloaded. The data bases are not unloaded.

4. If a restart is desired, a copy of the partially unloaded data base from the previously interrupted utility program execution must be included. The number of the last successful checkpoint record is entered via SYSLOG.

Output from the HD reorganization unload utility consists of:

1. One copy (HDUNLD1) or two copies (HDUNLD1 and HDUNLD2) of the unloaded data base, including checkpoint records.

2. Checkpoint data written to SYSLOG.

3. Messages and statistics written to the SYSLST device.

Subtopics:



---

## 9.2.1 Unload Output Statistics

The HD reorganization unload utility program maintains the following statistics:

- Total number of segments (by segment type) in the data base.
- Total number of segments in data base.
- Average number of segments (by segment type) in a data base record.
- Average number of bytes in a data base record.
- Maximum and average number of children for each segment type under an immediate parent segment.
- Maximum and average number of children (at all subordinate levels for all child segment types) under any parent segment.

Figure 60 is an example of the output messages and statistics obtained from the HD reorganization unload utility program. Following the page heading are the various messages generated as a result of the options selected for this execution. An explanation of all numbered messages may be found in *VSE/SP Messages and Codes*.

---

```
     DL/I DOS/VS  DLZURGU0   -  HD DATA BASE REORGANIZATION UNLOAD              DATE: 11/30/88
TIME: 13:30:22   PAGE    1


     DLZ345I     WARNING - HDUNLD2 NOT ASSIGNED, ONE COPY CREATED

     DATA BASE - DLIDBD4   HAS BEEN UNLOADED

     COPY 1 ON DISK

     DLZ339I     NO ERRORS DETECTED,  UNLOAD SUCCESSFUL

                                        D A T A   B A S E   S T A T I S T I C S


                  SEGMENT LEVEL STATISTICS                                                RECORD
```

```
LEVEL STATISTICS

     MAXIMUM      AVERAGE     MAXIMUM     AVERAGE             SEGMENT    SEGMENT       TOTAL
SEGMENTS      AVERAGE COUNT PER
      TWINS        TWINS      CHILDREN    CHILDREN             NAME       LEVEL        BY SEGMENT
TYPE     DATA BASE RECORD

          1          1.00        12          7.33             J1             1
3                 1.00

          2          1.00         2          0.66             J2             2
3                 1.00

          1          0.33         1          1.00             J3             3
1                 0.33

          1          1.00         0          0.00             J4             4
1                 0.33

          2          1.33         3          1.75             J5             2
4                 1.33

          1          1.00         2          0.75             J6             3
4                 1.33

          1          0.50         1          0.50             J7             4
2                 0.66

          1          0.50         0          0.00             J8             5
1                 0.33

          2          1.00         1          1.00             J9             2
3                 1.00

          1          1.00         0          0.00             J10            3
3                 1.00




                    TOTAL SEGMENST IN DATA BASE =          25        AVERAGE DATA BASE RECORD LENGTH
=      266 BYTES
```

---

Figure 60. Example of HD Unload Output Statistics

Following all the messages, the heading "Data Base Statistics" is printed. Then, under the subheading "Segment Level Statistics," the following fields occur:

***Maximum Twins:*** This field contains the maximum number of segments of this type encountered under an immediate parent segment. At the root level, this value is 1.

***Average Twins:*** The field contains the average number of segments of this

type encountered under an immediate parent segment. This value is calculated to two decimal places.

***Maximum Children:*** This field contains the maximum number of dependent segments (at all subordinate levels) under a given parent.

***Average Children:*** This field contains the average number of dependent segments (at all subordinate levels) under a given parent. This value is calculated to two decimal places. Note that the lowest level segment in any hierarchical path has a value of 0 in this field.

The next two fields are:

***Segment Name:*** The segment name to which this line of statistics applies.

***Segment Level:*** The hierarchical level of this segment in the data base. The segments are printed in the same order they were described in the DBD.

The two fields which occur under the subheading "Record Level Statistics" are:

***Total Segments by Segment Type:*** This field contains the total of the number of occurrences of this segment type in the entire data base. Note that the count field in the level 1 segment type reflects the total number of data base records (root segments) in the data base.

***Average Count per Data Base Record:*** This field contains the average number of occurrences of this segment type within a given data base record. The value is calculated to two decimal places.

The last two fields are:

***Total Segments in Data Base:*** This field contains a count of the total number of all segments in the data base.

***Average Data Base Record Length:*** The average data base record length includes both the data length and the prefix size. Note that the product of the number of segments at level 1 and the average data base record length is the total number of bytes in the data base. Since all physically stored records may not use all available data positions, this figure can only be used as an approximation of the file space required. For variable length segments, this statistic is not accurate because the maximum segment length is used in computing the statistics.

---

## 9.2.2 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // DLBL | filename | This statement defines the symbolic name of the data base file to be unloaded.<br><br>One statement must be present for each file that appears in the DBD describing this data base. One statement must also be present for each file that appears in DBDs of all other data bases that are related by logical relationships or secondary indexes.<br><br>For HD randomized and indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements (if any).<br><br>For HDAM, HIDAM (and primary INDEX), and SHISAM data bases, use the filename specified in the DD1 operand of DATASET statements.<br><br>For HISAM data bases use the filenames specified in the DD1 and OVFLW operands of DATASET statements. |
| // ASSGN<br><br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS010,{cuu}<br>{IGN}<br>RESTART ]<br><br>RESTART ]<br>extent data] | These statements define the input dump file if a utility restart is to be attempted. The file that defines the input dump file contains the partially dumped data base and is the same file as the HDUNLD1 (or HDUNLD2) from the previously interrupted unload utility. It may be DASD, or tape with standard labels. RESTART is the name of the input file. If no restart is to be performed IGN must be specified but the other statements in this group may then be omitted. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS011,cuu<br>HDUNLD1<br><br>HDUNLD1<br>extent data | These statements define the first copy of the output file which contains the dumped data base. It may be DASD, or tape with standard labels. HDUNLD1 is the symbolic name of the output file (DTF). |
| // ASSGN<br><br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS012,{cuu}<br>{IGN}<br>HDUNLD2 ]<br><br>HDUNLD2 ]<br>extent data] | These statements define the second copy of the output file if one is desired. The file may be DASD, or tape with standard labels. HDUNLD2 is the symbolic name of the output file. If a second copy is not desired, IGN must be specified, but the other statements in this group may then be omitted. |

| | | |
|---|---|---|
| // EXEC | DLZRRC00,<br>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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature*.

---

---

## 9.2.3 Parameter Statement Requirements

The HD reorganization unload utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column 1, must be entered either via SYSIPT or SYSLOG:

```
ULU,DLZURGU0,dbname[,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
                     {1  }
```

**dbname**
is the name of the DBD that contains the data base to be unloaded.

The positional parameter 'buf' and the keyword parameters HDBFR, HSBFR, and TRACE are optional parameters that may also be entered in the parameter statement. These parameters have the same usage as for application programs and are described in Chapter 15, "Executing DL/I Programs," under "DL/I Parameter Information Requirements."

If you are doing a selective unload, the parameter requirements are different. See "Doing a Selective Unload" in this chapter for details.

## 9.2.4 Control Statement Requirements

```
[REW=option]
```

The REW control statement invokes tape rewind options. Its use is optional.

If used, columns 1-4 must contain REW= and column 5 must contain one of these letters to indicate the tape option you want:

N - means no rewind,
R - means rewind only, and
U - means rewind and unload.

No other columns are used. Put the completed statement after the DL/I parameter statement in your job stream.

If this control statement is not used, tapes will automatically rewind and unload at end-of-job.

**Note:** When writing multiple files on a single tape, use one of the following to avoid overwriting a file:

1. Specify the N rewind option for all the files on the tape; then use an MTC command to rewind the tape when all jobs writing on the tape are finished.

2. Specify the R or U rewind option for a file which has a file sequence number on its corresponding TLBL statement. The R or U rewind option causes the tape to rewind when a file on the tape is "opened" and then again when it is closed. If a file sequence number is not specified in this case, the first file on the tape may be overwritten.

```
[CHKPT={NO    }]
       {xxxxxx}
```

The CHKPT control statement invokes the checkpoint option. Its use is optional. If used, column 1-6 must contain CHKPT= and must be immediately

followed by one of these options. Either,

- NO must be specified in columns 7-8 to indicate that no checkpoints are to be taken, or

- a 1-6 digit decimal number must be specified in columns 7-12 to indicate the number of segments that are to be processed before a checkpoint is taken.

No other columns are used. Put the completed statement after the DL/I parameter statement in your job stream.

If this control statement is not used, checkpoints are automatically taken at intervals of approximately 5000 segments.

---

---

## 9.2.5 Examples

***Example 1:*** This example shows the unloading of an HD indexed data base. Two DLBL statements are provided. The first one is for the filename defined in the DATASET statement DD1 operand and the second is for the filename defined in the ACCESS statement REF operand. A restart is not requested. One tape and one disk output copy are requested. Checkpoint records are taken at intervals of 3000 segments.

```
// JOB    HDUNLOAD1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDDB,'hd.data.base',,VSAM,CAT='catalog'
// DLBL   HDDX,'hd.primary.index',,VSAM,CAT='catalog'
// ASSGN  SYS010,IGN
// ASSGN  SYS011,180
// TLBL   HDUNLD1,'hdunload'
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   HDUNLD2,'hdunload',,VSAM,RECORDS=20,RECSIZE=6454,*
                DISP=(NEW,KEEP),CAT='catalog'
// EXTENT SYS012,'volser'
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGU0,DLIDBD3
CHKPT=3000
/*
/&
```

***Example 2:*** This example shows the unload of an HDAM data base using one output copy. A restart is not requested. One DLBL statement is provided for the HDAM data base. The output tape does not unload and no checkpoints are taken.

```
// JOB    HDUNLOAD2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// ASSGN  SYS010,IGN
// ASSGN  SYS011,180
// TLBL   HDUNLD1,'hdamunload'
// ASSGN  SYS012,IGN
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGU0,DLIDBD4
REW=R
CHKPT=NO
/*
/&
```

***Example 3:*** This example shows a restart after an abnormal termination of the execution of Example 1. The console operator must enter the number of

the last successful checkpoint record.

```
// JOB    HDUNLOAD3
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDDB,'hd.data.base',,VSAM,CAT='catalog'
// DLBL   HDDX,'hd.primary.index',,VSAM,CAT='catalog'
// ASSGN  SYS010,IGN
// TLBL   RESTART,'hdunload'
// ASSGN  SYS011,181
// TLBL   HDUNLD1,'hdunld1rst'
// ASSGN  SYS012,182
// TLBL   HDUNLD2,'hdunld2rst'
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGU0,DLIDBD3
CHKPT=3000
/*
/&
```

# 9.2.6 Doing a Selective Unload

The HD reorganization unload utility can also be used to do a 'selective unload.' This enables you, for example, to reformat data, add new fields for an application program, and move a subset of the data base to another location for faster processing. Segment and field level sensitivity functions are used to accomplish these changes.

To implement selective unload, the only change you have to make to the HD unload utility is in the parameter data entered either via SYSIPT or SYSLOG. For selective unload you must enter:

```
PLU,DLZURGU0,psbname[,{buf}]([,HDBFR=][,HSBFR=][,TRACE=]
                       {1  }
```

where psbname is the name of a generated PSB which may have fields in an order that is different from the physical order, or not have all segments or fields in the DBD.

Here is an example showing the selective unload of an HDAM data base using one output. One DLBL statement is provided to specify the filename defined in the DATASET statement DD1 operand. The output tape does not unload and no checkpoints are taken.

An example of the original physical data base, the PSB for unload, and the revised physical DBD is also shown.

```
// JOB    HDUNLOAD4
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// ASSGN  SYS010,IGN
// ASSGN  SYS011,180
// TLBL   HDUNLD1,'hdamunload'
// ASSGN  SYS012,IGN
// EXEC   DLZRRC00,SIZE=300K
PLU,DLZURGU0,PSB1
REW=R
CHKPT=NO
/*
/&
```

***Original Physical Data Base***

```
DBD      NAME=DB,ACCESS=HDAM,...
DATASET  DD1=HDAMDB,...
SEGM     NAME=SEG1,...
FIELD    NAME=(F1,SEQ,U),...
SEGM     NAME=SEG2,PARENT=SEQ1,...
FIELD    NAME=(TYPE,SEQ,U),...
FIELD    NAME=DATA,...
FIELD    NAME=MORE,...
SEGM     NAME=SEG3,...
   .
   .
   .
```

***PSB for Unload***

```
PCB     DBDNAME=DB,PROCOPT=G,...
SENSEG  NAME=SEG1,...
SENSEG  NAME=SEG2,...
SENFLD  NAME=TYPE,...
SENFLD  NAME=DATA,...
VIRFLD  NAME=V1,...
PSBGEN  PSBNAME=PSB1,...
   .
   .
   .
```

**Notes:**

1. The SENFLD statements unload fields TYPE and DATA. The field named MORE is not referenced and will not be unloaded. The SENFLD statements could specify a type of data that was different from that in the DBD causing data conversion. The SENFLD statements also could have reordered the fields.

2. The VIRFLD statement adds a virtual field.

3. The SEGM named SEG3 is not referenced and will not be unloaded.

After the selective unload is complete, it must be reloaded using a physical DBD which matches the PSB used for unloading.

***Revised Physical DBD***

```
DBD      NAME=NEWDB,ACCESS=HDAM,...
DATASET  DD1=HDAMDB1,...
SEGM     NAME=SEG1,...
FIELD    NAME=(F1,SEQ,U),...
SEGM     NAME=SEG2,...
FIELD    NAME=TYPE,...
FIELD    NAME=DATA,...
FIELD    NAME=V1,...
   .
   .
   .
```

---

# 9.3 Chapter 19. HD Reorganization Reload Utility (DLZURGL0)

The HD reorganization reload utility reloads a data base from a sequential file created by the HD reorganization unload utility. The user must execute the VSAM Access Method Services utility command DELETE to remove the name of the file from the VSAM catalog to release the space allocated for it. The user must then define the new file to VSAM to cause its definition to be recorded on the VSAM catalog. As explained below, the DMB of the data base being reorganized is part of the input for this utility. By replacing this DMB with a new version, certain structural changes can be made to a data base during the process of reorganization. The following rules and restrictions apply:

- HD reorganization unload must have been executed using the existing DMB.
- New segment types may be added in the DBD if they do not change the hierarchical relationship of existing segment types.
- Names of existing segment types must not be changed.
- The key field cannot be changed, added, or deleted.
- The DL/I access method may be changed.
- Segment pointer options for HD, HDAM, or HIDAM may be changed.
- An existing segment type may be deleted from the DBD if *all* segments of this type had been deleted from the data base prior to execution of the HD reorganization unload utility.
- Prior to executing the HD reorganization reload utility, the new DBD must be assembled, link-edited, and cataloged, and the existing DMB must be deleted. The application control blocks creation and maintenance utility must also be executed for all PSBs which reference the changed DBD and its output must be link-edited and cataloged. Of course, if no structural changes are planned, the HD reorganization reload utility uses the existing DMB of the data base being reorganized without change.

If any data bases are logically related to the data base being reorganized, the scan utility must be used as indicated by the data base prereorganization utility.

The general operation consists of reading an input record and generating a DL/I insert call (ISRT) to cause the segment to be inserted. The result is a data base organized in an efficient accessing format with all fragmented free space consolidated.

Input to the HD reorganization reload utility, consists of:

1. A DMB and PSB loaded from a 'DL/I user lib'. The DMB and PSB names are obtained by DL/I by expanding the DBD generation name obtained from the parameter statement.

2. A copy of the non-empty unloaded data base.

3. A control file, if logical relationships exist. This file has been previously created by the prereorganization utility.

Output from the HD reorganization reload utility consists of:

1. One of the following reorganized data bases:

   - HD randomized data base
   - HD indexed data base
   - HDAM data base
   - HIDAM data base with its newly created INDEX
   - HISAM data base
   - SHISAM data base

2. A work file, if logical relationships or secondary indexes exist. This file is used as input to the prefix resolution program.

3. Messages and statistics on the SYSLST device.

Subtopics:



---

## 9.3.1 Reload Output Statistics

The HD reorganization reload utility maintains the following statistics:

- Total number of segments (by segment type) in the data base
- Total number of segments in the data base.

The audit capability compares these calculated statistics against those returned by the HD reorganization unload program and marks any differences. If segment types are added to or deleted from the DBD, the result by segment type may not be equal. However, the total number of segments unloaded and reloaded should be equal. When the reload utility is restarted, the number of segments unloaded and reloaded will never be equal. In this case, the number of segments reloaded includes only those reloaded since the restart.

Figure 61 is an example of the messages and statistics obtained from the HD reorganization reload utility program. An explanation of all numbered messages may be found in the *VSE/SP Messages and Codes*.

---

```
     DL/I DOS/VS  DLZURGL0  -  HD DATA BASE REORGANIZATION RELOAD            DATE: 11/30/88
TIME: 13:39:42   PAGE    1


                                  SEGMENT LEVEL STATISTICS

                                                    TOTAL SEGMENTS BY SEGMENT TYPE
                      SEGMENT     SEGMENT
                       NAME        LEVEL           RELOADED     DIFFERENCE

                       J1              1                  3

                       J2              2                  3

                       J3              3                  1

                       J4              4                  1

                       J5              2                  4

                       J6              3                  4
```

```
                    J7              4                   2

                    J8              5                   1

                    J9              2                   3

                    J10             3                   3

                    J11             4                   0


                        TOTAL  SEGMENTS  IN  DATA  BASE
                   UNLOADED          RELOADED      DIFFERENCE

                           25              25


  DLZ339I     NO ERRORS DETECTED,  RELOAD SUCCESSFUL
```

---

Figure 61. Example of HD Reload Output Statistics

Following the messages, the heading "Segment Level Statistics" is printed and, under that, the following fields occur:

***Segment Name:*** The segment name to which this line of statistics applies.

***Segment Level:*** The hierarchical level of this segment in the data base. Note that the segments are printed in the same order they were described in the DBD and appeared in the HD unload statistics.

Under the heading "Total Segments by Segment Type" are the fields:

***Reloaded:*** The number of occurrences of this segment type in the reloaded data base.

***Difference:*** This field is blank if the reload count equals the unload count for this segment. A value greater than 0 indicates more records counted in reload; a value less than 0 indicates more records counted in unload.

Under the heading "Total Segments in Data Base" are the fields:

***Unloaded:*** The total of all segments in the data base counted by the HD reorganization unload utility.

***Reloaded:*** The total of all segments in the data base counted by this program.

***Difference:*** The difference, if any, between the previous two totals.

---

*© Copyright IBM Corp. 1981, 1989*

---

## 9.3.2 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS011,cuu<br>HDUNLD1<br><br>HDUNLD1<br>extent data | These statements define the input file containing the data base to be reloaded. This file was created by the HD reorganization unload utility and may be DASD, or tape with standard labels. HDUNLD1 is the symbolic name of the input file. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS013,cuu]<br>WORKFIL ]<br><br>WORKFIL ]<br>extend data ] | These statements define the output work file. The work file is required only if logical relationships exist. It is also required for secondary indexes when the indexes are being built at prefix resolution time. The work file may be DASD, or tape with standard labels. WORKFIL is the symbolic name. If there are no active logical relationships, the HD reorganization reload utility will open and close the output work file, WORKFIL. A warning message (DLZ388I) will be issued, indicating that no records were written to the output work file; the job will continue. |
| [// ASSGN<br>[// DLBL<br>[// EXTENT | SYS012,cuu ]<br>CONTROL ]<br>extent data ] | These statements define the input control file created by the data base prereorganization utility. It must be DASD. CONTROL is the symbolic name of the input file. These statements are not required if logical relationships do not exist. |
| // ASSGN<br><br><br>[ TLBL(Tape)<br>or<br>[ DLBL(DASD)<br>[ EXTENT | SYS010,{cuu}<br>{IGN}<br>{UA }<br>RSTFILE]<br><br>RSTFILE]<br>extent data] | These statements are used when doing a "reload restart." They define the partially created output work file (WORKFIL), if any, that was being created during the initial reload. This work file is now used as input during restart. Assign it as SYS010 with a filename of RSTFILE. If no work file was created during initial reload, SYS010 must be assigned IGN or UA when doing the restart. |
| // DLBL | filename | This statement defines the symbolic name of the data base file to be reloaded.<br>One statement must be present for each file that appears in the DBD describing this data base.<br>For HD randomized and indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements (if any).<br>For HDAM, HIDAM (and primary INDEX), and SHISAM data bases, use the |

| | | |
|---|---|---|
| | | filename specified in the DD1 operand of DATASET statements. For HISAM data bases, use the filenames specified in the DD1 and OVFLW operands of DATASET operands. If secondary index relationships exist, DLBL statements for the secondary index data bases are also required when the secondary indexes are to be created automatically. |
| // 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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

---



---

## 9.3.3 Parameter Statement Requirements

The HD reorganization reload utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column 1, must be entered via SYSIPT or SYSLOG:

```
ULU,DLZURGL0,dbname[,{buf}]([,HDBFR=][,HSBFR=][,TRACE=]
                      {1  }
```

**dbname**
is the name of the data base description that contains the data base to be reloaded.

The positional parameter "buf" and the keyword parameters HDBFR, HSBFR, and TRACE are optional parameters that may also be entered in the parameter statement. These parameters have the same usage as for application programs and are described in Chapter 15, "Executing DL/I Programs," under "DL/I Parameter Information Requirements."

 *© Copyright IBM Corp. 1981, 1989*

## 9.3.4 Control Statement Requirements

```
[BLDINDEX=NO]
```

This control statement indicates that secondary indexes are not to be created during reload. Instead, they will be created later using the logical relationship utilities. Use of this control statement is optional. If used, it must begin in column 1.

If the BLDINDEX=NO statement is not provided, the secondary indexes will automatically be created during reload, as long as DLBL statements are provided for them. Secondary indexes will not be created if no DLBL statements are provided.

```
[REW=option]
```

This control statement invokes tape rewind options. Its use is optional.

If used, columns 1-4 must contain REW= and column 5 must contain one of these letters to indicate the tape option you want:

N - means no rewind,
R - means rewind only, and
U - means rewind and unload.

No other columns are used. Put the completed statement after the DL/I parameter statement in your job stream.

If this control statement is not used, tapes will automatically rewind and unload at end-of-job.

**Note:** When reading multiple files from a single tape, use one of the following to avoid reading the wrong file:

1. Specify the N rewind option for all files on the tape; then use an MTC command to rewind the tape when all jobs reading from the tape are finished.

2. Specify the R or U rewind option for a file which has a file sequence number on its corresponding TLBL statement. The R or U rewind option causes the tape to rewind when a file on the tape is "opened" and then again when it is closed. If a file

```
sequence number is not specified in this case, the first file
on the tape may be read again by mistake.
```

---

---

# 9.3.5 Doing a Reload Restart

During HD reorganization unload, DL/I creates a checkpoint record each time it unloads the number of records specified in the CHKPT control statement. These checkpoint records can be used to restart the unload if it fails before completion. Reload restart permits restarting a HD reorganization reload from a checkpoint record (if the job fails to complete) without the need to rerun the entire job. Each VSAM data set referenced by a restarted HD reload must have the VSAM RECOVERY attribute. This attribute must be set in the file's catalog entry when the file is defined to VSAM.

During the reload, a message is issued each time a checkpoint record is encountered. This message is the same in content and format as that issued during unload when the checkpoint record was created, and will identify the checkpoint by number. When the reload is restarted, the operator is asked, by message DLZ318A ENTER RESTART NUMBER - nnnn, to provide the number of the checkpoint record from which to restart. The utility will locate the designated checkpoint record and begin processing from that point.

Note that reload restart should not be attempted unless messages DLZ001I and DLZ002I have been received.

If the reload utility fails and a restart is desired, the Access Method Services utility Verify job should be run for each file referenced at the time the reload failed. The reload job may then be resubmitted with the parameter card changed from ULU to ULR. This change identifies the job as being restarted. An additional ASSGN card and the necessary TLBL or DLBL and EXTENT cards are also required. The reload utility then requests that the operator supply the checkpoint identification number. This number should be the last checkpoint previously encountered. As such, it will be identifiable from console message DLZ381I. For a complete description of all reload restart messages, see *VSE/SP Messages and Codes*.

If a work file (for logical relationships or secondary indexes) was being created during the initial reload, the partially created work file should be submitted as input to the restarted job. It should be assigned as SYS010 with a filename of RSTFILE. The utility uses this file to create a complete work file on SYS013 as is normally done.

---

## 9.3.6 Examples

***Example 1:*** This example shows an HDAM reorganization reload operation.

---

```
// JOB    HDRELOAD1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN  SYS011,180
// TLBL   HDUNLD1,'hdamunload'
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   CONTROL,'control file',,VSAM,CAT='catalog'
// EXTENT SYS012,'volser'
// ASSGN  SYS013,181
// TLBL   WORKFIL,'hdam work file'
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGL0,DLIDBD4
/*
/&
```

---

***Example 2:*** This example shows a HIDAM reorganization reload operation. Note that in this example the HIDAM INDEX data base data set is also described.

---

```
// JOB    HDRELOAD2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN  SYS011,180
// TLBL   HDUNLD1,'hisamunload'
// ASSGN  SYS013,DISK,VOL='volser',SHR
// DLBL   WORKFIL,'hidam work file',,VSAM,RECORDS=5,RECSIZE=6454,*
                 DISP=(NEW,KEEP),CAT='catalog'
// EXTENT SYS013,'volser'
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   CONTROL,'control file',,VSAM,CAT='catalog'
// EXTENT SYS012,'volser'
// DLBL   HIDAMDB,'hidam.data.base',,VSAM,CAT='catalog'
// DLBL   HIDAMDX,'hidam.index',,VSAM,CAT='catalog'
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGL0,DLIDBD3
/*
/&
```

---

***Example 3:*** This example shows an HDAM reorganization reload operation similar to that shown in Example 1. In this case, however, secondary indexes exist. The BLDINDEX=NO control statement is specified to suppress creation of the secondary indexes.

---

```
// JOB    HDRELOAD3
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// DLBL   HDAMDX,'hdam.secondary.index',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   HDUNLD1,'hdamunload'
// ASSGN  SYS013,181
// TLBL   WORKFIL,'hdam work file'
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   CONTROL,'control file',,VSAM,CAT='catalog'
// EXTENT SYS012,'volser'
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGL0,DLIDBD2
BLDINDEX=NO
/*
/&
```

---

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 9.4 Chapter 20. HISAM Reorganization Unload Utility (DLZURUL0)

The HISAM reorganization unload utility program is primarily designed to provide an efficient means of reorganizing SHISAM, HISAM, and INDEX data bases. Additionally, the reorganized output can be used as input to the data base data set recovery utility.

If the output of the HISAM reorganization unload utility is to be used for recovery purposes, the user must reload the data base with the HISAM reorganization reload utility prior to applying changes to the data base. The reload is necessary so that log tapes created after this unload refer to the correct data (which may have moved as a result of the unload). (The HISAM reorganization reload utility program is discussed in Chapter 21, "HISAM Reorganization Reload Utility (DLZURRL0)").

Periodically, it is advantageous to reorganize HISAM, SHISAM, and INDEX data base structures to remove unused space generated as the result of segment deletions. A separate index reorganization can be done without the concurrent reorganization of the associated data base. Features of the program allow the data base logical record lengths and blocking factors to be changed to create a more efficient data base structure. This may be accomplished by executing the DBD generation procedure and link-editing a DMB, created by the application control blocks creation and maintenance utility, *prior to unloading* the data base.

The general operation consists of reading KSDS records, picking up any overflow dependent segments from the ESDS, dropping deleted root and dependent segments, and writing a sequential file arranged in a physically related sequence.

Note that two copies of the data base may be produced. This facility has the advantage that, should a permanent error occur on one output volume, the remaining one continues until normal end of job. The slight performance decrease is usually acceptable, particularly when compared against the requirements that a total rerun would involve.

Input to the HISAM reorganization unload utility, as illustrated in Figure 62 consists of:

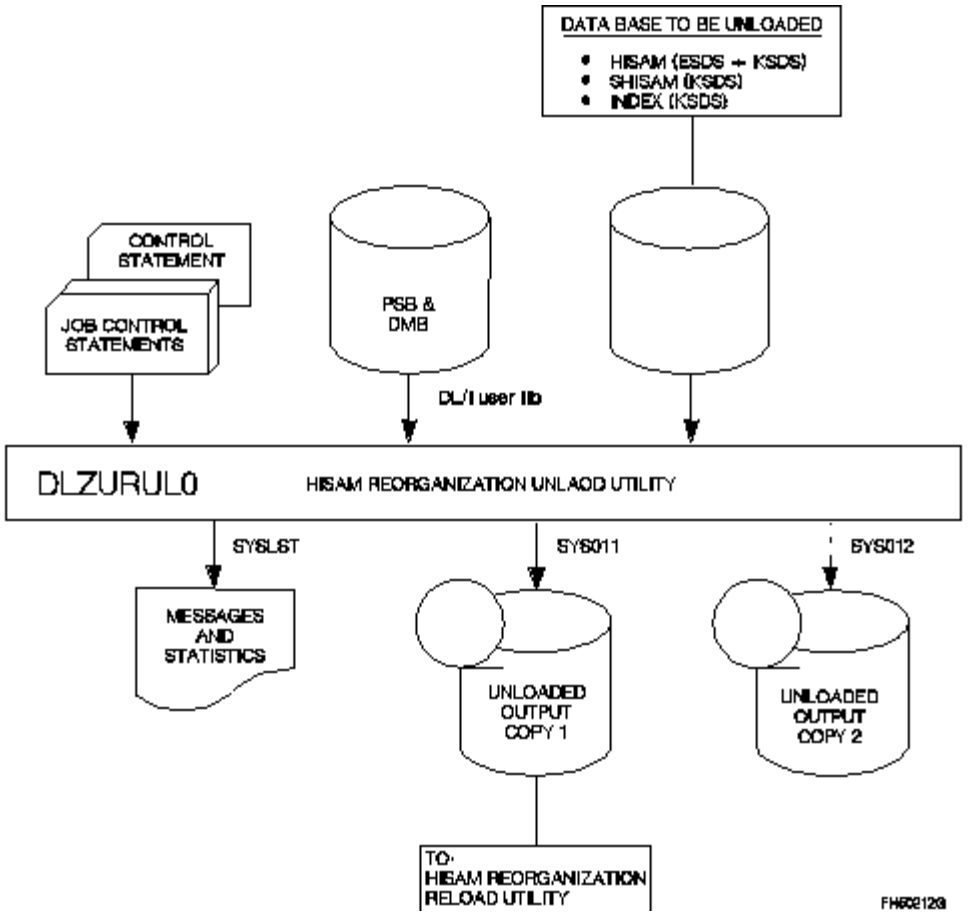



Figure 62. HISAM Reorganization Unload Utility

1. A control statement read from the SYSIPT device containing the DBD name of the SHISAM, HISAM, or INDEX data base to be reorganized, the KSDS filename, and output filename(s).

2. A DBD loaded from a 'DL/I user lib'.

3. The input KSDS and ESDS which constitute the HISAM data base (KSDS only if SHISAM or INDEX).

Output from the HISAM reorganization utility (Figure 62) consists of:

1. One or two copies of the reorganized data base. Tape or DASD may be used and multiple copies may be produced on mixed device types.

2. Messages and statistics on the SYSLST device.

Subtopics:

*© Copyright IBM Corp. 1981, 1989*

---

# 9.4.1 Unload Output Statistics

The HISAM reorganization unload utility provides statistics on data base content for the data base. In addition, it provides an audit trail capability.

Figure 63 is an example of the output messages and statistics obtained from the HISAM reorganization unload utility. Following the page heading are the various messages generated as a result of the options selected for this execution. An explanation of all numbered messages may be found in *VSE/SP Messages and Codes*. The message "COPY 1 FILENAME IS - filename1" shows the filename of the TLBL or DLBL statement for the primary output file and appears if the primary output file was successfully written. If a second copy was requested, and the second copy was successfully written, another message, "COPY 2 FILENAME IS - filename2" appears. The fields which are listed after the heading "Data Base Statistics" are described below.

---

```
     DL/I DOS/VS  DLZURUL0  -  HISAM DATA BASE REORGANIZATION UNLOAD            DATE: 11/30/88
TIME: 17:01:00   PAGE    1



                                        D A T A   B A S E   S T A T I S T I C S
        DATA BASE - DHISAM2

          KSDS NAME HISAMK2
          KSDS NAME HISAME2

                                    KSDS ROOTS                                                ESDS
DEPENDENTS
                           IN       OUT      DELETED                                        IN
OUT

                            9         9           0                                         22
22


         TOTAL NUMBER OF RECORDS OUT =     31

     COPY 1 FILENAME IS - DBOUT1

     COPY 2 FILENAME IS -


           DEPENDENT OVERFLOW CHAINS (#)                                              ROOTS WITHOUT
ESDS CHAINS (BYTES)
```

```
          NO.  LONGEST  SHORTEST  AVERAGE                                                    NO.  LONGEST
SHORTEST  AVERAGE
            5        6         2      4.40                                                     3      76
42     53



                                        S E G M E N T   L E V E L   S T A T I S T I C S

     MAXIMUM     AVERAGE     MAXIMUM     AVERAGE               SEGMENT   SEGMENT        TOTAL
SEGMENTS    AVERAGE COUNT PER
      TWINS       TWINS      CHILDREN    CHILDREN               NAME      LEVEL         BY SEGMENT
TYPE   DATA BASE RECORD

          1         1.00         75        33.00               A1111111      1
8              0.88

          3         1.25         12         4.00               AA222222      2
10             1.11

          3         1.00          9         3.00               AAA33333      3
10             1.11

          3         1.00          0         0.00               AAAA4444      4
10             1.11

          3         1.00          3         1.00               AAAB4444      4
10             1.11

          3         1.00          0         0.00               AAABA555      5
10             1.11

          4         1.75          0         0.00               AB222222      2
14             1.55


     TOTAL SEGMENTS IN DATA BASE =                   72          AVERAGE DATA BASE RECORD LENGTH =
1,951 BYTES
```

---

Figure 63. Example of HISAM Unload Output Statistics

***Data Base:*** Data base name.

***KSDS Name:*** Name of KSDS.

***ESDS Name:*** Name of ESDS (blank if SHISAM or INDEX).

***KSDS Roots:*** Statistics dealing with root records in the KSDS.

- IN--Number of old KSDS roots read
- OUT--Number of new KSDS roots written
- DELETED--Number of old KSDS roots deleted. If the number of roots in

and out contains one more than the user has inserted, this is the high-key record that terminates the KSDS.

***ESDS Dependents:*** Statistics dealing with records in the ESDS.

- IN--Number of old records in ESDS read
- OUT--Number of new records in ESDS written.

***Total Number of Records Out:*** Number of records, both KSDS root segments and ESDS segments, written out.

***Dependent Overflow Chains:*** Statistics dealing with root segments that had dependents in the ESDS.

- NO.--Number of root segments with ESDS records.
- LONGEST--Largest number of ESDS records chained off one root.
- SHORTEST--Smallest nonzero number of ESDS records chained off one root segment.
- AVERAGE--Average number of ESDS records chained off one root segment (of those root segments that had dependents).

***Roots Without ESDS Chains:*** Statistics dealing with root segments that had no dependents.

- NO.--Number of root segments without dependent chains.
- LONGEST--Longest data base record (in bytes) with no ESDS records.
- SHORTEST--Shortest data base record (in bytes) with no ESDS records.
- AVERAGE--Average data base record length (in bytes) of those root segments with no ESDS records.

After the heading "Segment Level Statistics" the following fields occur:

***Maximum Twins:*** This field contains the maximum number of segments of this type encountered under an immediate parent segment. At the root level, this value is 1.

***Average Twins:*** This field contains the average number of segments of this type encountered under an immediate parent segment. This value is calculated to two decimal places.

***Maximum Children:*** This field contains the maximum number of dependent segments (at all subordinate levels) under a given parent.

***Average Children:*** This field contains the average number of dependent

segments (at all subordinate levels) under a given parent. This value is calculated to two decimal places. Note that the lowest level segment in any hierarchical path has a value of zero in this field type.

***Segment Name:*** The segment name to which this line of statistics applies.

***Segment Level:*** The hierarchical level of this segment in the data base.

***Total Segments by Segment Type:*** This field contains the sum of the occurrences of this segment type in the entire data base. Note that the sum of the level 1 segment type reflects the total number of data base records (root segments) in the data base.

***Average Count per Data Base Record:*** This field contains the average number of occurrences of this segment type within a given data base record. The value is calculated to two decimal places.

***Total Segments in Data Base:***

***Average Data Base Record Length:*** The average length (in bytes) of the record.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 9.4.2 Job Control Statement Requirements

The HISAM reorganization unload utility is executed as a standard VSE application program and requires the following job control statements:

| | | |
|---|---|---|
| // DLBL | filename | This statement defines the symbolic name of the data base file to be unloaded. One statement must be present for each file that appears in the DBD describing this data base. For SHISAM data bases, use the filename specified in the DD1 operand of DATASET statements. For INDEX data bases, use the filename specified in either the DD1 operand of DATASET statements or the REF operand of ACCESS statements. For HISAM data bases, use the filenames specified in the DD1 and OVFLW operands of DATASET statements. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS011,cuu<br>filename<br><br>filename<br>extent data | These statements define the first copy of the reorganized output file. One is required for each data base (KSDS/ESDS pair) to be reorganized. The filename may be any name, but must appear in the associated utility control statement. The file may be on tape or DASD. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS012,cuu ]<br>filename ]<br><br>filename ]<br>extent data] | These statements are optional and are required only if the associated utility control statement requests two copies of the dump. The name must appear in the control statement. It may be tape or DASD. |
| // EXEC | DLZURUL0,<br>SIZE=xxxK | HISAM reorganization unload utility module. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

## 9.4.3 Control Statement Requirements

One control statement is required for the HISAM reorganization unload utility. Its form and content are described below. Input control statements are read from the SYSIPT device.

```
1 2 3 4-11     13-20    22-29     31-38       40-80
R x x dbname   infile   outfile1  [outfile2]  [comments]
```

| Column | Description |
|---|---|
| 1 | Must be R. |
| 2 | Must be a 1 or 2, depending on the number of copies required. |
| 3 | Invokes the tape rewind option:<br>N means no rewind,<br>R means rewind only, and<br>U (or blank) means rewind and unload.<br><br>**Note:** When writing multiple files on a single tape, use one of the following to avoid overwriting a file:<br><br>1. Specify the N rewind option for all the files on the tape; then use an MTC command to rewind the tape when all jobs writing on the tape are finished.<br><br>2. Specify the R or U rewind option for a file that has a file sequence number on its corresponding TLBL statement. The R or U rewind option causes the tape to rewind when a file on the tape is *opened* and then again when it is closed. If a file sequence number is not specified in this case, the first file on the tape may be overwritten. |
| 4-11 | Must be the name of the DBD that includes the symbolic filenames of the KSDS/ESDS pair (only KSDS if HISAM or INDEX) to be reorganized. |
| 13-20 | Must be the symbolic filename of the KSDS of the SHISAM, HISAM or INDEX data base to be reorganized. It must appear as the DD1 parameter of the DATASET statement or the REF parameter of ACCESS statements in the referenced DBD, and corresponding DOS/VSE DLBL statements must be provided. |
| 22-29 | Filename of the TLBL or DLBL statement for the primary output file (SYS011). |
| 31-38 | Filename of the TLBL or DLBL statement for the second copy of the reorganized output file (SYS012). This field must be blank if card column 2 contains a 1. |

| | |
|---|---|
| 40-80 | Comments. |

## 9.4.4 Examples

***Example 1:*** This example unloads a HISAM data base, creating one output copy. The KSDS input filename is HISDB1, and the ESDS is HISDB2. These names appear in the DBD named DLIDBD1. The output filename is DBOUT1.

```
// JOB    HISUNLD1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HISDB1,'hisam.data.base.ksds',,VSAM,CAT='catalog'
// DLBL   HISDB2,'hisam.data.base.esds',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   DBOUT1,'hsunld'
// EXEC   DLZURUL0,SIZE=100K
R1 DLIDBD1  HISDB1   DBOUT1
/*
/&
```

***Example 2:*** This example unloads SHISAM or INDEX data bases which consists of only a KSDS. Two copies of the data base are created.

```
// JOB    SHISUNL2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   DB,'data.base.ksds',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   DBOUT1,'shisunld1'
// ASSGN  SYS012,181
// TLBL   DBOUT2,'shisunld2'
// EXEC   DLZURUL0,SIZE=100K
R2 DLIDBD2  DB       DBOUT1   DBOUT2
/*
/&
```

# 9.5 Chapter 21. HISAM Reorganization Reload Utility (DLZURRL0)

The HISAM reorganization reload utility is a means of reloading SHISAM, HISAM, or INDEX data base which have been unloaded and reorganized by the HISAM reorganization unload utility. The general operation consists of reading an input file created by the HISAM reorganization unload utility and writing the records to the appropriate KSDS or ESDS.

Prior to executing the HISAM reorganization reload utility, the user must execute the VSAM Access Method Services utility command DELETE to remove the name of the file from the VSAM catalog and release the space allocated for it. He must then define the new file to VSAM to cause its definition to be recorded on the VSAM catalog.

Input to the HISAM reorganization reload utility, as illustrated in Figure 64, consists of:

1. A control statement read from the SYSIPT device containing the symbolic filename of the input file.

2. The input file containing the unloaded SHISAM, HISAM, or INDEX data base.

Output from the HISAM reorganization reload utility, Figure 65, includes:

1. The reloaded KSDS and ESDS which constitute the HISAM data base (KSDS only if SHISAM or INDEX).

2. Messages and statistics on the SYSLST device.

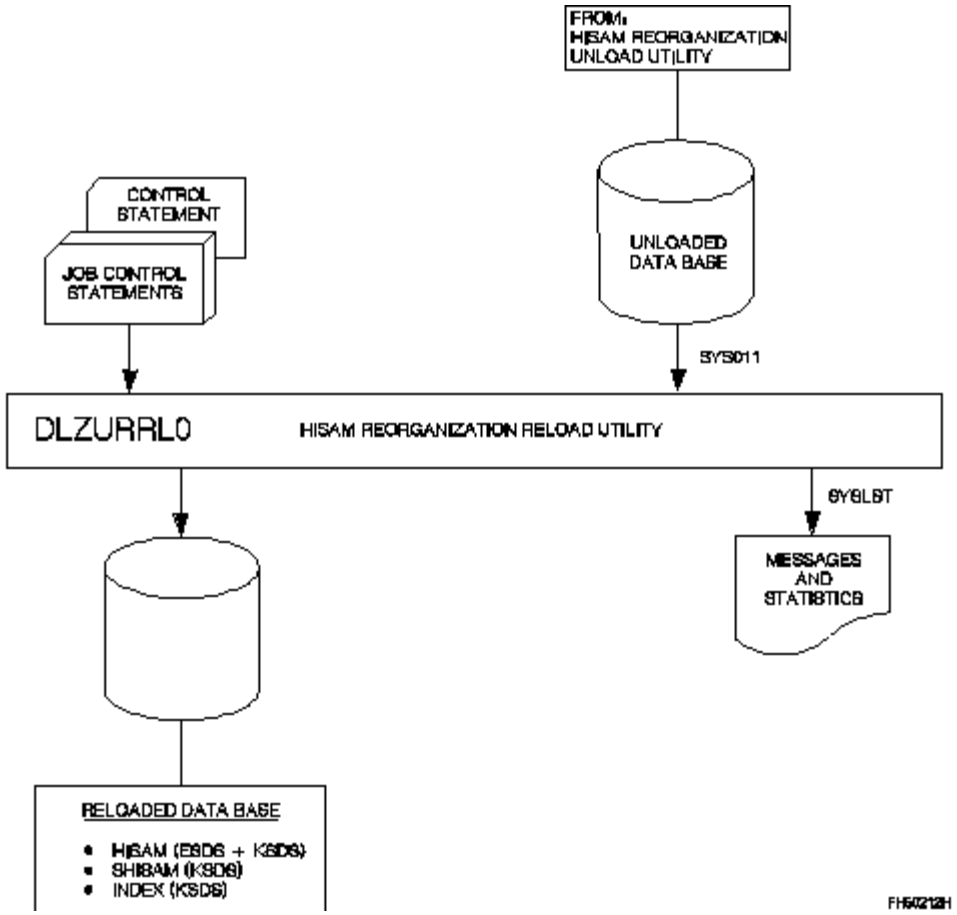



Figure 64. HISAM Reorganization Reload Utility

Subtopics:

- 9.5.1 Reload Output Statistics
- 9.5.2 Job Control Statement Requirements
- 9.5.3 Control Statement Requirements
- 9.5.4 Examples

---

---

# 9.5.1 Reload Output Statistics

The HISAM reorganization reload utility provides statistics and audit trail for the data base reloaded. Figure 65 is an example of the output messages and statistics obtained. Following the page heading are the various messages generated if errors occurred for this execution. An explanation of all numbered messages may be found in *VSE/SP Messages and Codes*.

---

```
     DL/I DOS/VS  DLZURRL0  -  HISAM DATA BASE REORGANIZATION RELOAD            DATE: 11/30/88
TIME: 17:11:32   PAGE    1



                                        D A T A   B A S E   S T A T I S T I C S
     DATA BASE - DHISAM2D

        KSDS NAME HISAMK2
        KSDS NAME HISAME2

                                                 DEPENDENT OVERFLOW CHAINS (#)        ROOTS WITHOUT
ESDS CHAINS(BYTES)
            KSDS ROOTS    ESDS DEPENDENTS        NO.  LONGEST  SHORTEST  AVERAGE       NO.  LONGEST
SHORTEST  AVERAGE
                     9                 22          5        6         2      4.4         3       76
42       53


                                    S E G M E N T   L E V E L   S T A T I S T I C S


                          SEGMENT       SEGMENT        TOTAL SEGMENTS BY SEGMENT TYPE
                           NAME          LEVEL           RELOADED     DIFFERENCE

                          A1111111         1                  8

                          AA222222         2                 10

                          AAA33333         3                 10

                          AAAA4444         4                 10

                          AAAB4444         4                 10
```

```
                    AAABA555         5                   10

                    AB222222         2                   14

                    AB333333         6                    0


                                  TOTAL SEGMENTS IN DATA BASE


                                 UNLOADED           RELOADED           DIFFERENCE

                                     72                 72
```

---

Figure 65. Example of HISAM Reload Output Statistics

The fields which occur after the heading "Data Base Statistics" are:

***Data Base:*** Data base name.

***KSDS Name:*** Name of KSDS.

***ESDS Name:*** Name of ESDS (blank if SHISAM or INDEX).

***KSDS Roots:*** Number of root segments loaded onto the KSDS.

***ESDS Dependents:*** Number of overflow records loaded onto the ESDS.

***Dependent Overflow Chains:*** Statistics dealing with roots with dependents chained into the ESDS.

- NO.--Number of KSDS root segments with ESDS records.
- LONGEST--Largest number of ESDS records chained to one KSDS root segment.
- SHORTEST--Smallest nonzero number of ESDS records chained to one KSDS root segment.
- AVERAGE--Average number of ESDS records chained to KSDS root segments (of those with chains).

***Roots without ESDS Chains:*** Statistics dealing with KSDS root segments that have no overflow dependent records in the ESDS.

- NO.--Number of root records with no ESDS records.
- LONGEST--Longest root record (in bytes) with no ESDS records.

- SHORTEST--Shortest root record (in bytes) with no ESDS records.

- AVERAGE--Average length of root records (in bytes) with no ESDS records.

After the heading "Segment Level Statistics" the following fields occur:

***Segment Name:*** The segment name to which this line of statistics applies.

***Segment Level:*** The hierarchical level of this segment in the data base.

***Total Segments by Segment Type:***

- RELOADED--The total number of segments of this type reloaded.

- DIFFERENCE--This field is blank if the counts by reload and unload are equal. If they are not equal, a value greater than 0 indicates more records counted in reload; a value less than 0 indicates more records counted in unload.

The following fields occur under the heading "Total Segments in Data Base":

***Unloaded:*** The total number of segments unloaded by the HISAM reorganization unload utility.

***Reloaded:*** The total number of segments reloaded by this program.

***Difference:*** The difference, if any, between the previous two totals.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 9.5.2 Job Control Statement Requirements

The HISAM reorganization reload utility is executed as an application program and requires the following job control statements:

| | | |
|---|---|---|
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS011,cuu<br>filename<br><br>filename<br>extent data | These statements define the input file that was created by the HISAM reorganization unload utility. The filename parameter may be any name but must appear in the associated utility control statement. |
| // DLBL | filename | This statement defines the symbolic name of the data base file to be reloaded.<br><br>One statement must be present for each file that appears in the DBD describing the data base. (If multiple data bases are to be reloaded during a single job execution, all files must be defined.)<br><br>For SHISAM data bases, use the filename specified in the DD1 operand of DATASET statements.<br><br>For INDEX data bases, use the filename specified in either the DD1 operand of DATASET/DATASET statements or the REF operand of ACCESS statements.<br><br>For HISAM data bases, use the filenames specified in the DD1 and OVFLW operands of DATASET statements. |
| // EXEC | DLZURRL0,<br>SIZE=xxxK | HISAM reorganization reload utility module. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

## 9.5.3 Control Statement Requirements

One control statement is required for the HISAM reorganization reload utility. Its form and content are described below. Input control statements are read from the SYSIPT device.

```
1 3                    22-28          40-80
L n                    filename       [comments]
```

| Column | Description |
|---|---|
| 1 | Must be L. |
| 3 | Invokes the tape rewind option:<br>N means no rewind,<br>R means rewind only, and<br>U (or blank) means rewind and unload.<br><br>**Note:** When reading multiple files from a single tape, use one of the following to avoid reading the wrong file:<br><br>1. Specify the N rewind option for all files on the tape; then use an MTC command to rewind the tape when all jobs reading from the tape are finished.<br><br>2. Specify the R or U rewind option for a file that has a file sequence number on its corresponding TLBL statement. The R or U rewind option causes the tape to rewind when a file on the tape is "opened" and then again when it is closed. If a file sequence number is not specified in this case, the first file on the tape may be read again by mistake. |
| 22-28 | Filename of the TLBL or DLBL statement for the input file (SYS011) that contains the data base to be reloaded. |
| 40-80 | Comments. |

 *© Copyright IBM Corp. 1981, 1989*

## 9.5.4 Examples

***Example 1:*** This example reloads a HISAM data base consisting of a KSDS and an ESDS.

```
// JOB    HISRELD1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HISDB1,'hisam.data.base.ksds',,VSAM,CAT='catalog'
// DLBL   HISDB2,'hisam.data.base.esds',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   DBRLDIN,'hisunld'
// EXEC   DLZURRL0,SIZE=100K
L                  DBRLDIN
/*
/&
```

***Example 2:*** This example reloads SHISAM or INDEX data bases.

```
// JOB    HISRELD2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   DB,'data.base.ksds',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   HISAMIN,'hisunld'
// EXEC   DLZURRL0,SIZE=100K
L                  HISAMIN
/*
/&
```

 *© Copyright IBM Corp. 1981, 1989*

# 9.6 Chapter 22. Data Base Prereorganization Utility (DLZURPR0)

The data base prereorganization utility program controls the execution of the other logical relationship resolution utilities. If the data bases being loaded and/or reorganized are known to contain no logical relationships, this program need not be executed.

The program may optionally be used with the initial load or the reorganization of data bases for which secondary indexes exist, in which case its output controls proper execution of the prefix resolution and prefix update programs.

Input to the prereorganization utility consists of one or more control statements that name the data base(s) being loaded and/or reorganized. Output consists of messages and a control file that is used by the data base scan utility, the data base prefix resolution utility, and the data base work file generator module during HD reload or initial load. An optional punched deck of scan control cards, for use with the data base scan utility, can also be provided.

The messages issued by this program indicate the data base operations that must be performed prior to execution of the prefix resolution and prefix update utilities. Required data base operations are indicated by output messages as follows: data bases listed after the characters "DBIL=" must be initially loaded; data bases listed after the characters "DBR=" must be reorganized using the HD reorganization utilities; data bases listed after the characters "DBS=" must be scanned using the scan utility.

If no errors are detected by this program, and no data bases need to be scanned, then the only messages provided are a listing of input control statements and a termination message.

The DBDs which are used for the data bases named on the input control statements must define each data base as it is to exist after logical relationships are resolved. The information in these DBDs that concern logical relationships must not be modified until the prefix update utility has successfully completed its execution.

The general operation of this program consists of an examination of the DL/I control blocks for each data base named on an input control statement. For each data base and segment that participates in secondary index or logical relationships, a control file list entry is generated that describes the prefix fields to be resolved, the work file records that must be generated, and their format options, etc. The programs using this information are controlled by the contents of each list entry.

Input to the data base prereorganization utility, as illustrated in Figure 66, consists of:

1. One or more control statements read from the SYSIPT device. They contain the names of the data bases that are being either initially loaded or reorganized.

2. One or more DMBs and PSBs loaded from a 'DL/I user lib'. These names are obtained by expanding the DBD name(s) specified in the control statement(s).

Output from the data base prereorganization utility

1. The output control file to be used by the prefix resolution utility, the data base work file generator module during HD reload or initial load, and the data base scan utility.

2. Data base scan control statements on SYSPCH, to be used as input to the data base scan utility program (optional).

3. Messages and statistics on the SYSLST device.

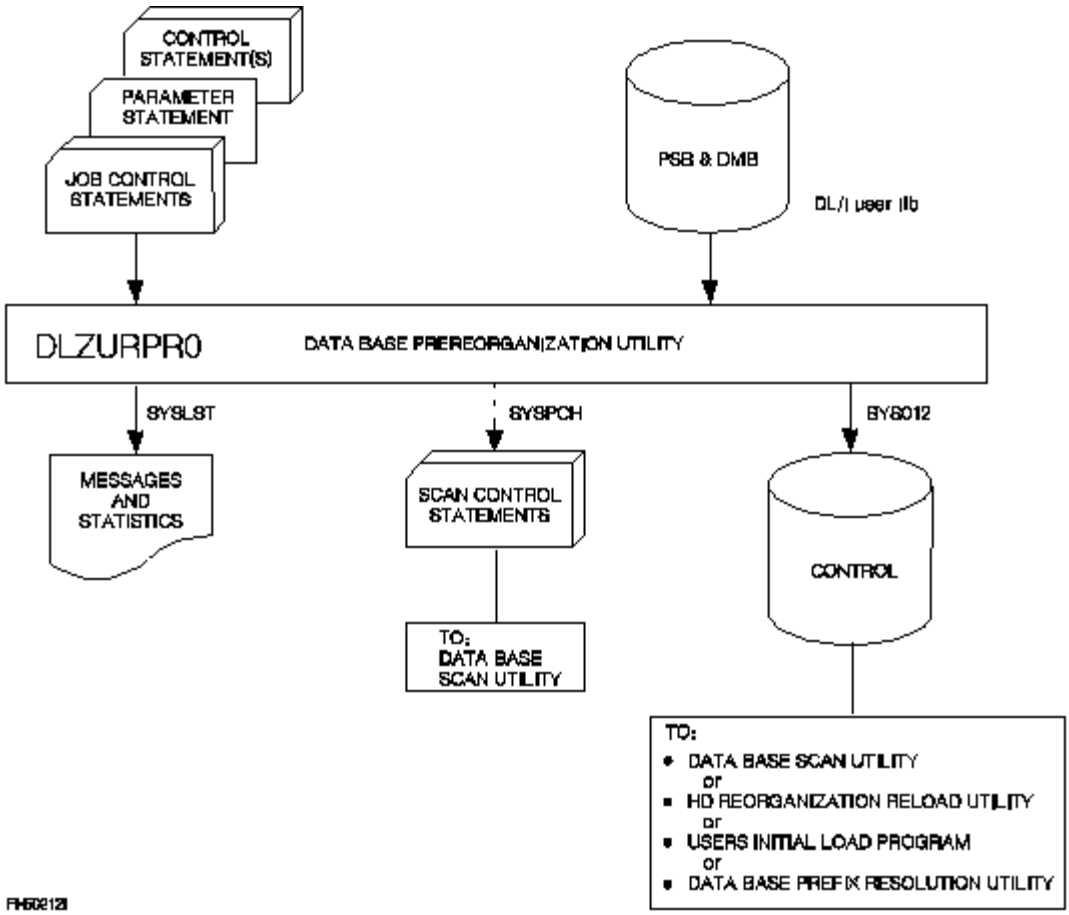



Figure 66. Data Base Prereorganization Utility

Subtopics:

- 9.6.1 Job Control Statement Requirements
- 9.6.2 Parameter Statement Requirements
- 9.6.3 Control Statement Requirements

9.6.4 Example

---



---

# 9.6.1 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // ASSGN<br>// DLBL<br>// EXTENT | SYS012,cuu<br>CONTROL<br>extent data | These statements define the output control file. This file is used as input to the prefix resolution utility, work file generator module, and data base scan utility. It must be DASD. CONTROL is the symbolic name of the output file. |
| [// ASSGN | SYSPCH,cuu] | This statement is optional. SYSPCH is required only if an OPTIONS=PUNCH control statement is provided (see below). A data base scan list is written on SYSPCH. |
| // EXEC | DLZRRC00,<br>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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

## 9.6.2 Parameter Statement Requirements

The data base prereorganization utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column 1, must be entered either via SYSIPT or SYSLOG:

```
ULU,DLZURPR0
```

# 9.6.3 Control Statement Requirements

```
[DBIL=databasename1,databasename2,...]
```

The DBIL control statement names data bases that are being *initially loaded.* One or more of these control statements may be provided. Each data base name must be right-padded with necessary blanks (that is, left-justified) to provide a total length of eight characters. A blank must follow the last entry on each control statement. If a HIDAM or HD indexed data base is to be initially loaded, only its DBD name should be listed on a DBIL control statement and not the INDEX DBD name.

```
[DBR=databasename1,databasename2,...]
```

The DBR control statement names data bases that are being *reorganized.* One or more of these control statements may be provided. Each data base name must be right-padded with necessary blanks (that is, left-justified) to provide a total length of eight characters. A blank must follow the last entry on each control statement. Note that if a HIDAM or HD indexed data base is to be reorganized, only its DBD name should be listed on a DBR control statement and not the INDEX DBD name. Likewise, secondary indexes must not be listed.

```
[OPTIONS=({NOPUNCH}[,STAT][,SUMM])]
          {PUNCH  }
```

The OPTIONS control statement indicates whether any optional information is to be provided during the reorganization of the data base. Information specified on this statement affects output in the execution of pre-reorganization utility, the prefix resolution utility, and the prefix update utility. The parentheses are required even if only one option is specified.

**PUNCH**
Specifying this option will cause the data base scan list to be written to the file defined by the SYSPCH ASSGN statement. Refer to the description of the data base scan utility for use of the SYSPCH output. If this option is not specified, the default is the NOPUNCH option.

**STAT**

Specifying this option causes the data base prefix resolution utility to print statistics of segments that are updated.

**SUMM**

Specifying this option causes the data base prefix resolution utility to print counts of logical relationships rather than to print individual warning messages DLZ978I for each logical parent not connected with a logical child.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 9.6.4 Example

```
In this example two data bases are to be initially loaded.  The DBD names
for the two data bases are PAYRLDB and SKILINV.
```

```
// JOB    PREREORG
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   CONTROL,'control file',,VSAM,RECORDS=1,RECSIZE=1600,*
                 DISP=(NEW,KEEP),CAT='catalog'
// EXTENT SYS012,'volser'
// EXEC   DLZRRC00,SIZE=250K
ULU,DLZURPR0
DBIL=PAYRLDB ,SKILINV
/*
/&
```

 *© Copyright IBM Corp. 1981, 1989*

# 9.7 Chapter 23. Data Base Scan Utility (DLZURGS0)

The data base scan utility is designed to search one or more data bases for all segments that are involved in logical relationships. For each such segment, the program writes one or more output records, depending upon the relationships in which the segment is involved, onto a work file that is one of the inputs to the prefix resolution utility.

The data base is scanned by issuing GN calls qualified by segment name. The work file generator module (DLZDSEH0) is used to create the work file output.

Checkpoint and restart capabilities are provided in this utility if they are requested. This is useful if many large data bases are to be scanned during one execution of the program. The checkpoint option is requested by including a CHKPT control statement as described below.

If execution of this program is abnormally terminated for any reason other than a data base I/O error, and checkpoints have been requested, program execution may be resumed by requesting a restart action. To request a restart, provide a RSTRT control statement containing the appropriate information as described below.

If execution of the program is abnormally terminated because of a data base I/O error, the cause of the I/O error should be determined and rectified. The data base must then be restored to its condition before execution of the program, and the program should then be restarted as described above.

Input to the data base scan utility, as illustrated in Figure 66, consists of:

1. One or more control statements read from the SYSIPT device containing the data base scan control statements.

2. The input control file created by the data base prereorganization utility.

3. One or more data bases to be scanned: valid data bases are HD randomized, HDAM, HD indexed, and HIDAM. If the data base being scanned contains logical relationships or indexes, the related data base(s) must also be online. Secondary index data bases are not accessed by this utility, but job control language statements for them are required.

4. If a restart is desired, a copy of the partially created work file from the previously interrupted utility program execution must be included. The number of the last successful checkpoint record (written to SYSLOG) is entered from the SYSIPT device.

Output from the data base scan utility, as illustrated in Figure 67,

consists of:

1. An output work file to be used as input to the prefix resolution utility.

2. Checkpoint data written to SYSLOG (if requested).

3. Messages and statistics on the SYSLST device.

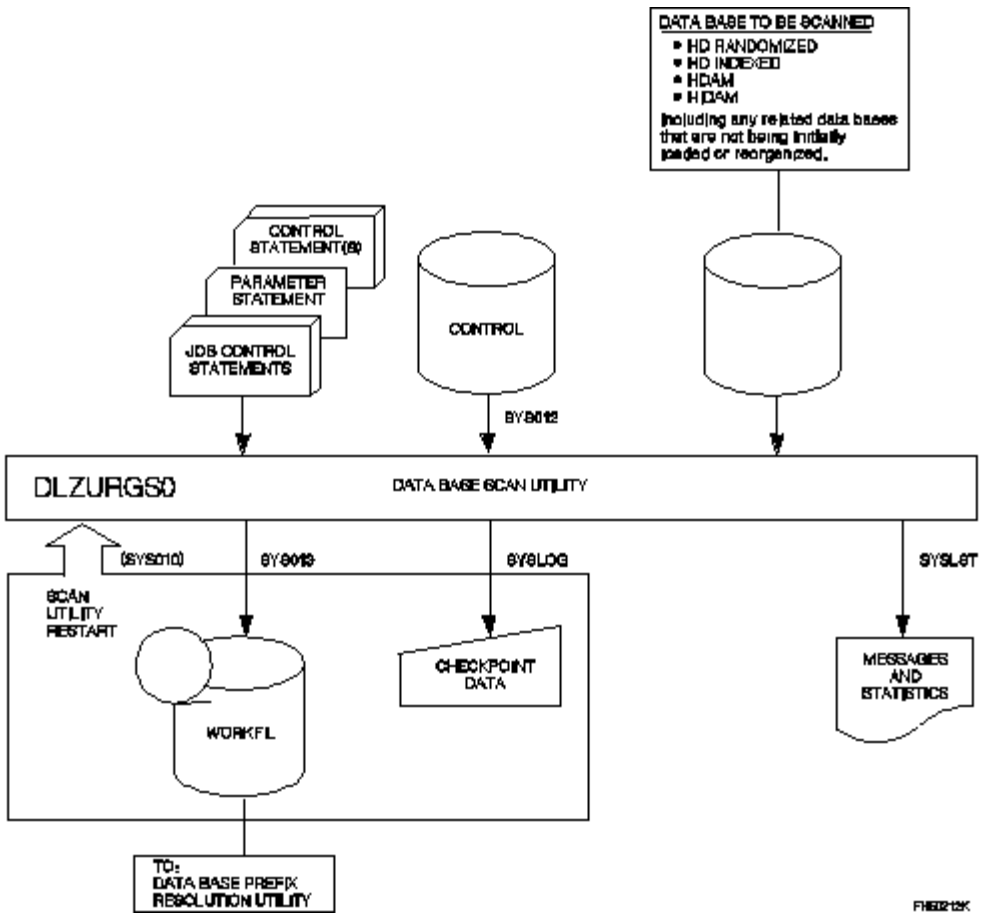



Figure 67. Data Base Scan Utility

Subtopics:

- 9.7.1 Job Control Statement Requirements
- 9.7.2 Parameter Statement Requirements
- 9.7.3 Control Statement Requirements
- 9.7.4 Example

---

## 9.7.1 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // DLBL | filename | This statement defines the symbolic name of the data base file to be scanned.<br><br>One statement must be present for each file that appears in the DBD describing the data base. One statement must also be present for each file that appears in DBDs of all other data bases that are related by logical relationships or indexes (primary or secondary).<br><br>For HD randomized and indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements (if any).<br><br>For HDAM, HIDAM, and INDEX (primary or secondary) data bases, use the filename specified in the DD1 operand of DATASET statements. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS010,cuu ]<br>RESTART ]<br><br>RESTART ]<br>extent data ] | These statements are optional and are required only if a utility restart is to be attempted. The file which defines the input work file contains the partially created work file and is the same file as the WORKFIL from the previously interrupted scan utility. The symbolic name of the input file is RESTART. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS013,cuu<br>WORKFIL<br><br>WORKFIL<br>extent data | These statements define the output work file. This file is used as input to the prefix resolution utility. It may be DASD, or tape with standard labels. WORKFIL is the symbolic name of the output file. |
| // ASSGN<br>// DLBL<br>// EXTENT | SYS012,cuu<br>CONTROL<br>extent data | These statements define the input control file created by the data base prereorganization utility. It must be DASD. CONTROL is the symbolic name of the input file. |
| // EXEC | DLZRRC00,<br>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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

---

---

## 9.7.2 Parameter Statement Requirements

The data base scan utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column 1, must be entered either via SYSIPT or SYSLOG:

```
ULU,DLZURGS0
```

## 9.7.3 Control Statement Requirements

```
DBS=dbdname,segmentname
```

The DBS control statement names a data base segment that is to be scanned. One or more of these control statements may be provided. The dbdname and segmentname must be padded with blanks to provide a total length of eight characters each. Any other data may appear on each statement. If no DBS control statements are provided to this program, the control file is used. The SYSPCH output of the prereorganization utility may be used as input to this program. The value of using the SYSPCH output as input is that, if multiple data bases need to be scanned, the SYSPCH output may be separated by data base name. Then the set of scan control statements for each data base may be submitted to different executions of the data base scan utility.

Note that if any DBS control statements are provided to this program, the entire scan list contained on the control file is ignored, and work file records are generated only for those segments named on DBS control statements. Even if a scan list is provided through DBS control statements, a control file must be provided to this program as the control file contains information (other than the scan list) that the scan utility requires. The user must ensure that all data bases indicated on the scan list provided by the data base prereorganization utility are scanned before attempting to execute the prefix resolution utility.

```
CHKPT={NO   }
      {nnnnn}
```

The CHKPT control statement indicates whether checkpoints are to be taken during operation of this program. A checkpoint record is written onto the work filenamed WORKFIL every time the number of records specified by "nnnnn" is generated. The parameter "nnnnn" must be a five-digit decimal number. As each checkpoint record is generated, a checkpoint message is issued to the SYSLOG device. The message indicates the name of the program, and the checkpoint number of the checkpoint record written. The checkpoint messages should be saved in case a restart operation is required. If this statement is omitted or CHKPT=NO is specified, no checkpoints will be taken.

```
RSTRT={NO   }
      {nnnnn}
```

The RSTRT control statement indicates whether a restart operation is to be performed by the program. If this statement is omitted or RSTRT=NO is specified, no restart operations occur. The parameter nnnnn is a five-digit decimal number and its value may be obtained from the checkpoint messages that were written to the SYSLOG device. The restart module reads records on the RESTART file until the checkpoint record numbered nnnnn is encountered. After each record is read, it is written onto the output WORKFIL file. When the correct checkpoint record is located, a message is written onto the SYSLOG device indicating the program name, and the checkpoint number, and the checkpoint record will be written onto the output WORKFIL file. Processing continues from that restart point. The volume containing the specified checkpoint record, and any succeeding volumes that were written during a previous execution of this program, must not be supplied to the prefix resolution utility.

### ABEND

The ABEND control statement indicates whether the utility will produce a dump on program termination in case of an abnormal condition. This ABEND card is mainly designed for test purpose and should be used with caution.

---

---

## 9.7.4 Example

```
This example shows the scan of an HDAM data base which is logically
related to another data base which the user indicated in either DBIL or
DBR control statements supplied to the data base prereorganization
utility.  The second data base is HIDAM and therefore the index is also
assigned.
```

```
// JOB    SCAN
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// DLBL   HIDAMDB,'hidam.data.base',,VSAM,CAT='catalog'
// DLBL   HIDAMDX,'hidam.index',,VSAM,CAT='catalog'
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   CONTROL,'control file',,VSAM,DISP=(OLD,KEEP),CAT='catalog'
// EXTENT SYS012,'volser'
// ASSGN  SYS013,181
// TLBL   WORKFIL,'scan work file'
// EXEC   DLZRRC00,SIZE=300K
ULU,DLZURGS0
DBS=PAYRLDB ,SKILINV
/*
/&
```

 *© Copyright IBM Corp. 1981, 1989*

# 9.8 Chapter 24. Data Base Prefix Resolution Utility (DLZURG10)

The data base prefix resolution utility is designed to accumulate the information generated on work files during the loading and/or reorganization of one or more data bases. It produces one or more output files that contain the prefix information needed to complete the logical relationships defined for the data base(s).

The general operation of the program consists of combining and sorting all work files which are defined as input to the program.

The input work files consist of records that describe the use of each segment of a data base in each of its logical or secondary index relationships. For example, there may be one record describing the use of a segment as a logical parent of logical child type A, another record describing its use as a logical parent of logical child type B, and so on. Based upon this input data, the prefix resolution utility determines the actual values that must be placed in the various prefix fields of each segment to complete all logical and secondary index relationships defined for the data base(s) being loaded or reorganized. The output work files produced by this program contain the data that will be placed in each segment prefix. The work files are used as input to the prefix update utility.

If no errors are detected by the program, the only output message issued is a normal program termination message. If segment prefix updates are found to be required, records are written to the output work file. If statistics were requested in a prereorganization utility control statement, they are written on the SYSLST device.

The DOS/VS Sort/Merge program is then automatically loaded. Since DL/I makes no assumption of sequence for unkeyed segments during initial data base loading and since the DOS/VS Sort/Merge does not guarantee first-in/first-out sequence of records with equal key values, the sequence of logical child segments of this type may be inconsistent.

Input to the data base prefix resolution utility as illustrated in Figure 68 consists of:

1. The control file created by the data base prereorganization utilities.

2. All work files generated by data base initial load, reload, and scan utilities.

3. One control statement read from the SYSIPT device.

Output from the data base prefix resolution utility as illustrated in Figure 68 consists of:

1. A work file that contains all the sorted input work file records for logical relationships and/or a similar file for secondary index relationships.

2. Statistics on the SYSLST device.

3. Messages on SYSLST and SYSLOG.

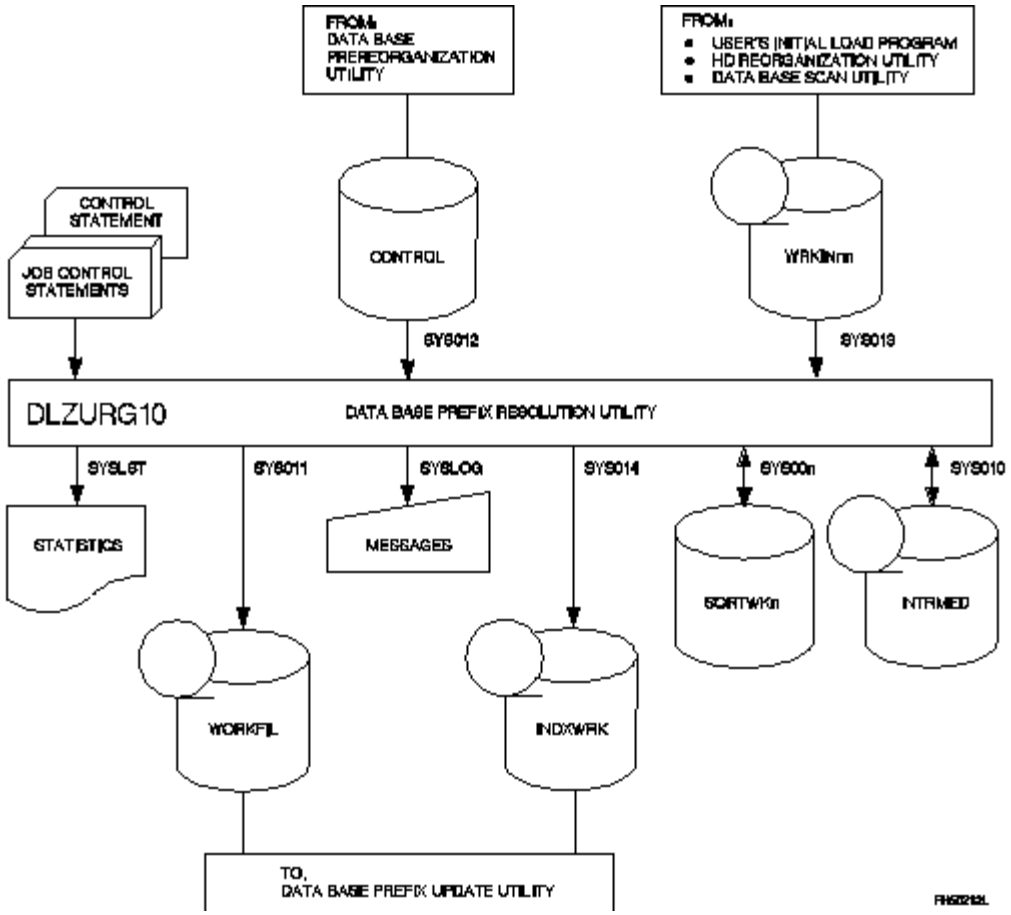



Figure 68. Data Base Prefix Resolution Utility

Subtopics:

- 9.8.1 Job Control Statement Requirements
- 9.8.2 Control Statement Requirements
- 9.8.3 Example

---

## 9.8.1 Job Control Statement Requirements

The data base prefix resolution utility is executed as a VSE application program and requires the following job control statements:

| | | |
|---|---|---|
| // ASSGN<br>// DLBL<br>// EXTENT | SYS00n,cuu<br>SORTWKn<br>extent data | These statements define the intermediate storage files for the sort/merge. Refer to *DOS/VS Sort/Merge Reference* regarding specification of number and size of intermediate storage files. Sort work files are numbered consecutively from SYS001 to SYS008. Filename assignments are from SORTWK1 to SORTWK8. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS010,cuu<br>INTRMED<br><br>INTRMED<br>extent data | These statements define the intermediate sort work file. It may be tape or DASD.<br><br>INTRMED is the symbolic name of the intermediate work file. This file is used to hold data between executions of the SORT utility by prefix resolution.<br><br>**Note:** If message 4450A (NO MORE AVAILABLE EXTENTS) occurs, and if SYSLOG is assigned to a keyboard, ensure that the system operator does not type in new extent limits for INTRMED work file. To do so results in the loss of data because, due to the way DOS/VS works, data written on the extended part in the first sort is not read when INTRMED is used as input to the second sort. If message 4450A occurs, rerun the prefix resolution utility with a larger INTRMED sort work file. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS013,cuu<br>WRKINnn<br><br>WRKINnn<br>extent data | These statements define the input work files generated by data base initial load, data base reload, and data base scan utilities. They may be DASD, or tape with standard labels. WRKINnn is the symbolic name of the input file(s), where nn is a numeric value from 01 to 99. Work files are numbered consecutively from WRKIN01 to WRKIN99. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS011,cuu ]<br>WORKFIL ]<br><br>WORKFIL ]<br>extent data ] | These statements define the sorted output work file for logical relationships. The file is used as input to the data base prefix update utility. It may be DASD, or tape with standard labels. WORKFIL is the symbolic name of the output |

| | | |
|---|---|---|
| | | file. These statements must be provided if the utility control statement indicates L or blank in column 10, that is, the input work files contain logical relationship information. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS014,cuu ]<br>INDXWRK ]<br><br>INDXWRK ]<br>extent data ] | These statements define the sorted output work file for secondary index relationships. The file will be used as input to the data base prefix update utility. It may be DASD or tape with standard labels. INDXWRK is the symbolic name of the output file. These statements must be provided if the utility control statement indicates I or blank in column 10, that is, the input work files contain index relationship information. |
| // ASSGN<br>// DLBL<br>// EXTENT | SYS012,cuu<br>CONTROL<br>extent data | These statements define the input control file created by the data base prereorganization utility. It must be DASD. CONTROL is the symbolic name of the input file (DTF). |
| // EXEC | DLZURG10,<br>SIZE=xxxK | Data base prefix resolution utility module. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

---

# 9.8.2 Control Statement Requirements

```
1         10  15          21-22        25-28

R         {L} number of   number of    [DUMP]
          {I} sort work   input files
          {} files
```

| Column | Description |
|---|---|
| 1 | Must be R. |
| 10 | Must be either L, I, or blank ().<br><br>L specifies that the input files contain only logical relationships to be resolved.<br><br>I specifies that the input files contain only secondary index relationships to be resolved.<br><br>specifies that both logical and secondary index relationships are to be resolved. |
| 15 | Must be a numeric value and specify the number of work files supplied to the Sort/Merge program. The number of files can be 1 through 8. |
| 21-22 | Must be a numeric value from 01 to 99 or blank and specify the number of input work files supplied to the utility. Blank defaults to 01. |
| 25-28 | The DUMP parameter is optional. If specified, a storage dump will be provided. The DUMP parameter must be specified to ensure that all the steps in the job are cancelled in case of a severe error. If it is not specified and such an error occurs, the step will abnormally end, but any other steps in the job will be allowed to execute. |

 *© Copyright IBM Corp. 1981, 1989*

## 9.8.3 Example

This example illustrates the creation of a sorted work file from two input work files, one from the HD reorganization reload utility and one from the data base scan utility. Three previously defined sort work files are used.

```
// JOB    PREFIX RESOLUTION
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN  SYS001,DISK,VOL='volser',SHR
// DLBL   SORTWK1,'sort work file1',,VSAM,CAT='catalog'
// EXTENT SYS001,'volser'
// ASSGN  SYS002,DISK,VOL='volser',SHR
// DLBL   SORTWK2,'sort work file2',,VSAM,CAT='catalog'
// EXTENT SYS002,'volser'
// ASSGN  SYS003,DISK,VOL='volser',SHR
// DLBL   SORTWK3,'sort work file3',,VSAM,CAT='catalog'
// EXTENT SYS003,'volser'
// ASSGN  SYS010,DISK,VOL='volser',SHR
// DLBL   INTRMED,'intermediate work file',,VSAM,CAT='catalog'
// EXTENT SYS010,'volser'
// ASSGN  SYS011,180
// TLBL   WORKFIL,'prefix work file'
// ASSGN  SYS012,DISK,VOL='volser',SHR
// DLBL   CONTROL,'control file',,VSAM,CAT='catalog'
// EXTENT SYS012,'volser'
// ASSGN  SYS013,182
// TLBL   WRKIN01,'hdam work file'
// TLBL   WRKIN02,'scan work file'
// EXEC   DLZURG10,SIZE=100K
R        L    3     02
/*
/&
```

 *© Copyright IBM Corp. 1981, 1989*

# 9.9 Chapter 25. Data Base Prefix Update Utility (DLZURGP0)

The data base prefix update utility is designed to use the output generated by the prefix resolution utility which consists of one or more update records to be applied to each segment that contains prefix information affected by a data base initial load or reorganization. The update records have been sorted into data base and segment physical location order by the prefix resolution utility. The prefix fields updated by this program include the logical parent, logical twin, logical child, and index pointer fields.

The general operation of this program consists of reading an input record from the input work file, reading the segment indicated by the input record, and applying the changes indicated by the input record to the prefix of the segment.

If no errors are detected by this program, the only output message issued is a normal program termination message. The message indicates the number of input records processed.

If execution of the program is abnormally terminated because of a data base I/O error, the cause of the I/O error should be determined and rectified. The data base should then be restored to its condition before execution of this program, and the program should then be re-executed using the complete original input work file specified by WORKFIL.

Input to the data base prefix update utility as illustrated in Figure 69 consists of:

1. One or two input work file(s) created by the data base prefix resolution utility.

2. One or more data bases that were initially loaded and/or reorganized and any logically related data bases: valid data bases are HD randomized, HDAM, HD indexed, HIDAM, and primary INDEX.

3. One control statement read from the SYSIPT device.

Output from the data base prefix update utility as illustrated in Figure 69 consists of:

1. The updated data bases that were input to the data base prefix update (see 2 above) and, optionally, secondary index data bases created by this utility.

2. Messages and statistics on the SYSLST device.

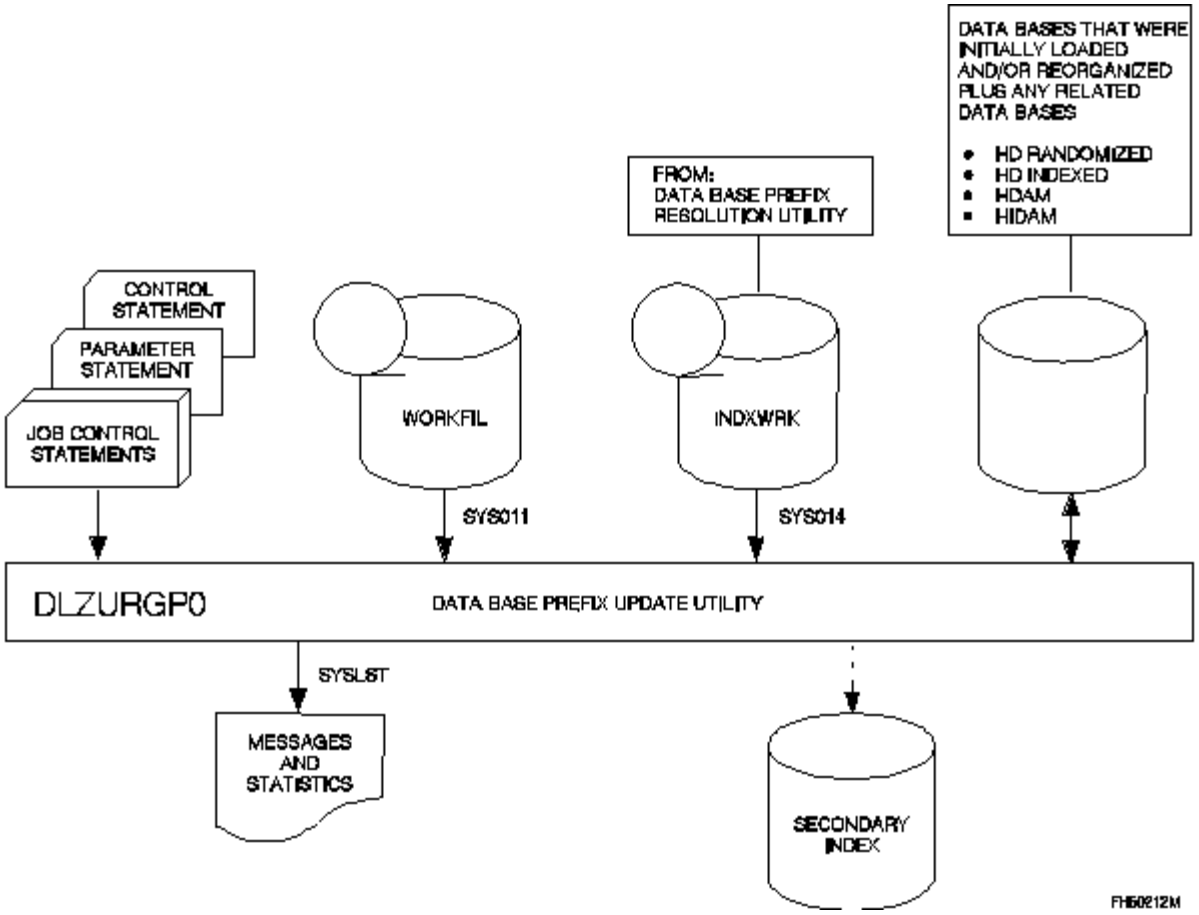



Figure 69. Data Base Prefix Update Utility

Subtopics:

- 9.9.1 Job Control Statement Requirements
- 9.9.2 Parameter Statement Requirements
- 9.9.3 Control Statement Requirements
- 9.9.4 Example

---



---

## 9.9.1 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // DLBL | filename | This statement defines the data bases that were initially loaded and/or reorganized and any logically related data bases. One statement must be present for each data base and for all related index data bases. For HD randomized and indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements (if any). For HDAM, HIDAM, and INDEX (primary and secondary) data bases, use the filename specified in the DD1 operand of DATASET statements. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS011,cuu ]<br>WORKFIL ]<br><br>WORKFIL ]<br>extent data ] | These statements define the input work file for logical relationships created by the data base prefix resolution utility. It may be DASD or tape with standard labels. WORKFIL is the symbolic name of the input file. These statements must be provided if the utility control statement indicates L or blank in column 10, that is, a logical relationship work file has been created by the prefix resolution utility. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS014,cuu ]<br>INDXWRK ]<br><br>INDXWRK ]<br>extent data ] | These statements define the input work file for secondary index relationships created by the data base prefix resolution utility. It may be DASD or tape with standard labels. INDXWRK is the symbolic name of the input file. These statements must be provided if the utility control statement indicates I or blank in column 10, that is, an index relationship workfile has been created by the prefix resolution utility. |
| // EXEC | DLZRRC00,<br>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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific

track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 9.9.2 Parameter Statement Requirements

The data base prefix update utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column 1, must be entered either via SYSIPT or SYSLOG:

```
ULU,DLZURGP0
```

# 9.9.3 Control Statement Requirements

```
1           10            25-28
U          {L}           [DUMP]
           {I}
           {}
```

| Column | Description |
|---|---|
| 1 | Must be U. |
| 10 | Must be either L, I, or blank.<br><br>L specifies that the logical relationship resolution output of the prefix resolution utility is to be used as input to this program.<br><br>I specifies that the secondary index relationship resolution output of the prefix resolution utility is to be used as input to this program.<br><br>specifies that both output files of the prefix resolution utility are to be used as input to this program. |
| 25-28 | The DUMP parameter is optional; if specified, a PDUMP will be produced if the prefix update utility ends with message DLZ958I, DLZ959I, DLZ960I, DLZ982I, or DLZ983I. |

## 9.9.4 Example

```
This example illustrates prefix updating for two data bases PAYRLDB and
SKILINV that are logically related.  The file defined by the SKILINV DD1
operand is SKLDAM.  The file defined by the PAYRLDB DD1 operand is
PAYROLL.  Additionally, since the PAYRLDB data base is HIDAM, the INDEX
data base must be defined.
```

```
// JOB    PREFIX UPDATE
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   SKLDAM,'hdam.data.base',,VSAM,CAT='catalog'
// DLBL   PAYROLL,'hidam.data.base',,VSAM,CAT='catalog'
// DLBL   HIDAMDX,'hidam.index',,VSAM,CAT='catalog'
// ASSGN  SYS011,181
// TLBL   WORKFIL,'prefix work file'
// EXEC   DLZRRC00,SIZE=350K
ULU,DLZURGP0
U        L
/*
/&
```

# 9.10 Chapter 26. Partial Data Base Reorganization Utility (DLZPRCTn)

This chapter describes how to run the partial data base reorganization utility.

The purpose of this utility is to let you unload and reorganize a selected range of data base records, and reload them into a designated target area in the data base. This utility can be used with HD, HDAM, or HIDAM data bases.

**For Information....**

about available utilities that may be used to determine areas of a data base that would benefit from a partial reorganization, see "Identifying Areas to be Reorganized" at the end of this chapter.

The partial reorganization utility has two parts. Because each part can be run separately, they are described separately in this chapter.

Subtopics:



# 9.10.1 Part 1 - DLZPRCT1

Part 1 of the Partial Data Base Reorganization Utility is illustrated in Figure 70.

Note that Part 1 does not require use of the data base. It's purpose is to perform certain pre-reorganization functions. For example, input control statements are processed to identify the range of the data base to be reorganized. For HD indexed and HIDAM data bases, the range is defined by a set of low-high key values in the primary INDEX data set. For HD randomized or HDAM data bases, the range is defined by a set of low-high relative block numbers in the root addressable area.

Part 1 also accesses a 'DL/I user lib' to determine the logically related data base(s) needing pointer resolution. Control tables are then built for Part 2 processing. Part 1 also produces a report containing information on the data base and data set(s) to be partially reorganized, the range(s) selected for reorganization, and the names of the data bases that may optionally be scanned. Error messages, if any, are also shown.

Because a special PSB is required for use in Part 2, source statements to create this PSB may be optionally produced in Part 1. These source statements are then used to do a PSBGEN and and associated ACBGEN before Part 2 processing. Once the PSB and ACB generation is done for the data base to be reorganized, the process does not have to be repeated for subsequent reorganizations of the same data base.

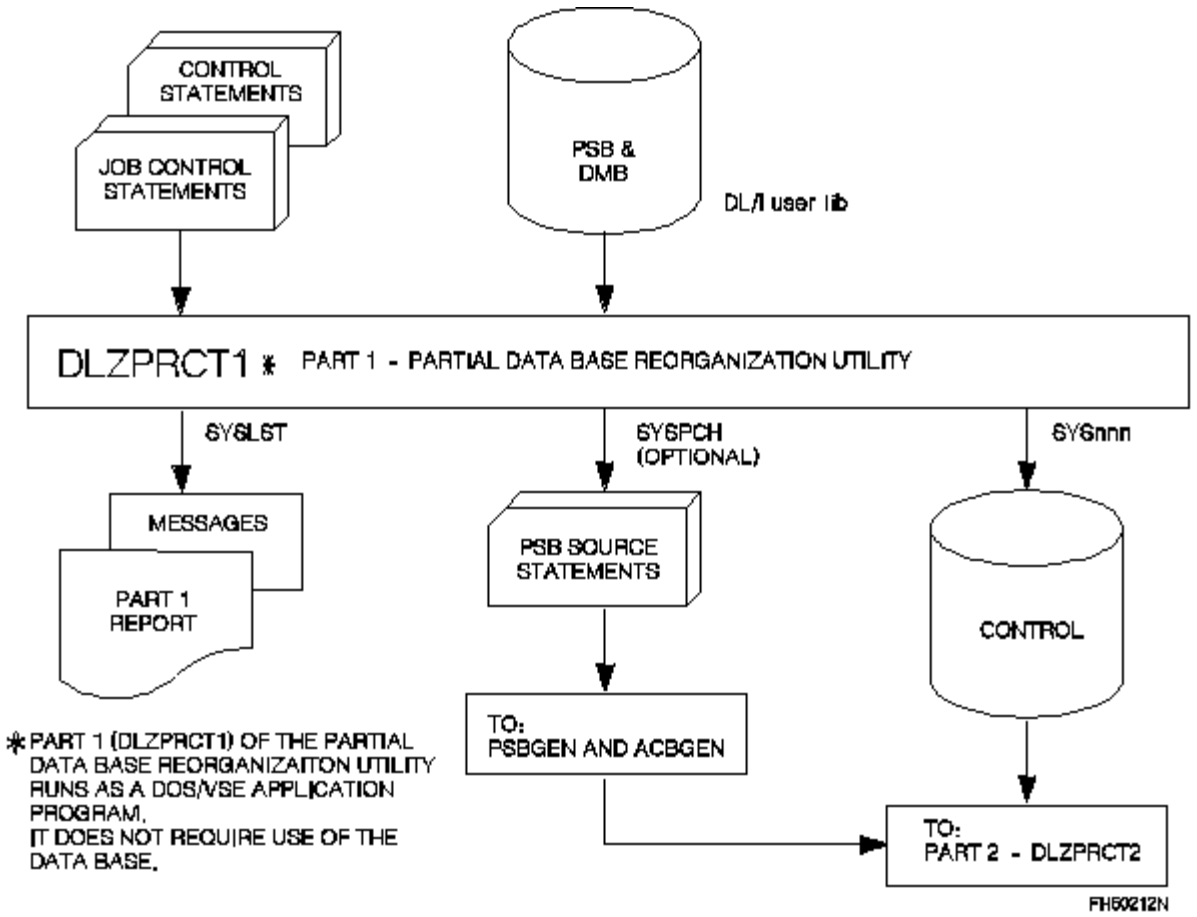



Figure 70. Part 1 of Partial Data Base Reorganization Utility

Subtopics:

- 9.10.1.1 Job Control Statement Requirements
- 9.10.1.2 Control Statement Requirements
- 9.10.1.3 Example

---



---

# 9.10.1.1 Job Control Statement Requirements

The following job control statements are used to run Part 1 of the Partial Data Base Reorganization utility. Note the placement of the Partial Data Base Reorganization utility. Note the placement of the input control statements within the job stream.

| | | |
|---|---|---|
| // ASSGN<br>// DLBL<br>// EXTENT | SYSnnn,cuu<br>CONTROL<br>extent data | These statements define the output data set. It contains control tables that are built in Part 1 and used as input for Part 2 processing. CONTROL is the symbolic name of the output file. It must be DASD. |
| // EXEC | DLZPRCT1,<br>SIZE=AUTO | This statement identifies the Part 1 program name of the Partial Data Base Reorganization utility and causes it to execute. |
| Put the Part 1 input control statements here. | | |
| /*<br>/& | | |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

 *© Copyright IBM Corp. 1981, 1989*

# 9.10.1.2 Control Statement Requirements

This section describes input control statements used for Part 2, "Describing the Characteristics of DL/I Data Bases." These statements are used to specify pre-reorganization processing options. Six keyword options are available; each keyword requires its own input statement.

Input statements follow normal coding conventions. Usually a single statement is adequate to specify the requirements of any given option. If an option extends beyond one input statement, however, continuation statements may be used. The following rules must be observed for continuation statements:

1. The preceding statement must contain a non-blank in column 72.

2. The last string character in the preceding statement must be a comma, except for the KEYRANGE statement. A KEYRANGE statement character string must end in column 71 if it is to be continued, and the next statement must start in column 16. If any character in either key is a blank and the KEYRANGE statement does not fit on one line (continuation is required), you should specify the keys in hexadecimal format.

Comment-only statements may be specified and placed anywhere within your stream of input control statements. Comment-only statements are specified by coding an asterisk in column 1.

Subtopics:



## 9.10.1.2.1 DBNAME Statement

```
 _____________________________________________________________________
|                                                                     |
|   DBNAME=dbdname                                                    |
|                                                                     |
|_____________________________________________________________________|
```

This statement must be your first input control statement and must be the only one of its type used. It identifies the HD, HDAM, or HIDAM data base to be partially reorganized. The DBNAME operand specifies the data base DBD name.

## 9.10.1.2.2 KEYRANGE Statement

```
[KEYRANGE=(low-key-value,{high-key-value})]
                         {EOD           }
```

This statement may be used only with HD indexed or HIDAM data bases. It identifies the low-to-high range of key values in the primary index data set to be reorganized. At least one of this statement type must be provided. Optionally, up to 10 may be used.

The KEYRANGE operand specifies the low and high root segment keys, with a maximum of 236 bytes each. Keys are specified as either character or hexadecimal values. Character keys are the default. Any character may be used in the keys except a comma in the low key and a right parenthesis in the high key. This restriction does not apply to keys in the hexadecimal format. Keys in hexadecimal format must be preceded with "X" and enclosed in quotes; for example, KEYRANGE=(X'C8C5E7',X'D2C5E8'). The high-key-value may be specified as EOD if the upper limit is the end of the data base.

**Note:** Each KEYRANGE statement used must be paired with a TOAREA statement. Put the TOAREA statement immediately after the KEYRANGE statement to which it is paired.

## 9.10.1.2.3 FROMAREA Statement

```
[FROMAREA=(low-block-number,{high-block-number})]
                            {EOD              }
```

This statement may be used only with HD randomized or HDAM data bases. It identifies the range of relative block numbers in the root addressable area to be reorganized. The segments to be reorganized are those that originate from, or are chained from root anchor points within the specified range. Note that this may include segments that are located physically outside the range and exclude segments that are located physically inside the range. At least one of this statement type must be provided; optionally, up to ten may be used.

The FROMAREA operand specifies the low and high block numbers in the root addressable area. The low-block-number must be at least 1. The high-block-number must either be equal to the low-block-number or greater than the low-block-number. The high-block-number may also be specified as EOD if the upper limit is the last block in the root addressable area.

**Note:** Each FROMAREA statement that you specify must be paired with a TOAREA statement. Put the TOAREA statement immediately after the FROMAREA statement to which it is paired.

## 9.10.1.2.4 TOAREA Statement

```
TOAREA=(ddname,{low-block-number,high-block-number})
               {low-block-number,EOD               }
               {EOD                                }
```

This statement identifies an area of the data set into which reorganized segments may be placed. Remember that each TOAREA statement must be paired with either a KEYRANGE or FROMAREA statement to form a pair. Put the TOAREA statement immediately after its associated KEYRANGE or FROMAREA statement.

The ddname parameter identifies the data set name of the data base being partially reorganized. This name must be identical to the name specified for the data set in the DD1 operand of the DATASET statement during DBDGEN.

The area of the data set into which reorganized segments are to be placed may be defined in three different ways:

- A specific area may be named by specifying a starting low-block-number and an ending high-block-number. The low-block-number must be at least 2. The high-block-number must either be equal to the low-block-number or greater than the low-block number.

- A starting low-block-number may be specified, but with EOD used in place of a high-block-number. In this case, the reorganized segments will be placed into an area of the data set that extends from the specified low-block-number to the end of the data set.

- EOD may be specified. In this case, the reorganized segments will be placed, starting at the end of the data set. If sufficient space is not available, an abnormal termination will occur.

Note that for HD randomized or HDAM data bases, the specification of a TOAREA statement low-to-high range that differs from the paired FROMAREA statement low-to-high range can cause the partial reorganization utility to attempt a sparser data density. Sufficient data space must be available to accommodate this expansion.

## 9.10.1.2.5 PSB Statement

```
[PSB=psbname]
```

This statement is optional. If specified, it must be the only one of its type used. The statement specifies a name for a special PSB whose source statements are produced in Part 1 of partial data base reorganization.

The source statements must then be processed by the PSB generation step (Chapter 10, "Doing a PSB Generation") and by the associated ACBGEN (Chapter 11, "Doing the ACBGEN Procedure"). This must be done before you can execute Part 2 of partial data base reorganization

Once PSBGEN and ACBGEN is done for the data base to be reorganized, the process does not have to be repeated for subsequent partial reorganizations of the same data base. You may, therefore, omit this statement if a PSB has already been created by a previous execution of partial data base reorganization and a new PSB is not desired. An example of when it would not be necessary to produce another PSB would be in a rerun of Part 1 that changes a FROMAREA specification from a previous run.

## 9.10.1.2.6 SORTOPT Statement

```
[SORTOPT='[STORAGE={n }][,PRINT={ALL     }][,DIAG]']
                  {nK}         {NONE    }
                               {CRITICAL}
```

This statement is optional.  If specified, it must be the only one of its type used.

The SORTOPT statement specifies three keyword sort options.  These options may be used to override those selected by your installation for the particular sort/merge operation used with Part 2 of partial data base reorganization.  For example, if your particular sort/merge program uses the NODIAG option (that is, no diagnostics), you can override it by specifying control statement SORTOPT='DIAG'.  This control statement will then cause special diagnostic messages to be produced during execution of Part 2 of partial data base reorganization.
For a complete description of the three sort options that may be specified, refer to *DOS/VS Sort/Merge*.

## 9.10.1.3 Example

Figure 71 shows an example of job control statements to run Part 1 of the Partial Data Base Reorganization utility and a sample output report. Page 1 of the sample output contains the input control statements. Page 2 contains the range values. Pages 3 and 4 contain the names of segments for required and optional scanning.

```
// JOB PRHI2D01  PARTIAL REORG PART1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN SYS005,DISK,VOL='volser',SHR
// DLBL CONTROL,'control file',,VSAM,RECORDS=5,RECSIZE=6000,*
               DISP=(NEW,KEEP),CAT='catalog'
// EXTENT SYS005,'volser'
// EXEC DLZPRCT1,SIZE=150K
```

```
    DL/I DOS/VS DLZPRCT1  -   PARTIAL REORGANIZATION PART 1                    DATE: 11/28/88
TIME: 08:06:08  PAGE  1
                              INPUT STATEMENTS

    0........1.........2.........3.........4.........5.........6.........7.........8
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
    DBNAME=DHD2DAA
    KEYRANGE=(A1 ,A2 )
    TOAREA=(SHD2DAA,EOD)
    PSB=PR2DAA

    DLZ627I NO PSB GENERATED FOR THIS EXECUTION OF PARTIAL REORGANIZATION
```

```
    DL/I DOS/VS DLZPRCT1 - PARTIAL REORGANIZATION PART 1                        DATE: 11/28/88
TIME: 08:06:08  PAGE  2
                           RANGE VALUES                    FOR DBD - DHD2DAA

     KEYRANGE = 'A1 '
             TO 'A2 '
     TOAREA   = EOD                DDNAME = SHD2DAA
```

```
     DL/I DOS/VS DLZPRCT1 - PARTIAL REORGANIZATION PART 1                          DATE: 11/28/88
TIME: 08:06:08  PAGE  3
                            REQUIRED SEGMENT SCAN         FOR DBD - DHD2DAA

              SEGMENT = F     DBNAME = DHD2DAA  SCAN REQUIRED
```

```
     DL/I DOS/VS DLZPRCT1 - PARTIAL REORGANIZATION PART 1                          DATE: 11/28/88
TIME: 08:06:08  PAGE  4
                            OPTIONAL SEGMENT SCAN            FOR DBD - DHD2DAA

              SEGMENT = B     DBNAME = DHD2DAA   SCAN OPTIONAL
              SEGMENT = E     DBNAME = DHD2DAA   SCAN OPTIONAL
              SEGMENT = K     DBNAME = DHD2DAA   SCAN OPTIONAL
              SEGMENT = Q     DBNAME = DHD2DAA   SCAN OPTIONAL
              SEGMENT = O     DBNAME = DHD2DAA   SCAN OPTIONAL
```

Figure 71. Example of Part 1 Job Control Statements and Output Report

# 9.10.2 Part 2 - DLZPRCT2

Part 2 of the Partial Data Base Reorganization utility is illustrated in Figure 72.

Part 2 has five major functions: unload, reload, scan, sort and update. Together they do the actual reorganization of the data base.

- Through the unload function, all root segments within a specified range and all segments dependent on those roots are unloaded in hierarchical order into work files and the space occupied by the unloaded segments is freed.

- The reload function inserts the unloaded segments into the user-designated target area of the data base. The old and new segment addresses are recorded for subsequent pointer resolution.

- The scan function accesses all segments of the logically related data base(s) that are indicated in the data base table created in Part 1. Work records are produced for segments that require pointer resolution. Resolving all pointers relating to the reloaded segments includes logically-related segments in the same data base, logically-related segments in other data bases, physical twin pointers of roots at boundaries of selected ranges, and secondary index pointers.

- The sort function updates the work records created during unload, reload, and scan processing that require new pointer values and combines relevant work records. Subsequently these records are sorted for update processing.

- The update function accesses, in a physical sequential order, those segments in each data base or secondary index affected by the partial reorganization, replacing the old pointer values with the resolved values from the sort function.

Part 2 of partial reorganization involves modification of the data base. If a failure occurs during processing, certain recovery procedures may only be taken if an output log file is available. For this reason, it is strongly recommended that DL/I logging be active during part 2 partial reorganization.

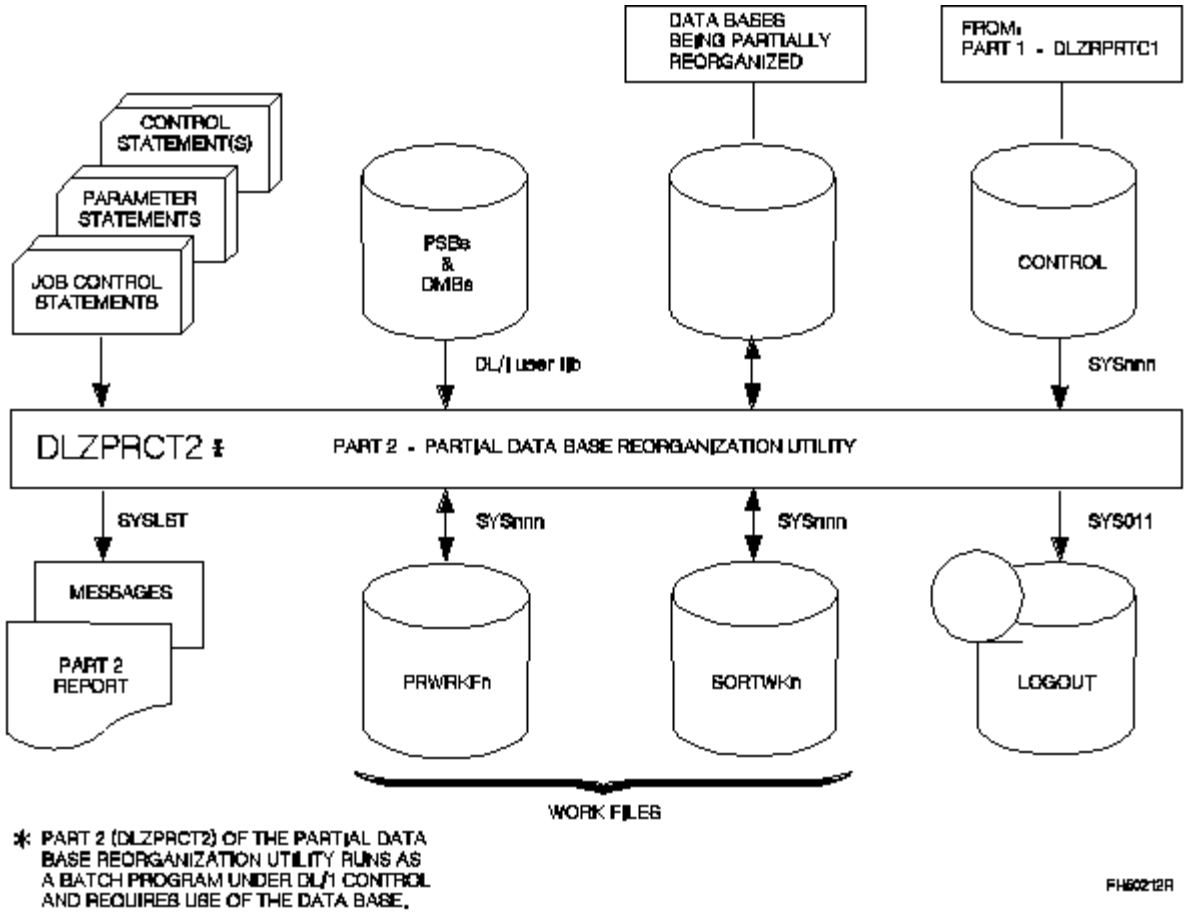



Figure 72. Part 2 of Partial Data Base Reorganization Utility

Subtopics:

- 9.10.2.1 Job Control Statement Requirements
- 9.10.2.2 Parameter Statement Requirements
- 9.10.2.3 Control Statement Requirements
- 9.10.2.4 Example

---



---

## 9.10.2.1 Job Control Statement Requirements

The following job control statements are used to run Part 2 of the Partial Data Base Reorganization Utility. Note the placement of the DL/I parameter statement and the input control statements within the job stream.

| | | |
|---|---|---|
| // UPSI | x0000xxx | The x's in bit 0, 5, 6, and 7 represent the desired DL/I function. See "UPSI Byte Settings for Batch DL/I" in Chapter 15, "Executing DL/I Programs" for more information. |
| // ASSGN<br>// DLBL<br>// EXTENT<br>°<br>°<br>° | SYSnnn,...<br>filename<br>extent data | These statements define the symbolic filename of the data set being reorganized. Each of its logically related data sets and secondary indexed data sets must also be defined. If this is a HIDAM or HD indexed data base, DLBL statements for the primary index data set must also be supplied. The filename must be the same as the data set name used in the DD1 operand of the DATASET statement during DBDGEN. |
| // ASSGN<br>// DLBL<br>// EXTENT | SYSnnn,...<br>CONTROL<br>extent data | These statements define the CONTROL data set created in Part 1 of this utility. It contains control tables required for Part 2 processing. |
| // ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN | SYSnnn,...<br>PRWRKF1<br>extent data<br><br>SYSnnn,...<br>PRWRKF2<br>extent data<br><br>SYSnnn,...<br>PRWRKF3<br>extent data<br><br>SYSnnn,...<br>PRWRKF4<br>extent data<br><br>SYSnnn,...<br>PRWRKF5<br>extent data<br><br>SYSnnn,...<br>PRWRKF6<br>extent data<br><br>SYSnnn,...<br>PRWRKF7<br>extent data<br><br>SYSnnn,...<br>PRWRKF8<br>extent data<br><br>SYSnnn,... | These statements define nine intermediate work files required for Part 2 processing. Note the symbolic filename used for each file. The intermediate workfiles must be on DASD.<br><br>Work file PRWRKF1 is used as the segment unload/reload dump file for segments being moved. All other intermediate work files contain work records that are used for pointer resolution (both data and secondary index) for segments being moved. |

| | | |
|---|---|---|
| // DLBL<br>// EXTENT | PRWRKF9<br>extent data | |
| // ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT<br><br>// ASSGN<br>// DLBL<br>// EXTENT | SYSnnn,...<br>SORTWK1<br>extent data<br><br>SYSnnn,...<br>SORTWK2<br>extent data<br><br>SYSnnn,...<br>SORTWK3<br>extent data | These statements define intermediate work files required for the sort/merge processing within Part 2. Refer to your DOS/VS SORT/MERGE manual for specific information regarding specification and size of these files. |
| [// ASSGN<br>[// TLBL<br>or<br>[// DLBL<br>[// DLBL | SYS011,...]<br>LOGOUT]<br><br>DSKLOG1]<br>DSKLOG2] | These statements define the DL/I log file. It may be tape with standard labels or DASD.<br><br>Although this output log file is optional, it is highly recommended that you always define one. Be aware that if you do not, and if a problem occurs during execution of Part 2 that prevents it from completing normally, your data base may be destroyed and some recovery procedures are not possible without logging. |
| // EXEC | DLZRRC00,<br>SIZE=xxxK | This statement identifies the program name of the DL/I Batch Initialization Module and causes it to execute. Refer to *DL/I DOS/VS Data Base Administration* for storage size requirements. |
| Put the DL/I parameter statement for Part 2 here.<br>Put the input control statements for Part 2 here. | | |
| /*<br>/& | | |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

---



---

## 9.10.2.2 Parameter Statement Requirements

A DL/I parameter statement is needed to run Part 2 of Partial Data Base Reorganization. The information it contains must begin in column 1.

```
DLI,DLZPRCT2,psbname[,{buf}][,HDBFR=][,HSBFR=][,TRACE=][,ASLOG=][,LOG=]
                      {1  }
```

DLZPRCT2 is the program name for Part 2 of the Partial Data Base Reorganization utility.

psbname is the name of the PSB created in Part 1 of this utility.

The keyword parameters buf, HDBFR, HSBFR, TRACE, ASLOG, and LOG are optional parameters that may also be entered in the parameter statement. These parameters have the same usage as for application programs and described in Chapter 15, "Executing DL/I Programs" under "DL/I Parameter Information Requirements." The LOG parameter must be specified if the output log is to be a VSAM disk file.

 *© Copyright IBM Corp. 1981, 1989*

# 9.10.2.3 Control Statement Requirements

This section describes input control statements used for Part 2. Two statements are available. The coding requirements for these statements are the same as those previously described for the input statements used in Part 1.

Subtopics:



## 9.10.2.3.1 DBNAME Statement

```
 _____________________________________________________________________________
|                                                                             |
|   DBNAME=dbdname                                                            |
|                                                                             |
|_____________________________________________________________________________|
```

This statement identifies the HD, HDAM, or HIDAM data base to be partially reorganized. The dbdname specified must be the same dbdname used in the Part 1 DBNAME statement.

## 9.10.2.3.2 SCANSEG Statement

```
[SCANSEG=(dbdname,segmentname)]
```

One or more SCANSEG statements may be optionally used in Part 2. It is recommended that all segment names listed in the Optional Segment Scan report be included in the scan process if there are required scan segments listed for the data base in the Part 1 output. It may result in improved performance. Only those segment names (and DBD names) listed in the "Optional Segment Scan" section of the report produced in Part 1 may be specified. No other segment types may be named for the scan process. It is not necessary to specify SCANSEG statements for those segment types listed in the "Required Segment Scan" section of the Part 1 report. They will be scanned automatically.

## 9.10.2.4 Example

Figure 73 shows an example of job control statements to run Part 2 of the Partial Data Base Reorganization utility for a HIDAM data base with file name SHD2DAA; the filename of the index data base is SAP2NDX. Note that the partial reorganization work files have been previously defined. Figure 74 shows a sample output report.

```
// JOB PRH12D01  PARTIAL REORG PART2
// LIBDEF ,SEARCH=('DL/I prod lib,DL/I user lib')
// DLBL SHD2DAA,'hidam data base',,VSAM,CAT='catalog'
// DLBL SAP2NDX,'index data base',,VSAM,CAT='catalog'
// ASSGN SYS008,DISK,VOL='volser',SHR
// DLBL CONTROL,'control file',0,VSAM,,CAT='catalog'
// EXTENT SYS008,'volser'
// ASSGN SYS004,DISK,VOL='volser',SHR
// DLBL PRWRKF1,'work.file1',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF2,'work.file2',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF3,'work.file3',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF4,'work.file4',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF5,'work.file5',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF6,'work.file6',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF7,'work.file7',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF8,'work.file8',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL PRWRKF9,'work.file9',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// ASSGN SYS005,DISK,VOL='volser',SHR
// DLBL SORTWK1,'sort.file1',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL SORTWK2,'sort.file2',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// DLBL SORTWK3,'sort.file3',,VSAM,CAT='catalog'
// EXTENT SYS004,'volser'
// ASSGN SYS011,DISK,VOL='volser',SHR
// DLBL DSKLOG1,'DL/I log file',,VSAM,CAT='catalog'
// EXTENT SYS011,'volser'
// EXEC DLZRRC00,SIZE=400K
DLI,DLZPRCT2,PR2DAA,LOG=(DISK1)
DBNAME=DHD2DAA
/*
/&
```

Figure 73. Example of Part 2 Job Control Statements

```
     DL/I DOS/VS DLZPRCT2 - PARTIAL REORGANIZATION PART 2                    DATE: 11/28/88
TIME: 08:06:08  PAGE  1
                          INPUT STATEMENTS

     0.......1.........2.........3.........4.........5.........6.........7.........8
     12345678901234567890123456789012345678901234567890123456789012345678901234567890
     DBNAME=DHD2DAA
```

---

```
     DL/I DOS/VS DLZPRCT2  -   PARTIAL REORGANIZATION PART 2                 DATE: 11/28/88
TIME: 08:06:08  PAGE  2
                               UNLOAD STATISTICS FOR RANGE 1         DBD - DHD2DAA
                                 *                SEGMENT STATISTICS                 *
RELOAD STATISTICS     *
     SEGMENT   SEG   DSG   BLOCK   % OF SEG IN SAME BLK AS:    AVERAGE  AVERAGE    AVERAGE   TOTAL
AVE SEG PER
     NAME      LVL   NUM   SIZE      PHY-TWIN     PHY-PAR      TWINS    CHILDREN   LENGTH
SEGMENTS  DB RECORD
     --------  ---   ---   -----   -------------------------   -------  --------   -------   ------
--  -----------

     A           1     1    1024       100.0         N/A           N/A      11.0      29.0
2           1.0
     B           2     1    1024       100.0       100.0           1.0       0.5      63.0
2           1.0
     C           3     1    1024         0.0       100.0           0.5       0.0      33.0
1           0.5
     E           2     1    1024       100.0       100.0           1.5       0.7      63.0
3           1.5
     F           3     1    1024         0.0       100.0           0.3       1.0      41.0
1           0.5
     G           4     1    1024         0.0       100.0           1.0       0.0      27.0
1           0.5
     H           2     1    1024       100.0       100.0           1.5       3.7      51.0
3           1.5
     I           3     1    1024        66.7        66.7           1.0       0.0      41.0
3           1.5
     K           3     1    1024         0.0        50.0           0.7       0.5      56.0
2           1.0
     L           4     1    1024         0.0       100.0           0.5       0.0      27.0
1           0.5
     O           3     1    1024         0.0         0.0           0.3       2.0      65.0
1           0.5
     P           4     1    1024         0.0       100.0           1.0       1.0      31.0
1           0.5
     Q           5     1    1024         0.0       100.0           1.0       0.0      65.0
1           0.5
     R           3     1    1024       100.0         0.0           0.7       0.0      33.0
2           1.0

     TOTAL SEGMENTS UNLOADED =        24

     AVERAGE DATA BASE RECORD LENGTH =     558.0

     NUMBER OF ROOT ANCHOR POINTS PER BLOCK =   1

     KEYRANGE = 'A1 '
             TO 'A2 '
```

---

```
     DL/I DOS/VS DLZPRCT2 - PARTIAL REORGANIZATION PART 2                    DATE: 11/28/88
TIME: 08:06:08  PAGE  3
                          UNLOAD STATISTICS FOR RANGE  1   DBD - DHD2DAA
           RANGE OF UNLOADED SEGMENTS


       DATA SET    LOW BLOCK    HIGH BLOCK
     GROUP NUMBER    NUMBER       NUMBER


           1             2            3
```

---

```
     DL/I DOS/VS DLZPRCT2 - PARTIAL REORGANIZATION PART 2                    DATE: 11/28/88
TIME: 08:06:08  PAGE  4
                          DISTRIBUTION OF DATA BASE RECORDS


            NUMBER OF   OBSERVED    PERCENT   CUMULATIVE   CUMULATIVE
             BLOCKS     FREQUENCY    TOTAL      PERCENT     REMAINDER


                1             0      0.00        0.00       100.00
                2             2    100.00      100.00         0.00


            MAXIMUM BLOCKS FOR A DATA BASE RECORD =   2
            MEAN OBSERVED FREQUENCY               =  2.00
```

---

```
     DL/I DOS/VS DLZPRCT2  -   PARTIAL REORGANIZATION PART 2                 DATE: 11/28/88
TIME: 08:06:08  PAGE  5
                              RELOAD STATISTICS FOR RANGE  1   DBD - DHD2DAA


     SEGMENT   SEG   DSG   BLOCK   % OF SEG IN SAME BLK AS:    RELOAD   DIFFERENCE
     NAME      LVL   NUM   SIZE      PHY-TWIN     PHY-PAR     COUNT    RELOAD-UNLOAD
     --------  ---   ---   -----   --------------------------   -------  --------


     A           1     1    1024       100.0         N/A            2         0
     B           2     1    1024       100.0       100.0            2         0
     C           3     1    1024         0.0       100.0            1         0
     E           2     1    1024       100.0         0.0            3         0
     F           3     1    1024         0.0       100.0            1         0
     G           4     1    1024         0.0       100.0            1         0
     H           2     1    1024        66.7        66.7            3         0
     I           3     1    1024        66.7       100.7            3         0
     K           3     1    1024         0.0       100.0            2         0
     L           4     1    1024         0.0       100.0            1         0
     O           3     1    1024         0.0       100.0            1         0
     P           4     1    1024         0.0       100.0            1         0
     Q           5     1    1024         0.0       100.0            1         0
     R           3     1    1024         0.0       100.0            2         0

     TOTAL SEGMENTS RELOADED =       24


     DL/I DOS/VS DLZPRCT2  -   PARTIAL REORGANIZATION PART 2                 DATE: 11/28/88
TIME: 08:06:08  PAGE  6
```

```
                              RELOAD STATISTICS FOR RANGE  1   DBD - DHD2DAA
          RANGE OF RELOADED SEGMENTS

        DATA SET    LOW BLOCK    HIGH BLOCK   BYTE COUNT INSERTED
      GROUP NUMBER    NUMBER       NUMBER         TO OVERFLOW

            1             2            3             N/A
```

---

```
     DL/I DOS/VS DLZPRCT2  -   PARTIAL REORGANIZATION PART 2                    DATE: 11/28/88
TIME: 08:06:08  PAGE  7
                              SCAN STATISTICS              DBD - DHD2DAA

                    DATA BASE NAME   SEGMENT NAME   SCAN COUNT

                       DHD2DAA         F                    2

                    TOTAL SEGMENTS SCANNED =        2

      °
      °
      °
      DLZ600I  SUCCESSFUL COMPLETION OF PARTIAL DATA BASE REORGANIZATION
```

---

Figure 74. Example of Part 2 Output Report

The various parts of the reports are explained here:

SEGMENT NAME
The name of the segment type being unloaded/reloaded.

SEG LVL
The hierarchical level number of the segment type being unloaded/reloaded.

DSG NUM
The data set group number of the segment type being unloaded/reloaded.

BLOCK SIZE
Block size of the data set group.

% OF SEG IN SAME BLK AS PHY-TWIN
The percentage of segments of this type which occupy the same data base block as the previously unloaded or reloaded physical twin segment.

% OF SEG IN SAME BLK AS PHY-PAR
The percentage of segments of this type which occupy the same data base block as the previously unloaded or reloaded physical parent segment.

AVERAGE TWINS
The average number of physical twins within the twin families.

AVERAGE CHILDREN
The average number of children of this segment type.

AVERAGE LENGTH
The average length of segments of this type.

TOTAL SEGMENTS
The total number of segments being unloaded/reloaded.

AVE SEG PER DB RECORD
The average number of segments per data base record.

RANGE OF UNLOADED SEGMENTS
The actual range of the segments being unloaded. (For an HD randomized or HDAM data base, this number may be different from the range specified.)

DATA SET GROUP NUMBER
Same as "DSG Number."

LOW BLOCK NUMBER
The lowest block number of the segments unloaded within the data set group.

HIGH BLOCK NUMBER
The highest block number of the segments unloaded within the data set group.

DISTRIBUTION OF DATA BASE RECORDS
The distribution of the data base records unloaded over the number of physical blocks. This report only tabulates one through forty blocks: distributions over forty blocks are accumulated in one entry.

NUMBER OF BLOCKS
The number of physical blocks occupied by a data base record.

OBSERVED FREQUENCY
The number of data base records observed for the distribution.

PERCENT TOTAL
The percentage of the data base records observed over the total data base records unloaded.

CUMULATIVE PERCENT
Total percentage up to this point.

CUMULATIVE REMAINDER
Total percentage remaining up to this point.

MAXIMUM BLOCKS FOR A DATA BASE RECORD
The maximum number of blocks occupied by a data base record.

MEAN OBSERVED FREQUENCY
The average number of blocks occupied by unloaded data base records.

RELOAD COUNT
The number of segments reloaded.

```
DIFFERENCE RELOAD-UNLOAD
   The difference between the number of segments unloaded and the number
   reloaded.


TOTAL SEGMENTS RELOADED
   The total number of segments reloaded for the specific range.


BYTE COUNT INSERTED TO OVERFLOW
   The number of bytes inserted into the overflow area (HDAM only)


SCAN COUNT
   The number of segments scanned for this segment type.
```

---

---

# 9.10.3 Identifying Areas to be Reorganized

An existing utility that may help you identify those data ranges that would benefit from partial reorganization and in selecting target areas for the reorganized data is the DL/I DOS/VS Space Management Utility.

The "Free Space Summary Report" created by the space management utility shows the free space that is available within a given data base.

For a complete description of the space management utility and how to use it, see *DL/I DOS/VS Space Management Utilities Program Description/Operations Manual*. Space management utility is an Installed User Program (Program Number 5796-PKF).

# 10.0 Part 10. Data Base Recovery Utilities

The following DL/I data base recovery utilities are described in this part:

***Chapter 27, "Data Set Image Copy Utility (DLZUDMP0)" in topic 10.1***
This program creates dump images of data base data sets.

***Chapter 28, "Data Base Change Accumulation Utility (DLZUCUM0)" in topic 10.2***
This program accumulates pertinent data base changes from log files and puts them into a sequential file.

***Chapter 29, "Data Set Recovery Utility (DLZURDB0)" in topic 10.3***
This program restores a damaged data base.

***Chapter 30, "Log Print Utility (DLZLOGP0)" in topic 10.4***
This program prints the contents of DL/I log files.

***Chapter 31, "Data Base Backout Utility (DLZBACK0)" in topic 10.5***
This program removes changes made to data bases by selected application programs.

Job streams for all utilities described in this section may be generated interactively on a 3270-type terminal by using the Interactive Utility Generation (IUG) Facility. See *DL/I DOS/VS Interactive Resource Definition and Utilities* for more information.

Subtopics:



# 10.1 Chapter 27. Data Set Image Copy Utility (DLZUDMP0)

The data base data set image copy utility is designed to provide an efficient dump format copy for a file, to be used as input to the data base data set recovery utility. In most cases, a data set is synonymous with a data base. This is true with HD randomized, HDAM, INDEX, and SHISAM data bases. HD indexed, HIDAM, and HISAM data bases consist of two files, a key sequenced file (KSDS) and an entry sequenced file (ESDS). The output from the HISAM reorganization utility may also be used as input to the data base data set recovery utility.

The data base data set image copy utility achieves a high rate of throughput by handling files in a physical sequential fashion, without concerning itself with physical or logical data relationships. VSAM is used to process the input files. SAM is used to process the output files.

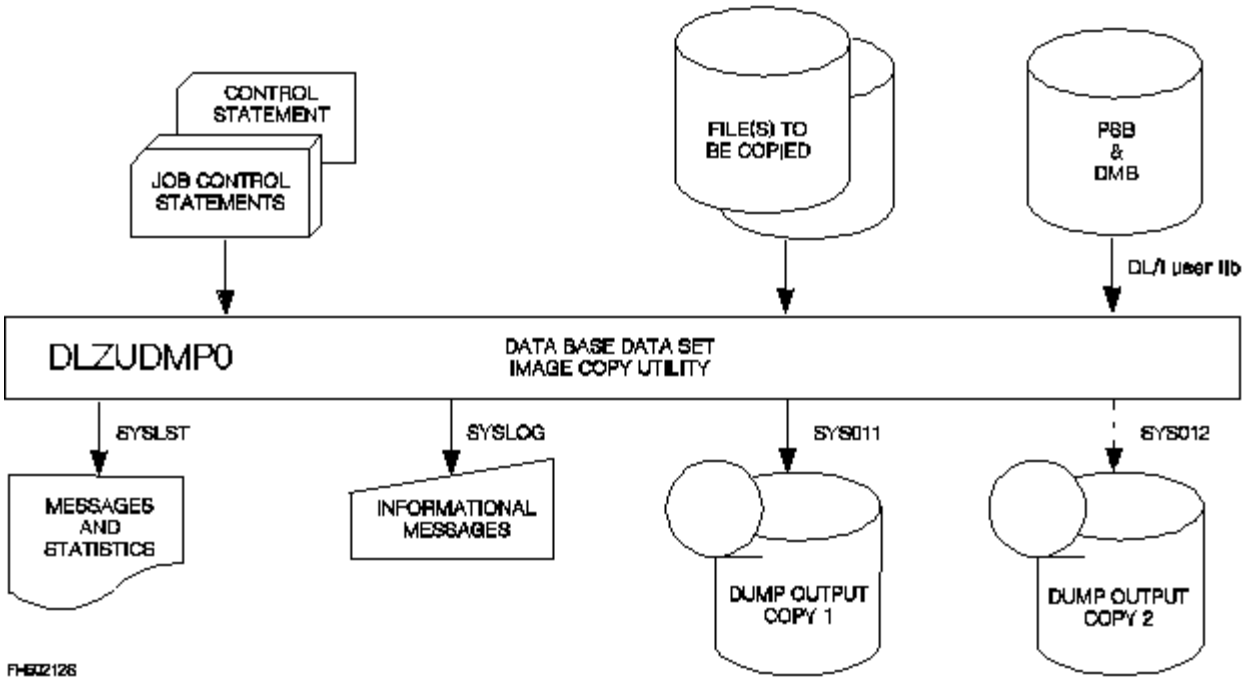


Figure 75. Data Base Data Set Image Copy Utility

Input to the data base data set image copy utility as illustrated in Figure 75 consists of:

1. A control statement read from the SYSIPT device containing the DBD name and the name of the file to be copied. One or more control statements may be processed in one execution.

2. A DMB loaded from a 'DL/I user lib'. The DMB name was obtained by expanding the DBD name specified in the control statement.

3. The input file(s) to be copied and referenced by the control statement(s).

Output from the data base data set image copy utility as illustrated in Figure 75 consists of:

1. One or two copies of the dumped file. Tape or DASD may be used and multiple copies may be produced on mixed device types. REWIND=UNLOAD is specified in the DTF for tape files, but may be changed to REWIND=NORWD by using the UPSI job control statement.

2. Informational messages and statistics on the SYSLST and SYSLOG devices.

3. Error messages on the SYSLST and SYSLOG devices.

Subtopics:



---

## 10.1.1 Image Copy Rewind Options

The options you can specify for tape, and the results of those options are:

**Nothing specified** The tape will rewind at open, and will rewind and unload at close.

**UPSI 00000001 only** The tape will not rewind either at open or close.

**N** The tape will not rewind either at open or close.

**R** The tape will not rewind at open, but will rewind at close.

**U** The tape will not rewind at open, but will rewind and unload at close.

**Note:** N, R, and U, when specified, override the UPSI specification.

 *© Copyright IBM Corp. 1981, 1989*

## 10.1.2 Job Control Statement Requirements

The data base data set image copy utility is executed as a standard VSE application program and requires the following job control statements:

| | | |
|---|---|---|
| [// UPSI | 00000001] | This optional statement changes the output tape rewind option default from REWIND=UNLOAD to REWIND=NORWD. This allows multiple output files to be placed on the same volume. You can establish the position of an output file to be dumped on a multiple tape volume by specifying a file sequence number on the TLBL statement. This statement is ignored for DASD files.<br><br>**Note:** When writing multiple files on a single tape, use one of the following to avoid overwriting a file:<br><br>1. Specify the N rewind option for all the files on the tape; then use an MTC command to rewind the tape when all jobs writing on the tape are finished.<br><br>2. Specify the R or U rewind option for a file that has a file sequence number on its corresponding TLBL statement. The R or U rewind option causes the tape to rewind when a file on the tape is closed. If a file sequence number is not specified in this case, the first file on the tape may be overwritten. |
| // DLBL | filename | This statement defines the symbolic name of the data set to be copied. One statement must be present for each file that appears in the DBD describing the data base.<br><br>For HD indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements.<br><br>For HD randomized, HDAM, HIDAM (and its primary INDEX), and SHISAM data bases use the filename defined in the DD1 operand of DATASET statements.<br><br>For HISAM data bases, use the filenames specified in the DD1 and |

| | | |
|---|---|---|
| | | OVFLW operands of DATASET statements. |
| // ASSGN<br>// TLBL(Tape)<br>or<br>// DLBL(DASD)<br>// EXTENT | SYS011,cuu<br>filename<br><br>filename<br>extent data | These statements define the first copy of the dumped output file. One is required for each file to be dumped. The filename may be any name but must appear in the associated utility control statement. The file may be a tape or DASD. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS012,cuu ]<br>filename ]<br><br>filename ]<br>extent data ] | These statements are optional and are required only if the associated utility control statement requests two copies of the dump. The filename must appear in the control statement. It may be a tape or DASD. |
| // EXEC | DLZUDMP0,<br>SIZE=xxxK | Data base data set image copy utility module. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 10.1.3 Control Statement Requirements

One control statement is required for the data base data set image copy utility. Its content is explained below. Input control statements are read from the SYSIPT device.

```
1  2 4-10     13-19   22-28    30  31-37     39  41-44  46-80

D  n dbdname  infile  outfile1 c  [outfile2] c   [fact] [comments]
```

| Column | Description |
|---|---|
| 1 | Must be D. |
| 2 | Must be a 1 or 2, depending on the number of copies required. |
| 4-10 | Must be the name of the DBD that includes the name of the file to be dumped. |
| 13-19 | Must be the symbolic filename of the input file to be dumped. It must appear as the DD1 or OVFLW parameter of the DATASET statement or the REF parameter of the ACCESS statement in the referenced DBD. Corresponding DLBL statements must be provided. |
| 22-29 | Filename of the TLBL or DLBL statement for the primary output data set (SYS011). |
| 30 | Rewind option for the primary output data set, if stored on tape. This field may be either N, for no rewind; R, for rewind; or U, for rewind and unload. The default value is U (N, if UPSI 00000001 was specified). Note that if a rewind option is specified, the tape will not rewind when it is opened. If you want the tape to rewind, use the MTC command. |
| 31-38 | Filename of the TLBL or DLBL statement for the second copy of the dumped data set (SYS012). This field must be blank if column 2 contains a 1. |
| 39 | Rewind option for the second copy of the dumped data set, if stored on tape. If column 2 contains a 2, this field may be either N, for no rewind; R, for rewind; or U, for rewind and unload. The default value is U (N, if UPSI 00000001 was specified). Note that if a rewind option is specified, the tape will not rewind when it is opened. If you want the tape to rewind, use the MTC command. |
| 41-44 | An optional factor which leads to the number of CIs that may fit into one output block. The CI size (control interval) was specified at DBD generation of the data base to be dumped. An optimum cannot be given, because factors such as type or size of data base or installation dependencies have an influence. You must specify this factor as either:<br><br>**'nnCI'** The block size is calculated to be nn (01...60) times the CI size. A 12-byte prefix for each CI within the block and four bytes for the whole must be considered. If the calculated block size exceeds the upper limit, the maximum possible block size (see 'MAX') will be selected. If the calculation is less than 4K the default is used.<br>**'MAX'** The maximum possible block size, depending on the CI size, will be selected. The maximum value for this block size is 31444 bytes.<br>If nothing is specified the default block size of 4112 bytes will be selected if the CI size is not greater than 4K. If it is greater than 4K the block size is one CI, with 16 additional bytes for internal information. |

| | |
|---|---|
| | If the algorithm to calculate the block size leads to the default value, then no information is given to the image copy header record.<br><br>Note that, depending on the CI size and the blocking factor, the additional storage requirement (GETVIS area) may vary from about 20K to 153K.<br><br>See also Table 91. |
| 46-80 | Comments. |

The table below gives sample calculations for various CI sizes and blocking factors. The calculated block sizes are in bytes. The highlighted block sizes are maximums for that CI. The same value would be calculated if 'MAX' is specified.

Table 91. Block Size Calculation Table

| CI-Size | Blocking Factor (nnCI) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 10 | 12 | 15 | 20 | 30 | 60 |
| 512 | 4112 | 4112 | 4112 | 4112 | 4112 | 4112 | 4112 | 4196 | 5244 | 6292 | 7864 | 10484 | 15724 | **31444** |
| 1024 | 4112 | 4112 | 4112 | 4148 | 5184 | 6220 | 7256 | 8292 | 10364 | 12436 | 15544 | 20724 | **31084** | |
| 1536 | 4112 | 4112 | 4648 | 6196 | 7744 | 9292 | 10840 | 12388 | 15484 | 18580 | 23224 | **30964** | | |
| 2048 | 4112 | 4124 | 6184 | 8244 | 10304 | 12364 | 14424 | 16484 | 20604 | 24724 | **30904** | | | |
| 2560 | 4112 | 5148 | 7720 | 10292 | 12864 | 15436 | 18008 | 20580 | 25724 | **30868** | | | | |
| 3072 | 4112 | 6172 | 9256 | 12340 | 15424 | 18508 | 21592 | 24676 | **30844** | | | | | |
| 3584 | 4112 | 7196 | 10792 | 14388 | 17984 | 21580 | 25176 | **28772** | | | | | | |
| 4096 | 4112 | 8220 | 12328 | 16436 | 20544 | 24652 | **28760** | | | | | | | |
| 4608 | 4624 | 9244 | 13864 | 18484 | 23104 | **27724** | | | | | | | | |
| 5120 | 5136 | 10268 | 15400 | 20532 | 25664 | **30796** | | | | | | | | |
| 5632 | 5648 | 11292 | 16936 | 22580 | **28224** | | | | | | | | | |
| 6144 | 6160 | 12316 | 18472 | 24628 | **30784** | | | | | | | | | |
| 6656 | 6672 | 13340 | 20008 | **26676** | | | | | | | | | | |
| 7168 | 7184 | 14364 | 21544 | **28724** | | | | | | | | | | |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7680 | 7696 | 15388 | 23080 | **30772** | | | | | | | | | | |
| 8192 | 8208 | 16412 | **24616** | | | | | | | | | | | |
| 10240 | 10256 | 20508 | **30760** | | | | | | | | | | | |
| 12288 | 12304 | **24604** | | | | | | | | | | | | |
| 14336 | 14352 | **28700** | | | | | | | | | | | | |
| 16384 | **16400** | | | | | | | | | | | | | |
| 18432 | **18448** | | | | | | | | | | | | | |
| .. | .. | | | | | | | | | | | | | |
| 30720 | **30736** | | | | | | | | | | | | | |

---



---

---

## 10.1.4 Examples

***Example 1:*** The first example copies one file with the filename HDAMDB from the HDAM data base named DLIDBD4 with a maximum possible block size. The name of the output file is DBOUT1.

---

```
// JOB    IMAGE1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   DBOUT1,'hdam image1'
// EXEC   DLZUDMP0,SIZE=100K
D1 DLIDBD4  HDAMDB   DBOUT1             MAX
/*
/&
```

---

***Example 2:*** The second example copies two files with the filenames HISDB1 and HISDB2 from the data base named DLIDBD1. Two copies of file HISDB1 and one copy of HISDB2 are created. Because the UPSI statement is specified in this example, the first dumped output is a multi-file tape volume. It contains the first copy of HISDB1 and the copy of HISDB2 as the first and second files, respectively. The second dumped output, also a tape, contains the second copy of HISDB1.

---

```
// JOB    IMAGE2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// UPSI   00000001
// DLBL   HISDB1,'hisam.data.base.ksds',,VSAM,CAT='catalog'
// DLBL   HISDB2,'hisam.data.base.esds',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   DBAOUT1,'ksds image',,,,1
// ASSGN  SYS012,181
// TLBL   DBAOUT2,'ksds image'
// TLBL   DBBOUT1,'esds image',,,,2
// EXEC   DLZUDMP0,SIZE=100K
D2 DLIDBD1  HISDB1   DBAOUT1 UDBAOUT2 U
D1 DLIDBD1  HISDB2   DBBOUT1 U
/*
/&
```

---

***Example 3:*** Same as the first example; one file named HDAMDB from the HDAM data base named DLIDBD4 will be copied. The calculated block size will be 24724 bytes because the CI size of the copied data base is 2048 bytes. The name of the output file is DBOUT2.

---

```
// JOB    IMAGE3
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// ASSGN  SYS011,180
// TLBL   DBOUT2,'hdam image2'
// EXEC   DLZUDMP0,SIZE=100K
D1 DLIDBD4  HDAMDB   DBOUT2            12CI
/*
/&
```

---

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 10.2 Chapter 28. Data Base Change Accumulation Utility (DLZUCUM0)

The data base change accumulation utility provides the data base data set recovery utility with a sequential file that contains only the information from the log files necessary for recovery. This is done by eliminating all log records which are not data base change records, by specifying a purge date(s) and eliminating all data base log records created before that date, by sorting the acceptable data base log records, and by combining all data base log records that update the same segment. The resulting records are grouped by data base and are sequenced by data base and data set. This utility may be executed periodically to incorporate additional data base changes and delete changes which are no longer significant. The input to this utility may be one or more DL/I system log files and a previous data base accumulation change file, if one exists.

The use of a purge date is optional. The user may specify one purge date for all data base log records that he is processing or for all data base log records that update a particular data base. If a purge date is specified on any DB0 or DB1 control card, log records which were created prior to the purge date are eliminated. When updating an old data base change accumulation file through the execution of this utility, old accumulation change records matching a DB0 identifier and having a date prior to the purge date are eliminated and not written onto the new accumulation change file.

The data base data set recovery utility does not support a change accumulation file as input for SHISAM. However, the data base change accumulation utility may be used to read all log files containing SHISAM updates and write them to one output log file. This is done by specifying the SHISAM data base name in a DB1 control card.

The input to the data base change accumulation utility, as illustrated in Figure 76, consists of:

1. All log files since the last data base data set image copy utility execution or the last run of this utility. This includes log files created after execution of the data base backout utility.

2. The previous data base change accumulation file (if any). This is the output from the last execution of this utility.

3. Control statements to specify the input and output files and how the data base records are to be processed. These statements are read from SYSIPT. There are seven types:

   a. ID statement, to define the general environment for this run of the utility.
   b. AI statement, to define the accumulation change input file.
   c. AO statement, to define the accumulation change output file.
   d. DB0 statement, to define which data base records are accumulated to the new accumulation change output file.
   e. DB1 statement, to define which data base records are selected to the new data base log output file.

f. LI statement, to define the data base log input file(s).
g. LO statement, to define the data base log output file(s).

Output from the data base change accumulation utility as illustrated in Figure 76 consists of some or all of the following files:

1. A new change accumulation file: a sequential file containing the combined data base records for the data base(s) identified in the DB0 control card(s).

2. A data base log file that contains data base log records identified on a DB1 control statement. The purpose of the data base log file is to avoid reprocessing the original log files. The new data base log may be used in place of the original log for another run of this utility or as the log input to the data base data set recovery utility.

3. Informational messages on the SYSLST device.

4. Error messages on the SYSLST and SYSLOG devices.

**Note:** Different record formats are used for log (variable-blocked) and accumulation (spanned) files.

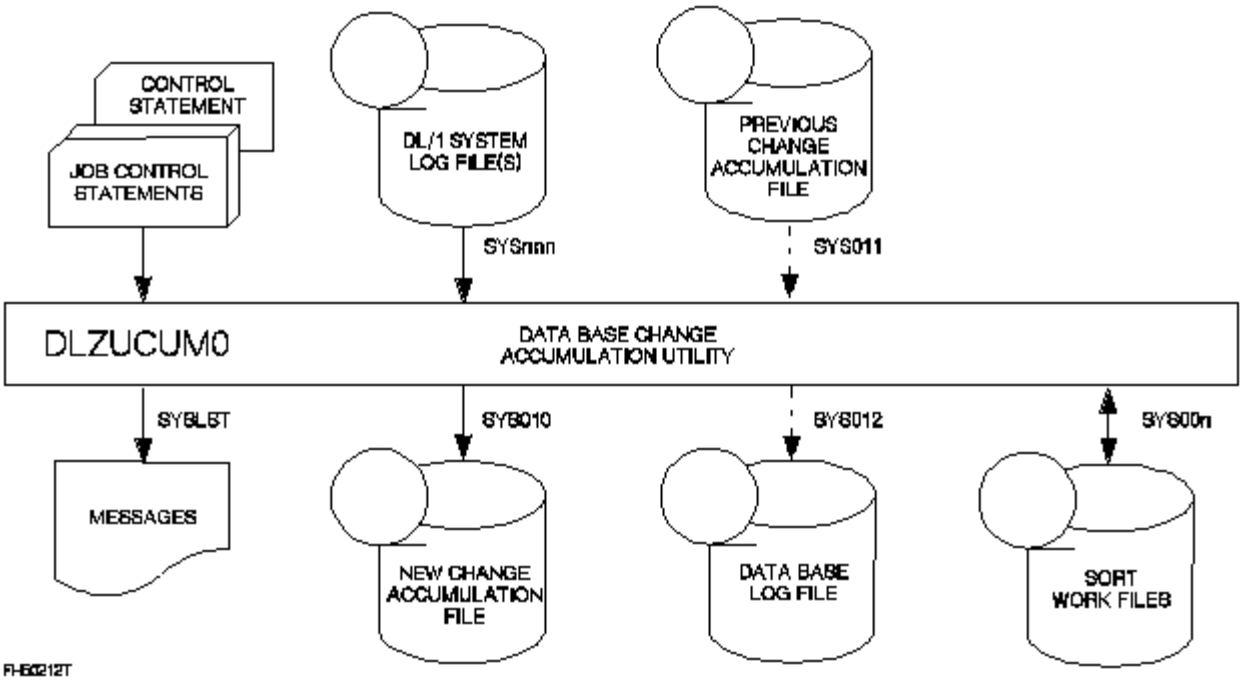



Figure 76. Data Base Change Accumulation Utility

Subtopics:



---

---

## 10.2.1 Job Control Statement Requirements

The data base change accumulation utility is executed as a DOS/VS application program and requires the following job control statements.

| | | |
|---|---|---|
| [// ASSGN<br>[// DLBL(DASD)<br>[// EXTENT | SYS00n,cuu]<br>SORTWKn]<br>extent data] | These statements define the intermediate storage files for the sort/merge program. Refer to *DOS/VS Sort/Merge Reference* regarding specification and size of intermediate storage files. If there is to be no new accumulation output, the sort program is not used and no work files need be assigned. Sort work files are numbered consecutively from SYS001 to SYS008. Filename assignments are from SORTWK1 to SORTWK8. |
| // ASSGN<br>for tape:<br>[// TLBL<br>or for DASD<br>[// DLBL<br>[// EXTENT | SYS010,{cuu}<br>{IGN}<br>CUMOUT]<br><br>CUMOUT]<br>extent data] | These statements define the new accumulated change output file. It may be either DASD or tape with standard labels. CUMOUT is the symbolic name of the output file (DTF). If there is to be no new accumulation output, specify IGN and omit the other statements in this group or do not assign this file. |
| // ASSGN<br>for tape:<br>[// TLBL<br>or for DASD:<br>[// DLBL<br>[// EXTENT | SYS011,{cuu}<br>{IGN}<br>CUMIN]<br><br>CUMIN]<br>extent data] | These statements define the old accumulated change input file. It may be either DASD or tape with standard labels. CUMIN is the symbolic name of the input file (DTF). If no accumulated change input file is used, specify IGN on the ASSGN statement and omit the other statements in this group. |
| // ASSGN<br>for tape:<br>[// TLBL<br>or for DASD:<br>[// DLBL | SYS012,{cuu}<br>{IGN}<br>LOGOUT]<br><br>LOGOUT] | These statements define the new output log file. It may be tape with standard labels or a VSAM disk file. LOGOUT is the symbolic name of the output file. If there is to be no new output log data set, specify IGN and omit the TLBL statement or DLBL statement or do not assign this file. |
| [// ASSGN<br>for tape:<br>[// TLBL<br>or for DASD:<br>[// DLBL<br>[// EXTENT | SYSnnn,X'cuu'<br>LOGINnn]<br><br>LOGINnn]<br>extent data] | These statements define the old input log file(s) containing the change records to be accumulated. It must be tape with standard labels unless specified differently in an LI control statement. SYSnnn is the logical unit to be assigned where nnn must be 013 unless specified differently in an LI control statement. LOGINnn is the symbolic name of the input file(s) (DTF), where nn is a numeric value from 01 to 99. Log files are numbered consecutively from LOGIN01 to LOGIN99. |

| | | |
|---|---|---|
| // EXEC | DLZUCUM0<br>SIZE=xxxK | Data base change accumulation utility module. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |
| Put your control statements here. | | |
| /*<br>/& | | |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM " Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

**Note:** VSE message OP47A UNX INTERV SYS013=cuu may appear on the SYSLOG-device whenever the second and successive input log files are opened, and the previous file is still rewinding. This is a normal condition, and processing continues when the correct log file is mounted.

---

## 10.2.2 Control Statement Requirements

Seven types of control statements are used by this program. All or any combination of statements may be supplied, subject to the limitations described in the following section. Input control statements are read from the SYSIPT device.

° ID statement

This statement is used to describe the table requirements and sort requirements for this change accumulation execution. This statement is optional. If it is supplied, it must be the first statement. If it is not supplied, default values are assigned as described below. Any number specified must be left-justified or have leading zeros.

If no ID control statement is present, the following defaults are assumed:

Maximum number of data base names - 16
Maximum prime key length - 10
Actual number of sort work files - 1
Actual number of log input files - 1.

| 1-2 | 11-13 | 31-33 | 41 | 51-52 |
|---|---|---|---|---|
| ID | number of DBs | maximum key length | number of sort work files | number of input log files |

| Column | Description |
|---|---|
| 1-2 | Must be ID. |
| 11-13 | Maximum number of successive DB0 and DB1 control statements. This value must be in the range 1-999 and must be left-justified or supplied with leading zeros. The default value for this entry is 16. |
| 31-33 | Maximum-length root sequence field contained within the log records to be processed as a result of a DB0 control statement. This need not be specified if no new change accumulation file is to be created. This value is used to pad the sequence field with binary zeros for sorting purposes. This value must be in the range 1-236 and must be left-justified or supplied with leading zeros. If there are no KSDS records to be processed, this value should be 4, the length of the relative byte address field. The default value for this entry is 10. |

| | |
|---|---|
| 41 | Number of work files assigned for the sort. This need not be specified if no new change accumulation file is to be created. The value must be in the range of 1-8. The default value is 1. |
| 51-52 | Number of input log files (from 1 to 99). This is equal to the maximum value nn used on LOGINnn TLBL or DLBL card(s). The default value is 1. This value is overridden if LI control statements are supplied. |

° AI statement

This statement is used to control positioning for the input change accumulation file. Only one AI statement is allowed. The statement format is:

```
    1-2    11-16   18

    AI     SYSnnn  U
                   N
                   R
```

| Column | Description |
|---|---|
| 1-2 | Must be AI. |
| 11-16 | Logical unit for the input change accumulation file. It must be of the form SYSnnn, where nnn is a valid logical unit assignment. If this parameter is omitted, SYS011 is assumed. |
| 18 | Rewind option for the input change accumulation file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, U is assumed. |

° AO statement

This statement is used to control positioning for the output change accumulation file. Only one AO statement is allowed. The statement format is:

```
    1-2    11-16   18

    AO     SYSnnn  U
                   N
                   R
```

| Column | Description |
|---|---|
| 1-2 | Must be AO. |
| 11-16 | Logical unit for the output change accumulation file. It must be of the form SYSnnn, where nnn is a valid logical unit assignment. If this parameter is omitted, SYS010 is assumed. |
| 18 | Rewind option for the output change accumulation file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, U is assumed. |

° DB0 statement

This statement is used to describe which records are to be accumulated from the input log files and the old change accumulation file, if one is provided as input, and written to the new change accumulation file. One or more of these statements may be used, or this statement may be omitted. If it is omitted and at least one DB1 statement is used, there will be no new change accumulation file.

If no DB0 or DB1 control statements are present, all data base log records are sorted and combined to produce a new change accumulation file. No purge date is assumed and no new output log file is created. This is equivalent to supplying DB0 and *ALL with no purge date. The options available are described below.

```
  1-3  4-10    12-20

  DB0  dbdname yydddhhmm
        *ALL
        *OTHER
```

| Column | Description |
|---|---|
| 1-3 | Must be DB0. |
| 4-10 | Data base name, *ALL, *OTHER, or blank.<br><br>*ALL - The purge date in columns 12-20, if any, is applied to all input change records. All records not purged are written onto the new change accumulation file. This must be the only DB0 statement.<br><br>*OTHER - All change records not matching data base names specified DB1 statements and not purged, are accumulated. *OTHER can be specified in conjunction with the data base name option on other DB0 statements, or as the only DB0 statement. In either case, only the data base name option is allowed on the DB1 statements, and at least one must be specified. |

| | |
|---|---|
| | dbdname - The purge date is applied to those change records which match the supplied data base name (DBD name), and the resulting records are accumulated. Any number of DB0 statements with this option may be supplied.<br><br>If an old change accumulation file is provided as input, all records described by the DB0 statement(s) are checked for purging as specified. Any other records are written to the new change accumulation file. |
| 12-20 | Purge date in the form YYDDDHHMM or blank, where YY=year, DDD=day of year, HH=hour (00-23), MM=minute (00-59). If a purge date is supplied, all records matching a data base name description and dated before the purge date are eliminated. If an old accumulated change input is supplied, any records matching the data base name described by the identification in columns 4-10 and dated before the purge date are not merged into the new change accumulation file. If this field is blank, no purge date is used. |

° DB1 statement

This statement is used to specify which records are to be written to the new log file. These records are not sorted and are written as read. Any log records that are not change records are eliminated. Any number of DB1 statements may be provided as input according to the rules below. If the statement is omitted, no new output log file is created.

If no DB0 or DB1 control statements are present, all data base log records are sorted and combined to produce a new change accumulation file. No purge date is assumed and no new output log file is created. This is equivalent to supplying DB0 and *ALL with no purge date.

| 1-3 | 4-10 | 12-20 |
|---|---|---|
| DB1 | dbdname | yydddhhmm |
| | *ALL | |
| | *OTHER | |

| **Column** | **Description** |
|---|---|
| 1-3 | Must be DB1. |
| 4-10 | Data base name, *ALL, *OTHER, or blank.<br><br>*ALL - All change records not purged are written to the new output log file. This must be the only DB1 statement supplied.<br><br>*OTHER - All change records not described by dbdnames on DB0 statements and not purged are written to the new log file. *OTHER cannot be specified on a DB0 statement in this case. A DB0 statement must be supplied if *OTHER is used in a DB1 statement.<br><br>dbdname - All input change records matching the data base name (DBD name) and not purged are written to the log file. Any number of DB1 statements with this option may be supplied. |

| | |
|---|---|
| 12-20 | Purge date in the form YYDDDHHMM or blank, where YY=year, DDD=day of year, HH=hour(00-23), MM=minute(00-59). All records matching a record identification combination and before the purge data are eliminated. |

° LI statement

This statement describes the input log files. Up to 99 LI statements may be specified. The first LI statement describes log file LOGIN01; the second LI statement, LOGIN02, etc. The necessary job control statements must also be specified for each log file. The LI control statement must follow any ID or DB control statements.

| 1-2 | 11-16 | 18 | 21 | 31-35 |
|---|---|---|---|---|
| LI | [SYSnnn] | U<br>N<br>*R* | *L*<br>U<br>C<br>V<br>S | [nnnnn] |

| Column | Description |
|---|---|
| 1-2 | Must be LI. |
| 11-16 | Logical unit for this log file. It must be of the form SYSnnn, where nnn is a valid logical unit assignment. If this parameter is omitted, SYS013 is assumed. |
| 18 | Rewind option for the input log file, if on tape. The following entries are valid:<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, R is assumed. |
| 21 | Log file type:<br><br>L - Specifies standard labeled tape log file (DL/I).<br>U - Specifies unlabeled tape log file (CICS/DOS/VS).<br>C - Specifies standard labeled tape log file (See Note below).<br>V - Specifies VSAM disk log file (DL/I).<br>S - Specifies SAM disk log file (CICS/DOS/VS).<br><br>If this parameter is omitted, L is assumed.<br><br>**Note:** If the output log tape from log print is used as input, the type should be specified as follows:<br><br>No CICS/DOS/VS journals used as input to log print, specify L.<br>If CICS/DOS/VS journal(s) used as input to log print, specify C. |
| 31 | Buffer size in bytes used for a CICS/DOS/VS journal. If U, C, or S is specified in column 21, the buffer size must be at least 1100 bytes, but not more than 32767. The value must be left justified. If this parameter is omitted, 1100 is assumed for U, C, or S types, and 1024 is assumed for L or V types. DL/I log files always have a buffer size of 1024 bytes. |

If no LI control statements are present, all input log files are assumed to be DL/I standard labeled tape log files on logical unit SYS013 with a buffer size of 1024 bytes. The number of input log files is determined from the ID control statement.

° LO statement

This statement is used to control positioning for the output log files. Only one LO statement is allowed. The statement format is:

| 1-2 | 11-16 | 18 |
|---|---|---|
| LO | SYSnnn | *N*<br>U<br>R |

| Column | Description |
|---|---|
| 1-2 | Mut be LO. |
| 11-16 | Logical unit for the output log file. It must be of the form SYSnnn, where nnn is a valid logical unit assignment. If this parameter is omitted, SYS012 is assumed. |
| 18 | Rewind option for the output log file tape.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, N is assumed. |

## 10.2.3 Examples

***Example 1:*** This example shows that all records for data base DLIDBD1 are to be accumulated, eliminating all records for the data base before August 5th, 1988 (that is day 218 of year 1988) and before 1200 hours. All records for other data bases are to be written to the new log file.

```
// JOB DBACCUM1 CHANGE ACCUMULATION
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL SORTWK1,'sort workfile 1',0,VSAM,CAT='catalog',RECSIZE=4096,*
               DISP=(,DELETE),RECORDS=2000
// DLBL SORTWK2,'sort workfile 2',0,VSAM,CAT='catalog',RECSIZE=4096,*
               DISP=(,DELETE),RECORDS=2000
// DLBL SORTWK3,'sort workfile 3',0,VSAM,CAT='catalog',RECSIZE=4096,*
               DISP=(,DELETE),RECORDS=2000
*
// DLBL LOGOUT,'DLI log 3',,VSAM,CAT='catalog'
// ASSGN SYS013,DISK,VOL='volser',SHR
*
// DLBL LOGIN01,'DLI log 1',,VSAM,CAT='catalog'
// ASSGN SYS014,DISK,VOL='volser',SHR
*
// DLBL LOGIN02,'DLI log 2',,VSAM,CAT='catalog'
// ASSGN SYS015,DISK,VOL='volser',SHR
*
// TLBL LOGIN03,'DLI log 4'
// ASSGN SYS015,180
// MTC REW,SYS015
*
// TLBL CUMIN,'old acc in'
// ASSGN SYS016,181
// MTC REW,SYS016
*
// TLBL CUMOUT,'new acc out'
// ASSGN SYS017,182
// MTC REW,SYS017
*
// EXEC DLZUCUM0
ID        001                 050       3         03
DB0DLIDBD1 882181200
DB1 *OTHER
LI        SYS013    V
LI        SYS014    V
LI        SYS015 U  L
AI        SYS016 N
AI        SYS017 R
/*
/&
```

***Example 2:*** This example shows the accumulation of all data base change records. The default ID statement specifies a maximum number of 16 data bases, a maximum prime key length of 10, an actual number of one sort work file, and an actual number of one input log file. The DB0 control statement specifies that all records are accumulated, with all records created before day 218 of year 1988 being eliminated. An old accumulation change file is to be merged with the new accumulation change file. The purge date is also applied to the old accumulation change file.

---

```
// JOB DBACCUM2 CHANGE ACCUMULATION
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL SORTWK1,'sort workfile 1',0,VSAM,CAT='catalog',RECSIZE=4096,*
               DISP=(,DELETE),RECORDS=2000
*
// ASSGN SYS012,IGN
*
// TLBL LOGIN01,'DLI log 1'
// ASSGN SYS013,180
*
// TLBL CUMIN,'old acc in'
// ASSGN SYS010,181
*
// TLBL CUMOUT,'new acc out'
// ASSGN SYS011,182
*
// EXEC DLZUCUM0
DB0 *ALL   882180000
/*
/&
```

---

***Example 3:*** This example shows the accumulation of all data base change records from a CICS journal. The DB0 control statement specifies that all records are accumulated, with no records being eliminated. An old accumulation change file is to be merged with the new one.

---

```
// JOB DBACCUM3 CHANGE ACCUMULATION
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// DLBL SORTWK1,'sort workfile 1',0,VSAM,CAT='catalog',RECSIZE=4096,*
               DISP=(,DELETE),RECORDS=2000
// DLBL SORTWK2,'sort workfile 2',0,VSAM,CAT='catalog',RECSIZE=4096,*
               DISP=(,DELETE),RECORDS=2000
*
// TLBL CUMOUT,'new acc out'
// ASSGN SYS010,180
*
// TLBL CUMIN,'old acc in'
// ASSGN SYS011,181
*
// ASSGN SYS012,IGN
*
// DLBL LOGIN01,'CICS system log A',0,SD
// EXTENT SYS013,'extent information'
// ASSGN SYS013,DISK,VOL='volser',SHR
*
// DLBL LOGIN02,'CICS system log B',0,SD
```

```
// EXTENT SYS014,'extent information'
// ASSGN SYS014,DISK,VOL='volser',SHR
*
// EXEC DLZUCUM0
ID        005                 050       2         02
DB0 *ALL
LI        SYS013    S
LI        SYS014    S
/*
/&
```

---

---

 © *Copyright IBM Corp. 1981, 1989*

---

# 10.3 Chapter 29. Data Set Recovery Utility (DLZURDB0)

The data base data set recovery utility program is a flexible, easy to use, fast utility program designed to recover a file that has become unusable due to a hardware failure.

The data base data set recovery utility achieves a high throughput by manipulating an individual file on a physical data replacement basis in a physical sequential fashion. The basic input consists of a data base (file) image copy file and, optionally, an accumulated change file (not supported for SHISAM). The image copy input may consist of a dump created by the data base data set image copy utility or, if the file to be recovered is of the HISAM organization, a dump created by the HISAM reorganization unload utility, as long as the data base was reloaded with the HISAM reorganization reload utility prior to applying changes under DL/I (log). The general operation consists of taking an image copy input, merging the application data from the accumulated change input, and creating a new data base file. Finally, any DL/I system logs which are not included in the accumulated change input are applied. This last operation is accomplished in a random access fashion using facilities provided by the DL/I system. It is considerably faster to apply the changes in a sequential order from the accumulated change input, and therefore to the user's advantage to make use of this facility.

**Note:** HD indexed and HIDAM data bases must have both the KSDS and ESDS files recovered to ensure integrity.

Input to the data base data set recovery utility as illustrated in Figure 77 consists of:

1. Control statements read from the SYSIPT device. The S control statement contains the DBD name and the name of the file to be recovered. The LI control statement(s) describes the input log file(s). The DI control statement specifies the tape rewind option for the image copy or HISAM unload input file.

2. A DMB and utility PSB loaded from a 'DL/I user lib'. The DMB and PSB names are obtained by DL/I by expanding the DBD name specified in the parameter statement.

3. Input files in any one of the following combinations:

    a. Image copy input only (from data base data set image copy utility or HISAM reorganization unload utility).
    b. Image copy input and change accumulation input (from data base change accumulation utility - not supported for simple HISAM).
    c. Image copy input and log file.
    d. Image copy input, change accumulation input, and log file (not supported for simple HISAM).
    e. Log file only. The VSE Fast Copy Disk utility may be used instead of the DL/I Image Copy utility to make periodic copies of a data base residing completely on one volume. If this is the case, then the Data Base Data Set Recovery utility may be run after a failure

with only log file input, after the volume containing the data base has been restored using the VSE Fast Copy Disk utility.

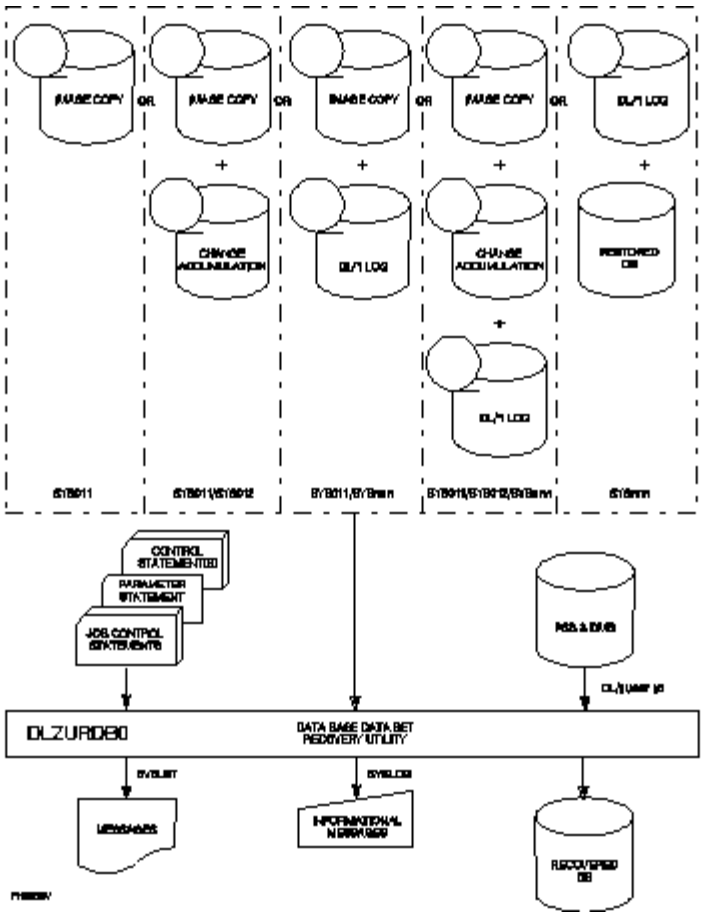



Figure 77. Data Set Recovery Utility

If image copy input is used, the user must execute the VSAM Access Method Services utility command DELETE to remove the data base data set name from the VSAM catalog and release the space allocated for it. He must then DEFINE the new file to VSAM to cause a definition of the new file to be recorded on the VSAM catalog.

Output from the data base data set recovery utility consists of:

1. A recovered file

2. Informational messages written to the SYSLST and SYSLOG devices.

3. Error messages written to the SYSLST and SYSLOG devices.

Note that if the HISAM unload output file is to be used as the input file to the data base data set recovery utility, the data base must be reloaded immediately following the unload. This procedure ensures that the recovery program ignores any records on the DL/I log which were created

prior to the unload/reload operation. This is necessary because the HISAM reorganization unload utility reorganizes the data base, and any log records created before the data base is reloaded do not reflect an image of the data base as it appears after the data base has been reloaded by the data base data set recovery utility.

Because data base activity and sizes of data bases vary, the frequency with which an image copy is created and change tapes are accumulated is left to the user's discretion.

Subtopics:



---

## 10.3.1 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // ASSGN<br><br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS011,{cuu}<br>{IGN}<br>DUMPIN]<br><br>DUMPIN]<br>extent data] | These statements define the input file. It may be a file created by the data base data set image copy utility or the HISAM reorganization unload utility. It may be DASD or tape with standard labels. DUMPIN is the symbolic name of the input file (DTF). If no input is supplied, IGN must be specified but the other statements in this group may then be omitted. |
| // ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYS012,{cuu}<br>{IGN}<br>DUMPIN]<br><br>DUMPIN]<br>extent data] | These statements define the accumulated change input file. It may be either DASD or tape with standard labels. CUMIN is the symbolic name of the input file (DTF). If no input is supplied, IGN must be specified but the other statements in this group may then be omitted. |
| // ASSGN<br><br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD) | SYSnnn,{cuu}<br>{IGN}<br>LOGINnn]<br><br>LOGINnn] | These statements define the input log file. It must be tape with standard labels unless specified differently in an LI control statement. SYSnnn is the logical unit to be assigned where nnn must be 013 unless specified differently in an LI control statement. LOGINnn is the symbolic name of the input file(s) (DTF or ACB) where nn is a numeric value from 01 to 99. Log files are numbered consecutively from LOGIN01 to LOGIN99. If no log input is supplied, IGN must be specified but the TLBL statement or DLBL statements may then be omitted. |
| // DLBL | filename | This statement defines the symbolic name of the data set to be recovered. One statement must be present for each file that appears in the DBD describing the data base.<br><br>For HD randomized and HD indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements (if any).<br><br>For HDAM, HIDAM, INDEX, and SHISAM data bases, use the filename defined in the DD1 operand of DATASET statements.<br><br>For HISAM data bases, use the filenames specified in the DD1 and OVFLW operands of DATASET |

| | | |
|---|---|---|
| | | statements. |
| // EXEC | DLZRRC00,<br>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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

Also, consider using the VSE/VSAM "Space Management for SAM Feature" whenever possible for your SAM disk files. The files can then take full advantage of VSAM's processing capabilities. For example, the job control data is simplified because specific track/block locations need not be specified. Also, jobs are less likely to terminate abnormally from lack of sufficient space because VSAM supports secondary allocation. See *Using the VSE/VSAM Space Management for SAM Feature* for more information.

VSE message OP47A UNX INTERV SYS013=cuu may appear on the SYSLOG device whenever the second and successive input log files are opened and the previous file is still rewinding. This is a normal condition, and processing continues when the correct log file is mounted.

---

## 10.3.2 Parameter Statement Requirements

The data base data set recovery utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column one, must be entered via either SYSIPT or SYSLOG:

```
UDR,DLZURDB0,dbdname[,{buf}][,HDBFR=][,HSBFR=][,TRACE=]
                      {1  }
```

**dbdname**
is the name of the SHISAM, HISAM, HD indexed, HD randomized, HIDAM, HDAM, or INDEX DBD that includes the file to be recovered. The DL/I utility PSB name is generated from this name and the PSB is loaded. When recovering a primary INDEX data base, the dbdname must be that of the HD indexed or HIDAM data base, since only one utility PSB is created for HD indexed or HIDAM.

The positional parameter 'buf' and the keyword parameters HDBFR, HSBFR, and TRACE are optional parameters that may also be entered in the parameter statement. These parameters have the same usage as for application programs and are described in Chapter 15, "Executing DL/I Programs," under "DL/I Parameter Information Requirements."

## 10.3.3 Control Statements Requirements

Three types of control statements are used by the data base data set recovery utility. Input control statements are read from the SYSIPT device.

° S statement

This statement describes the data set to be recovered. It is required and must be the first statement

| 1 | 4-11 | 13-20 | 22-23 | 25-28 | 30-80 |
|---|---|---|---|---|---|
| S | dbdname | filename | number of log files | [fact] | [comments] |

| Column | Description |
|---|---|
| 1 | Must be S. |
| 4-11 | Must be the name of the SHISAM, HD indexed, HD randomized, HISAM, HDAM, INDEX, or HIDAM DBD that describes the file to be recovered. Note that for a primary INDEX data base, this is the primary index dbdname and not the HD indexed or HIDAM dbdname required in the DL/I parameter statement described above. |
| 13-20 | Must be the symbolic name of the file to be recovered. It must appear as the DD1 or OVFLW parameter of the DATASET statement or the REF parameter of the ACCESS statement in the referenced DBD, and corresponding DLBL statements must have been provided. |
| 22-23 | This parameter contains the actual number of input log files (from 1 to 99). This is equal to the maximum value nn used on LOGINnn TLBL or DLBL card(s). If the value is omitted or zero, and no LI statements are supplied, and SYS013 is assigned to tape, one log tape input is assumed. This value is overridden if LI control statements are supplied. |
| 25-28 | An optional factor which leads to the number of CIs that may fit into one record block. If image copy input is used, the block size should be the same as or larger than that calculated when the file was dumped, as stated in message DLZ438I.<br><br>**'nnCI'** The block size is calculated to be nn (01...60) times the CI size. A 12-byte prefix for each CI within the block and four bytes for the whole must be considered. If the calculated block size exceeds the upper limit, |

| | |
|---|---|
| | the maximum possible block size (see 'MAX') will be selected. If the calculation is less than 4K the default is used.<br>**'MAX'** The maximum possible block size, depending on the CI size, will be selected. The maximum value for this block size is 31444 bytes.<br>**blank** If nothing is specified a default block size of 4112 will be selected, if the CI size is not greater than 4K. If it is greater than 4096 bytes, the block size selected is one CI, with 16 additional bytes for internal information. For more details, see Chapter 27, "Data Set Image Copy Utility (DLZUDMP0)" in topic 10.1.<br><br>Note that, depending on the CI size and the blocking factor, the additional storage requirement (GETVIS area) may vary from about 12K to 92K. |
| 30-80 | Comments. |

° LI statement

This statement describes the input log files. Up to 99 LI statements may be specified. The first LI statement describes log file LOGIN01; the second LI statement, LOGIN02, etc. The necessary job control statements must also be specified for each log file. If no LI statements are present, all input log files are assumed to be DL/I standard labeled tape log files on logical unit SYS013 with a buffer size of 1024 bytes. The number of input logs is determined from the S statement.

| 1-2 | 11-16 | 18 | 21 | 31-35 |
|---|---|---|---|---|
| LI | [SYSnnn] | U<br>N<br>*R* | *L*<br>U<br>C<br>V<br>S | [nnnnn] |

| **Column** | **Description** |
|---|---|
| 1-2 | Must be LI. |
| 11-16 | Identifies the logical unit for this log file. It must be of the form SYSnnn, where nnn is a valid logical unit assignment. If this parameter is omitted, SYS013 is assumed. |
| 18 | Identifies the rewind option for the input log file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, R is assumed. |
| 21 | Identifies the log file type and may be L, U, C, V, or S.<br><br>L - Specifies standard labeled tape log file (DL/I).<br>U - Specifies unlabeled tape log file (CICS/DOS/VS).<br>C - Specifies standard labeled tape log file (See Note below).<br>V - Specifies VSAM disk log file (DL/I).<br>S - Specifies SAM disk log file (CICS/DOS/VS). |

| | |
|---|---|
| | If this parameter is omitted, L is assumed.<br><br>**Note:** If the output log tape from log print is used as input, the type should be specified as follows:<br><br>No CICS/DOS/VS journals used as input to log print, specify L.<br>If CICS/DOS/VS journal(s) used as input to log print, specify C. |
| 31-35 | Identifies the buffer size in bytes used for a CICS/DOS/VS journal. If U, C, or S is specified in column 21, the buffer size must be at least 1100 bytes, but not more than 32767. The value must be left justified. If this parameter is omitted, 1100 is assumed for U, C, or S types, and 1024 is assumed for L or V types. DL/I log files always have a buffer size of 1024 bytes. |

° DI statement

This statement is used to control positioning for the image copy or HISAM unload input file. Only one DI statement is allowed. The statement format is:

| 1-2 | 11-16 | 18 |
|---|---|---|
| DI | SYSnnn | *U*<br>N<br>R |

| Column | Description |
|---|---|
| 1-2 | Must be DI. |
| 11-16 | Identifies the logical unit for the image copy or HISAM unload input file. It must be of the form SYSnnn, where nnn is valid logical unit assignment. If this parameter is omitted, SYS011 is assumed. |
| 18 | Identifies the rewind option for the image copy or HISAM unload input file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, U is assumed. |

## 10.3.4 Examples

***Example 1:*** This example shows recovery of an HDAM data base. Input is provided from an image copy and change accumulation tape. The data base created by the data base data set image copy utility has a maximum block size.

```
// JOB    RECOVER1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN  SYS011,180
// TLBL   DUMPIN,'hdam image1'
// ASSGN  SYS012,181
// TLBL   CUMIN,'new acc out'
// ASSGN  SYS013,IGN
// DLBL   HDAMDB,'hdam.data.base',,VSAM,CAT='catalog'
// EXEC   DLZRRC00,SIZE=300K
UDR,DLZURDB0,DLIDBD1
S   DLIDBD1  HDAMDB     MAX
/*
/&
```

***Example 2:*** This example shows recovery of an entry sequenced data set (ESDS) with a filename HISDB2 in a HISAM data base named DLIDBD1. Input is provided from an image copy tape and two DL/I disk logs (tape and disk).

```
// JOB    RECOVER2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN  SYS011,180
// TLBL   DUMPIN,'esds image'
// ASSGN  SYS012,IGN
// ASSGN  SYS013,DISK,VOL='volser',SHR
// DLBL   LOGIN01,'dli.log1',,VSAM,DISP=(OLD,KEEP),CAT='catalog'
// EXTENT SYS013,'volser'
// ASSGN  SYS014,182
// TLBL   LOGIN02,'dli.log.copy'
// DLBL   HISDB2,'hisam.data.base.esds',,VSAM,CAT='catalog'
// EXEC   DLZRRC00,SIZE=300K
UDR,DLZURDB0,DLIDBD1
S   DLIDBD1  HISDB2
LI        SYS013    V
LI        SYS014 U  L
DI        SYS011 N
```

```
/*
/&
```

---

---

---

# 10.4 Chapter 30. Log Print Utility (DLZLOGP0)

The log print utility enables you to print the contents of DL/I log files. The primary function of this utility is to help you recover from system failures. Used in conjunction with the backout utility, the information printed by the log print utility may be used to determine which data bases to backout or recover in case backout fails. The utility can be invoked when an abnormal termination or system incident occurs. At such times, it can be used to determine which PSBs have been left open and should be backed out.

The log print utility program runs as a VSE program and does not make any DL/I calls.

The log print utility can print the following types of log files:

- Standard labeled DL/I tape log files.
- Labeled CICS/DOS/VS tape log files.
- Unlabeled CICS/DOS/VS tape log files.
- VSAM DL/I disk log files.
- SAM CICS/DOS/VS disk log files.

Through an input control statement, you can request that all DL/I log records be printed, or that only those records associated with a specific PSB name be printed. By specifying a start date and/or an end date, you can additionally request that only records within a given time range be printed. You can select one of two output formats, keyword or dump. The keyword format individually identifies each field within the record. The dump format shows the record contents in both hexadecimal and character format, 32 bytes per line. You can optionally request the utility copy feature. If specified, the utility creates a new DL/I log tape by copying all log records from the old log file up to the first invalid record. The new log tape can then be used as input to the Backout utility. You can also request that CICS/DOS/VS journal records be printed in dump format. Records may also be selectively printed by DBD name, by CICS/DOS/VS task ID, or by relative block number.

Input to the log print utility, as illustrated in Figure 78 consists of:

1. The log files to be printed.
2. Control statements that specify the type of log files and which log records are to be printed. These control statements are read from SYSIPT and are of three types, LO, LS, and LI.

Messages are issued to both the programmer and the operator for invalid syntax or data selection parameters unless it is evident that the message would be useless to one or the other.
Output of the log print utility consists of:

1. Log records on the SYSLST device.

2. Informational messages on the SYSLST device.

3. Error messages on the SYSLST and SYSLOG devices.

4. Optionally, a new DL/I log tape suitable for use by the Backout utility.

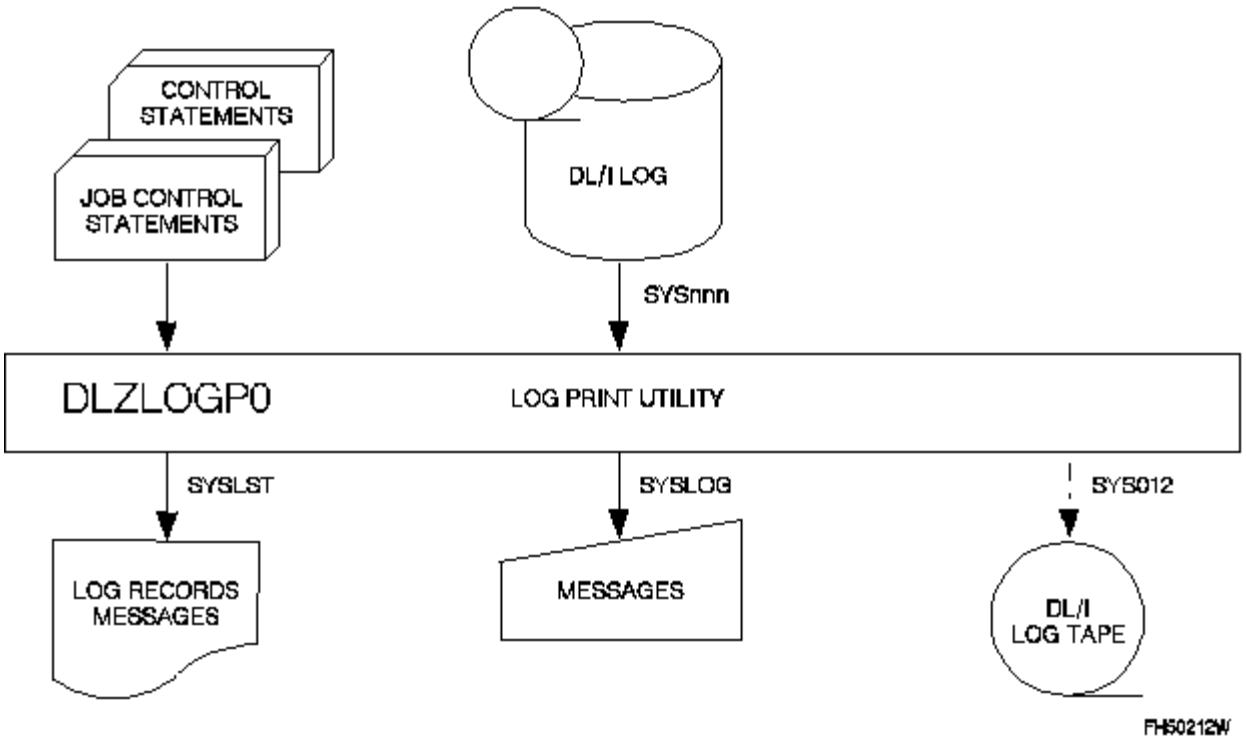



Figure 78. Log Print Utility Program

Subtopics:



---

## 10.4.1 Job Control Statement Requirements

The log print utility executes as a VSE/SP application program and requires the following job control statements:

| | | |
|---|---|---|
| // ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD)<br>[// EXTENT | SYSnnn,cuu<br>LOGINnn ]<br><br>LOGINnn ]<br>extent data] | These statements define the input log file(s) containing the records to be printed. It must be tape with standard labels unless specified differently in an LI control statement. SYSnnn is the logical unit to be assigned, where nnn must be 013 unless specified differently in an LI control statement. LOGINnn is the symbolic name of the input file(s), where nn is a numeric value from 01 to 99. Log files are numbered consecutively from LOGIN01 to LOGIN99. LI control statements must be specified if more than one log file is to be processed. |
| [// ASSGN<br>[// TLBL | SYS012,cuu]<br>LOGOUT ] | These statements define the new output log file. It must be tape with standard labels. LOGOUT is the symbolic name of the output file. These statements are required only if COPY is specified on the LO control statement. |
| // EXEC | DLZLOGP0<br>SIZE=xxxK | Log print utility module. Refer to *DL/I DOS/VS Data Base Administration* for storage requirements. |

**Note:** The above job control statements may contain additional parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

 *© Copyright IBM Corp. 1981, 1989*

## 10.4.2 Control Statement Requirements

This utility uses three types of control statements, LO, LS, and LI.
Input control statements are read from the SYSIPT device.

° LO statement

This statement describes which records are to be printed by the utility and which format is desired. The statement is optional, and if not supplied, default values are assigned as described below. Only one LO statement may be specified.

| 1-2 | 4-10 | 13-21 | 23-31 | 33-39 | 41-44 | 46-50 | 52 |
|---|---|---|---|---|---|---|---|
| LO | [psbname] | [yydddhhmm] | [yydddhhmm] | [*KEYWORD*]<br>[DUMP ] | [COPY] | [CICSD] | *N*<br>U<br>R |

| Column | Description |
|---|---|
| 1-2 | Must be LO. |
| 3 | Must be blank. |
| 4-10 | Optional. This parameter applies to DL/I records only. It identifies a 1- to 7-character PSB name.<br><br>If specified, only log records containing this PSB name will be printed. Because open records (log code X'2F') contain no PSB name, none will be printed.<br><br>If omitted, all log records will be printed unless excluded by another parameter on the LO or LS statement. |
| 11-12 | Must be blank. |
| 13-21 | Optional. This parameter contains a start date of the form yydddhhmm where yy=year (00-99), ddd=day (000-366), hh=hour (00-23), and mm=minute (00-59).<br><br>If specified, only records having a date equal to or later than this start date will be printed. Because only DL/I |

| | |
|---|---|
| | data base records (log codes X'50' and X'51') contain a recognizable date, other type records encountered before the first printable data base record will not be printed.<br><br>If omitted, all log records will be printed unless excluded by another parameter on the LO or LS statement. |
| 22 | Must be blank. |
| 23-31 | Optional. This parameter contains an end date of the form yydddhhmm where yy=year (00-99), ddd=day (000-366), hh=hour (00-23), and mm=minute (00-59).<br><br>If specified, only records having a date equal to or earlier than this end date will be printed. This date must not be earlier than that specified in columns 13-21. Because only DL/I data base records contain a recognizable date, other type records encountered after the last printable data record will not be printed.<br><br>If omitted, all log records will be printed unless excluded by another parameter on the LO or LS statement. |
| 32 | Must be blank. |
| 33-39 | Optional. This parameter applies to DL/I records only. It identifies the printed output format; either KEYWORD or DUMP.<br><br>If KEYWORD is specified, the record contents are printed in keyword format (each field individually identified) as illustrated in Figure 78 in topic 10.4.<br><br>If DUMP is specified, the record contents are printed in dump format (both hexadecimal and character format, 32 bytes per line).<br><br>If this parameter is omitted, KEYWORD is assumed. |
| 40 | Must be blank. |
| 41-44 | Optional. This parameter is used to request the utility copy feature.<br><br>If COPY is specified, the utility copies all log records from the input log file(s) to the new output log tape until an invalid record is encountered. At that time, the utility closes the new log tape file and terminates processing. The new log tape file may then be used as input to the Backout utility. If no invalid record is encountered, a complete copy of the input log file(s) is created.<br><br>If this parameter is omitted, no output log tape is created. |
| 45 | Must be blank. |
| 46-50 | Optional. This parameter is used to request printing of CICS/DOS/VS journal records.<br><br>If CICSD is specified, all CICS/DOS/VS journal records (and any other non-DL/I records) are printed in dump format. These records are printed in addition to the DL/I records. Records outside the time range specified in columns 13-21 and 23-31 will not be printed.<br><br>If this parameter is omitted, only DL/I records will be printed. |
| 51 | Must be blank. |
| 52 | Column 52 identifies the rewind option for the output log file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, N is assumed. |
| 53-80 | Must be blank. |

° LS statement

With this statement you can selectively print only those DL/I records associated with a particular DBD name, CICS/DOS/VS task ID or relative block number. This statement is optional. If it is not supplied, selective log printing is not performed. Only one LS statement may be specified.

```
 1-2 4-10       12-16      18-25
 LS  [dbdname]  [nnnnn]    [xxxxxxxx]
```

| Column | Description |
|---|---|
| 1-2 | Must be LS. |
| 3 | Must be blank. |
| 4-10 | Optional. This parameter identifies a 1- to 7-character DBD name.<br><br>If specified, only open records (log code X'2F') and data base records (log codes X'50' and X'51') containing this DBD name will be printed.<br><br>If omitted, all log records will be printed unless excluded by another parameter on the LS or LO statement. |
| 11 | Must be blank. |
| 12-16 | Optional. This parameter identifies a 1- to 5-digit CICS/DOS/VS task ID. It must be left-justified and contain only decimal digits (0-9).<br><br>If specified, only scheduling records (log code X'08'), termination records (log code X'07'), and data base records (log codes X'50' and X'51') associated with this ID will be printed.<br><br>**Note:** No log records will be printed if the log file is not a CICS/DOS/VS journal or if the appropriate scheduling record is not present.<br><br>If omitted, all log records will be printed unless excluded by another parameter on the LS or LO statement. |
| 17 | Must be blank. |
| 18-25 | Optional. This parameter identifies a 1- 8-digit RBN (relative block number). It must be left-justified and contain only hexadecimal digits (0-9, A-F).<br><br>If specified, only data base records (log codes X'50' and X'51') containing this RBN will be printed.<br><br>If omitted, all log records will be printed unless excluded by another parameter on the LS or LO statement. |
| 26-80 | Must be blank. |

° LI statement

This statement describes the input log files. Up to 99 LI statements may be specified. The first LI statement describes log file LOGIN01; the second LI statement, LOGIN02, etc. The necessary job control statements must also be specified for each log file. Multiple log files must be in chronological order, that is, LOGIN01 must have been created before LOGIN02; LOGIN02 before LOGIN03, etc. If this statement is omitted, one DL/I standard labeled tape log file on logical unit SYS013 with a buffer size of 1024 bytes is assumed. All columns between parameters must be blank.

| 1-2 | 11-16 | 18 | 21 | 31-35 |
|---|---|---|---|---|
| LI | [SYSnnn] | U<br>N<br>R | *L*<br>U<br>V<br>S | [nnnnn] |

| Column | Description |
|---|---|
| 1-2 | Must be LI. |
| 3-10 | Must be blank. |
| 11-16 | Optional. This parameter identifies the logical unit for this log file. If specified, it must be of the form SYSnnn, where nnn is a valid logical unit assignment. If omitted, SYS013 is assumed. |
| 17 | Must be blank. |
| 18 | Column 18 identifies the rewind option for the input log file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, U is assumed for unlabeled tape and R is assumed for labeled tape. |
| 19-20 | Must be blank. |
| 21 | Optional. This parameter identifies the log file type. If specified, it must be L, U, V, or S where:<br><br>L - specifies standard labeled tape log file (DL/I).<br>U - specifies unlabeled tape log file (CICS/DOS/VS).<br>V - specifies VSAM disk log file (DL/I).<br>S - specifies SAM disk log file (CICS/DOS/VS).<br><br>If omitted, L is assumed. |
| 22-30 | Must be blank. |
| 31-35 | Optional. This parameter identifies the buffer size in bytes used for a CICS/DOS/VS journal. If U or S is specified in column 21, the buffer size must be at least 1100 bytes, but not more than 32767. This value must be left justified. If this parameter is omitted, 1100 is assumed for U or S types, and 1024 is assumed for L or V types. DL/I log files always have a buffer size of 1024 bytes. |

| | |
|---|---|
| 36-80 | Must be blank. |

---

## 10.4.3 Examples

***Example 1:*** This example prints the contents of two DL/I VSAM disk log files and one standard labeled tape log file. Only records associated with PSB name DLIPSB1 occurring on or after February 1, 1988 will be printed. The records will be printed in dump format.

---

```
// JOB LOGP1
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN SYS013,DISK,VOL='volser',SHR
// DLBL LOGIN01,'dli.log1',0,VSAM,CAT='catalog'
// EXTENT SYS013,'volser'
// ASSGN SYS014,DISK,VOL='volser',SHR
// DLBL LOGIN02,'dli.log2',0,VSAM,CAT='catalog'
// EXTENT SYS014,'volser'
// ASSGN SYS015,182
// TLBL LOGIN03,'dli.log3'
// EXEC DLZLOGP0,SIZE=100K
LO DLIPSB1  880320000           DUMP
LI                  V
LI        SYS014    V
LI        SYS015 U  L
/*
/&
```

---

***Example 2:*** This example prints the contents of a standard labeled DL/I tape log file and copies the contents to a new tape log file until an invalid record is encountered. The new tape log may be used as input to the Backout utility. The log records printed will be in keyword format.

---

```
// JOB LOGP2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN SYS013,180
// TLBL LOGIN01,'log.file'
// ASSGN SYS012,181
// TLBL LOGOUT,'dli.log.copy'
// EXEC DLZLOGP0,SIZE=100K
LO                              KEYWORD COPY
/*
/&
```

---

***Example 3:*** This example prints the contents of an unlabeled CICS/DOS/VS tape journal whose buffer size is 4096. Only data base log records that are associated with CICS/DOS/VS task ID 27 and relative block number 3E and occurred before January 1, 1988 will be printed.

---

```
// JOB LOGP3
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN SYS013,180
// TLBL LOGIN01,'logfile'
// EXEC DLZLOGP0,SIZE=100K
LO                  873652359
LS         27    3E
LI        SYS013    U         4096
/*
/&
```

---

```
     DL/I DOS/VS  DLZLOGP0  -  LOG PRINT UTILITY              INPUT CONTROL        DATE: 12/06/88
TIME: 11:52:27  PAGE    1
     LO                            KEYWORD
     LI        SYS013    V
```

---

```
     DL/I DOS/VS  DLZLOGP0  -  LOG PRINT UTILITY              FILENAME = LOGIN01   DATE: 12/06/88
TIME: 11:52:27  PAGE    2
     SCHEDULING RECORD
     SEQNO = 00000001, PSBNAME = AHI30KAP, TASKID = 10
     OPEN RECORD, DSORG = ESDS
     SEQNO = 00000002, DMBNAME = HI30KA@D, FILENAME = HI30KA
     OPEN RECORD, DSORG = KSDS
     SEQNO = 00000003, DMBNAME = HI30KAPD, FILENAME = HI30KAP
     OPEN RECORD, DSORG = KSDS
     SEQNO = 00000004, DMBNAME = HI30KASD, FILENAME = HI30KAS
     NEW DATA BASE RECORD, DSORG = ESDS/HD, CALL = ISRT         , TYPE = PHYSICAL INSERT
     SEQNO = 00000005, PSBNAME = AHI30KAP, TASKID = 01, DMBNAME = HI30KA@D, DATE = 88341, TIME =
114518
     SEQST = 0000, CCODE =   , OFFSET = 267E, DSID = 01
     DATAID =  00000002                                                                  *....*
     DATA =    02000000 F05A0000 F03CC2F1 F1F1F1F1  F1F1F1F1 F1F1F1F1 F1F1F1F1 F1F1
*....0...0.B1111111111111111*
     FSEOFF = 0000, FSE = 269C0000, FSEOFF = 269C, FSE = 0000515C
     NEW DATA BASE RECORD, DSORG = ESDS/HD, CALL = ISRT         , TYPE = POINTER MAINTENANCE
     SEQNO = 00000006, PSBNAME = AHI30KAP, TASKID = 01, DMBNAME = HI30KA@D, DATE = 88341, TIME =
114518
     SEQST = 0001, CCODE =   , OFFSET = 003E, DSID = 01
     DATAID =  00000002                                                                  *....*
     DATA =    0001167E 0000F05A
*......0.*
     NEW DATA BASE RECORD, DSORG = ESDS/HD, CALL = ISRT         , TYPE = POINTER MAINTENANCE
     SEQNO = 00000007, PSBNAME = AHI30KAP, TASKID = 01, DMBNAME = HI30KA@D, DATE = 88341, TIME =
```

```
114518
     SEQST = 0002, CCODE =   , OFFSET = 0060, DSID = 01
     DATAID =  00000002                                                    *....*
     DATA =    0001167E 0000F03C
*......0.*
     TERMINATION RECORD
     SEQNO = 00000008, PSBNAME = AHI30KAP, TASKID = 10
     DLZ417I END OF FILE ON FILENAME - LOGIN01
     DLZ418I TOTAL FILE RECORDS READ = 8, RECORDS PRINTED = 8
     DLZ339I NO ERRORS DETECTED, LOG PRINT SUCCESSFUL
     DLZ420I TOTAL JOB RECORDS READ = 8, RECORDS PRINTED = 8
```

Figure 79. Example of KEYWORD Format Output

 *© Copyright IBM Corp. 1981, 1989*

# 10.4.4 Keyword Record Format to DSECT Mapping

**Scheduling Record**

| Keyword | Value | DSECT Field |
|---|---|---|
| SCHEDULING RECORD | -- | LOGFLAG (X'08') |
| SEQNO= | xxxxxxxx | Last 4 bytes of record |
| PSBNAME= | aaaaaaaa | PSBNAME |
| TASKID= | xx | SCHDCODE |
| CICSID= | nnnnn | CICSID (online only) |

**Termination Record**

| Keyword | Value | DSECT Field |
|---|---|---|
| TERMINATION RECORD | -- | ALLOGFLAG (X'07') |
| SEQNO= | xxxxxxxx | Last 4 bytes of record |
| PSBNAME= | aaaaaaaa | ALPSBNAM |
| TASKID= | xx | ALIDCODE |
| CICSID= | nnnnn | CICSID (online only) |
| ACCT= | xxxxxxxx (10 times) | TSKSTAT (online only) |

**Open Record**

| Keyword | Value | DSECT Field |
|---|---|---|
| OPEN RECORD | -- | DLOGCODE (X'2F') |
| DSORG= | KSDS | DLOGFLG1 (X'04') |
| | ESDS | DLOGFLG1 (X'00') |
| | {/HISAM} | DDSID (2) |
| SEQNO= | xxxxxxxx | Last 4 bytes of record |
| DMBNAME= | aaaaaaaa | DDBDNAME |
| FILENAME= | aaaaaaaa | DPGMNAME |

**Checkpoint Record**

| Keyword | Value | DSECT Field |
|---|---|---|
| CHECKPOINT RECORD | -- | CHKPCODE (X'41') |
| SEQNO= | xxxxxxxx | Last 4 bytes of record |
| PSBNAME= | aaaaaaaa | CHKPPSB |
| CHKID= | aaaaaaaa | CHKPID |

**Data Base Record**

| | | |
|---|---|---|

| Keyword | Value | DSECT Field |
|---|---|---|
| NEW DATA BASE RECORD | -- | DLOGCODE (X'50') |
| OLD DATA BASE RECORD | -- | DLOGCODE (X'51') |
| DSORG= | ESDS | DLOGFLG2 (bit 5=0) |
| | KSDS | DLOGFLG2 (bit 5=1) |
| | /HS | DLOGFLG2 (bit 6=0) |
| | /HD | DLOGFLG2 (bit 6=1) |
| CALL= | REPL | DLOGFLG3 (bit 0=1) |
| | DLET | DLOGFLG3 (bit 1=1) |
| | ISRT | DLOGFLG3 (bit 2=1) |
| | /BACKOUT | DLOGFLG3 (bit 5=1) |
| TYPE= | INDEX MAINTENANCE | DLOGFLG2 (bit 0=1) |
| | NEW BLOCK CALL | DLOGFLG2 (bit 7=1) |
| | LAST RECORD OF CHANGE GROUP | DLOGFLG2 (bit 4=1) |
| | COUNTER MAINTENANCE | DLOGFLG2 (bits 1-3=111) |
| | POINTER MAINTENANCE | DLOGFLG2 (bits 1-3=000) |
| | LOGICAL DELETE | DLOGFLG2 (bits 1-3=110) |
| | PHYSICAL INSERT | DLOGFLG2 (bits 1-3=100) |
| | PHYSICAL DELETE | DLOGFLG2 (bits 1-3=010) |
| | PHYSICAL REPLACE | DLOGFLG2 (bits 1-3=001) |
| SEQNO= | xxxxxxxx | Last 4 bytes of record |
| PSBNAME= | aaaaaaaa | DPGMNAME |
| TASKID= | xx | DCCODE (first byte) |
| DMBNAME= | aaaaaaaa | DDBDNAME |
| DATE= | yyddd | DDATE |
| TIME= | hhmmss | DTIME |
| SEQST= | xxxx | DSEQ |
| CCODE= | aa | DCCODE (second byte) |
| OFFSET= | xxxx | DOFFSET |
| DSID= | xx | DDSID |
| DATAID= | (Dump Format) | DDATAID |
| DATA= | (Dump Format) | DDATA |
| FSEOFF= | xxxx | DFSEOFF |
| FSE= | xxxxxxxx | DFSE |

# 10.5 Chapter 31. Data Base Backout Utility (DLZBACK0)

The data base backout utility removes (backs out) the effects of any abnormally terminated program that accessed data bases using DL/I calls. If the program was operating in a DL/I batch partition, all data base changes made by the program from the time it began processing until it terminated abnormally are backed out. If the program issued checkpoint calls, only the changes made since the last checkpoint are backed out. If the program was operating in a CICS/DOS/VS-DL/I teleprocessing partition, only the changes made from the last synchronization point or scheduling call until the loss of the online system are backed out. If the program terminated (either normally or abnormally) before the CICS/DOS/VS-DL/I online system terminated, its changes are not backed out by this utility. CICS/DOS/VS dynamic backout may have backed them out, however.

If the data bases were not closed by DL/I prior to the abnormal termination, the user must execute the VSAM access method services command VERIFY to update the VSAM catalog for each file. If this is not done, an error will occur when the DL/I system attempts to open the data bases.

A log file is created to reflect the backout history. The log file must be included in any data base recovery attempted for the data bases involved in the backout. The output log file created by the backout utility must not, however, be used as input to any subsequent backout attempt.

It is recommended that either the data base data set image copy utility or HISAM reorganization unload/reload utilities be executed immediately after the data base backout utility for all the affected files.

Input to the data base backout utility as illustrated in Figure 80 consists of:

1. One PSB and one or more DMBs loaded from a 'DL/I user lib' 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 file(s) for the DL/I job execution which abnormally terminated.

Output from the data base backout utility as illustrated in Figure 80 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 file reflecting the changes made to the data base during the data base backout utility execution. This log file as well as the input log must be used as input for any future recovery utility execution against the above data bases, unless the image dump utility or HISAM reorganization unload/reload utilities (HISAM, SHISAM, or INDEX) are executed after the data base backout utilities.

3. Messages on the SYSLST and SYSLOG devices.

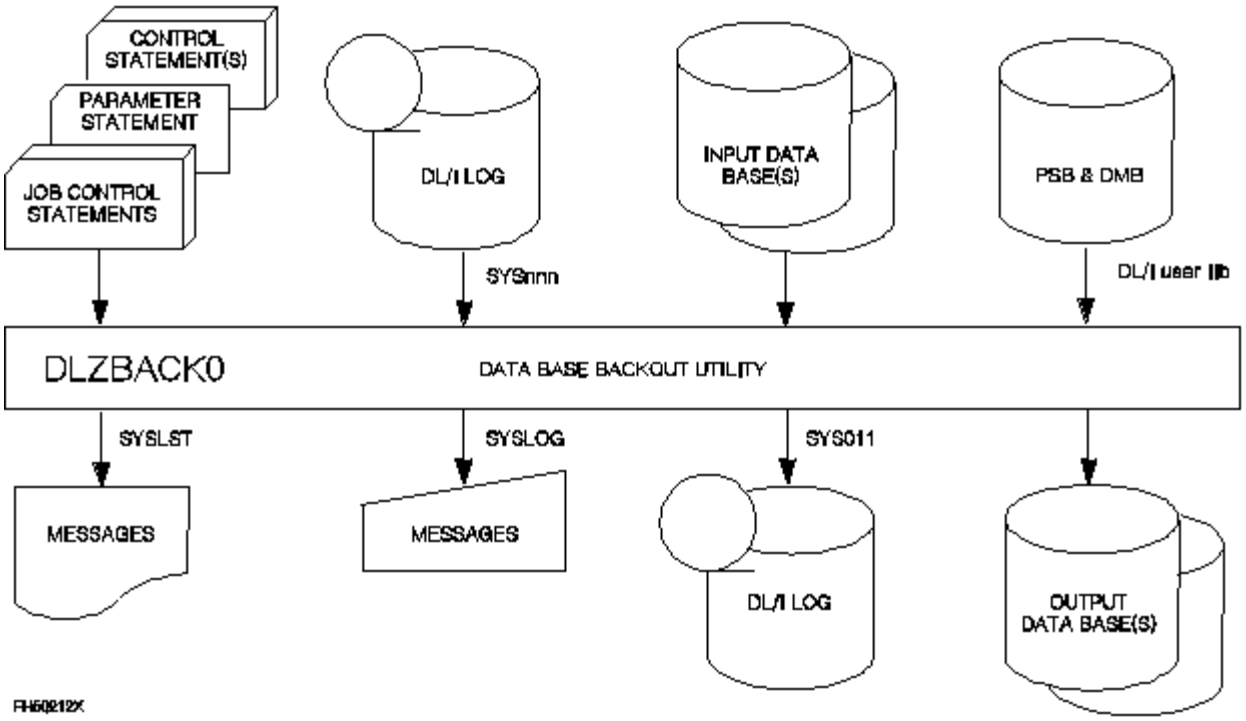



Figure 80. Data Base Backout Utility

Subtopics:

- 10.5.1 Job Control Statement Requirements
- 10.5.2 Parameter Statement Requirements
- 10.5.3 Control Statement Requirements
- 10.5.4 Examples

---

## 10.5.1 Job Control Statement Requirements

The following job control statements are required:

| | | |
|---|---|---|
| // ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD) | SYSnnn,cuu<br>LOGINnn]<br><br>LOGINnn] | These statements define the input log file(s). It must be tape with standard labels unless specified differently in an LI control statement. SYSnnn is the logical unit to be assigned, where nnn must be 012 unless specified differently in an LI control statement. LOGINnn is the symbolic name of the input file, where nn is a numeric value from 01 to 99. Log files are numbered consecutively from LOGIN01 to LOGIN99. LI control statements must be specified if more than one input log file is to be processed. Multi-volume files and 7-track tape files with data conversion cannot be read backward because of a DOS/VSE restriction. However, multi-volume files can be processed by describing each volume as a separate file. |
| [// ASSGN<br>[// TLBL(Tape)<br>or<br>[// DLBL(DASD) | SYS011,cuu]<br>LOGOUT]<br><br>{DSKLOG1}]<br>{DSKLOG2}] | These statements define the output log file. It may be tape with standard labels or a VSAM disk file. |
| // DLBL | filename | This statement defines the symbolic name of the data set to be backed out. One statement must be present for each file that appears in the DBD describing the data base.<br><br>For HD randomized and HD indexed data bases, use the filenames defined in the DD1 operand of DATASET statements and the REF operand of ACCESS statements (if any).<br><br>For HDAM, HIDAM, INDEX, and SHISAM data bases, use the filename defined in the DD1 operand of DATASET statements.<br><br>For HISAM data bases, use the filenames specified in the DD1 and OVFLW operand of DATASET statements. |
| // EXEC | DLZRRC00,<br>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 parameters. Refer to *VSE/Advanced Functions System Control Statements* for more information.

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 10.5.2 Parameter Statement Requirements

The data base backout utility is executed as an application program under the control of DL/I. The following parameter data, beginning in column 1, must be entered either via SYSIPT or SYSLOG:

```
DLI,DLZBACK0,psbname[,{buf}][,HDBFR=][,HSBFR=][,TRACE=][,ASLOG=][,LOG=]
                      {1  }
```

**psbname**
is the name of the PSB that references the data bases to be backed out. This must be the same PSB name specified for the DL/I application program which terminated abnormally.

The keyword parameters buf, HDBFR, HSBFR, TRACE, ASLOG, and LOG are optional parameters that may also be entered in the parameter statement. These parameters have the same usage as for application programs and are described in Chapter 15, "Executing DL/I Programs," under "DL/I Parameter Information Requirements." The LOG parameter must be specified if the output log is to be a VSAM disk file.

 *© Copyright IBM Corp. 1981, 1989*

## 10.5.3 Control Statement Requirements

One type of control statement is used by the Backout Utility. Input control statements are read from the SYSIPT device.

° LI statement

This statement describes the input log files. Up to 99 LI statements may be specified. The first LI statement describes log file LOGIN01; the second statement, LOGIN02, etc. The necessary job control statements must also be specified for each log file. Multiple log files must be in reverse chronological order. That is, LOGIN01 must have been created after LOGIN02; LOGIN02 after LOGIN03, etc. If this statement is omitted, one DL/I standard labeled tape log file on logical unit SYS012 with a buffer size of 1024 bytes is assumed.

| 1-2 | 11-16 | 18 | 21 | 31-35 |
|---|---|---|---|---|
| LI | [SYSnnn] | U<br>N<br>R | *L*<br>U<br>V | [nnnnn] |

| | |
|---|---|
| 1-2 | Must be LI. |
| 11-16 | Identify the logical unit for this log file. It must be of the form SYSnnn, where nnn is a valid logical unit assignment. If this parameter is omitted, SYS012 is assumed. |
| 18 | Identifies the rewind option for the input log file.<br><br>U - Rewind and unload;<br>N - No rewind;<br>R - Rewind only.<br><br>If this parameter is omitted, N is assumed for labeled tape and R is assumed for unlabeled tape. |
| 21 | Identifies the log file type.<br><br>L - Specifies standard labeled tape log file (DL/I).<br>U - Specifies unlabeled tape log file (CICS/DOS/VS).<br>V - Specifies VSAM disk log file (DL/I).<br><br>If this parameter is omitted, L is assumed. |
| 31-35 | Identifies the buffer size in bytes used for a CICS/DOS/VS journal. If U is specified in column 21, the buffer size must be at least 1100 bytes, but not more than 32767. The value must be left justified. If this parameter is omitted, |

| | |
|---|---|
| | 1100 is assumed for U types, and 1024 for L or V types.<br>DL/I log files always have a buffer size of 1024 bytes. |

---

---

## 10.5.4 Examples

***Example 1:*** This example backs out the data base changes made by the program which uses PSB DLIPSB1. Both the input log and the output log are standard labeled DL/I tape files.

```
// JOB BACKOUT
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN SYS012,180
// TLBL LOGIN01,'login.file'
// ASSGN SYS011,181
// TLBL LOGOUT,'logout.file'
// DLBL DB1,'data.base1',,VSAM,CAT='catalog'
// DLBL DB2,'data.base2',,VSAM,CAT='catalog'
// EXEC DLZRRC00,SIZE=400K
DLI,DLZBACK0,DLIPSB1,1
LI        SYS012 U  L
/*
/&
```

***Example 2:*** This example backs out the data base changes made by the program that uses PSB DLIPSB2. The log input consists of two DL/I VSAM disk log files. dli.log2 was created after dli.log1 and must be processed first by the utility. The new output log is also a VSAM disk file.

```
// JOB BACKOUT2
// LIBDEF *,SEARCH=('DL/I user lib,DL/I prod lib')
// ASSGN SYS014,DISK,VOL='volser',SHR
// DLBL LOGIN01,'dli.log2',0,VSAM,DISP=(OLD,KEEP),CAT='catalog'
// EXTENT SYS014,'volser'
// ASSGN SYS015,DISK,VOL='volser',SHR
// DLBL LOGIN02,'dli.log1',0,VSAM,DISP=(OLD,KEEP),CAT='catalog'
// EXTENT SYS015,'volser'
// ASSGN SYS011,DISK,VOL='volser',SHR
// DLBL DSKLOG1,'dsklog.out',,VSAM,RECORDS=20,RECSIZE=1024,*
               DISP=(NEW,KEEP),CAT='catalog'
// EXTENT SYS011,'volser'
// DLBL DB3,'data.base3',,VSAM,CAT='catalog'
// EXEC DLZRRC00,SIZE=400K
DLI,DLZBACK0,DLIPSB2,1,LOG=(DISK1)
LI        SYS014    V
LI        SYS015    V
/*
```

```
/&
```

---

---

 *© Copyright IBM Corp. 1981, 1989*

---

# 11.0 Part 11. Extracting DL/I Data to SQL/DS

The ISQL EXTRACT DEFINEs utility provides an ease-of-use facility for automatic creation of an ISQL routine for EXTRACT DEFINE commands. This routine is created from the information contained in the SQL/DS tables for DL/I Documentation Aid created during ACBGEN. These tables contain DBD and PSB information for a DL/I data base.

Based upon the information in these SQL/DS tables, the necessary EXTRACT DEFINE PCB, SEGMENT, and FIELD commands will be created and inserted into the SQL/DS ROUTINE table. The EXTRACT DEFINE FIELD commands will be created based on the definitions provided in the SENFLD/VIRFLD statements specified in the PSB. If no SENFLD/VIRFLD statements were given, then the information defined by the FIELD statements for the segment as specified in the PSB's associated DBD is used. The PSBFIELDDATA table is updated to include all the valid data (start position, data type, etc.) that is calculated during the processing of the field data.

A listing of the ISQL routine is provided by the utility as it is created. Since DL/I does not require that all fields within a segment be named and, if you want to extract the complete segment, the ISQL routine created by this utility can be updated through SQL/DS or ISQL. The listing of the routine which is provided can be used to assist in planning any changes.

The ISQL EXTRACT DEFINEs utility cannot be run unless SQL/DS, ISQL, and EXTRACT are installed on the VSE system.

### Performing ISQL EXTRACT DEFINEs Interactively

The ISQL EXTRACT DEFINEs utility can be generated interactively through the Interactive Utility Generation (IUG) facility. IUG automatically generates DL/I utility job streams in their correct order of execution, each having its own job control statement and parameters. In addition, IUG "remembers" information from one part of a job stream to another, thus reducing the amount of data you must enter. For additional information on generating utility job streams interactively, see *DL/I DOS/VS Interactive Resource Definition*.

Subtopics:

# 11.1 Chapter 32. ISQL EXTRACT DEFINEs Facility (DLZEXDFP)

Subtopics:



# 11.1.1 Installing the ISQL EXTRACT DEFINES Facility

A job stream similar to the following example must be run to install the ISQL EXTRACT DEFINEs facility:

```
// JOB PREP DLZEXDF
// LIBDEF *,SEARCH='(DL/I prod lib,SQL prodlib)'
// EXEC PROC='sqllabel'
// EXEC PGM=ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,PROGNAME=ARIPRPA/PREPNAM*
              E=DLZEXDFP,USERID=SQLDBA/'sqldbapw''
READ MEMBER DLZEXDF
/*
/&
```

This job stream creates the SQL/DS access module for program DLZEXDF.

Subtopics:

- 11.1.1.1 Modifying the Job Stream for Your Environment

## 11.1.1.1 Modifying the Job Stream for Your Environment

The following procedures should be followed to modify the job stream to meet your local needs:

° The job control statements are set to invoke the job in a SQL/DS single-partition mode. The appropriate changes must be made if SQL/DS multi-partition mode is used.

° The EXEC PROC= statement should be updated to include your Procedure Name for the SQL/DS data base identification statements.

° The installation DBA userid of 'SQLDBA' and DBA password of 'SQLDBAPW' are used in the EXEC job control statement.

- The userid of 'SQLDBA' is required.

- The password should be changed to the current password for 'SQLDBA'.

° Since the module's object code and phase are already cataloged in the DL/I libraries at DL/I installation time, output to SYSPCH and SYSLST can be suppressed using an // UPSI job control statement (SQL/DS Release 1) or the PRINT/NOPRINT and PUNCH/NOPUNCH options (SQL/DS Release 2). If output is desired, this job control statement should be removed. For SQL/DS, Release 2, the PRINT/NOPRINT and PUNCH/NOPUNCH options should be used instead of the UPSI byte settings.

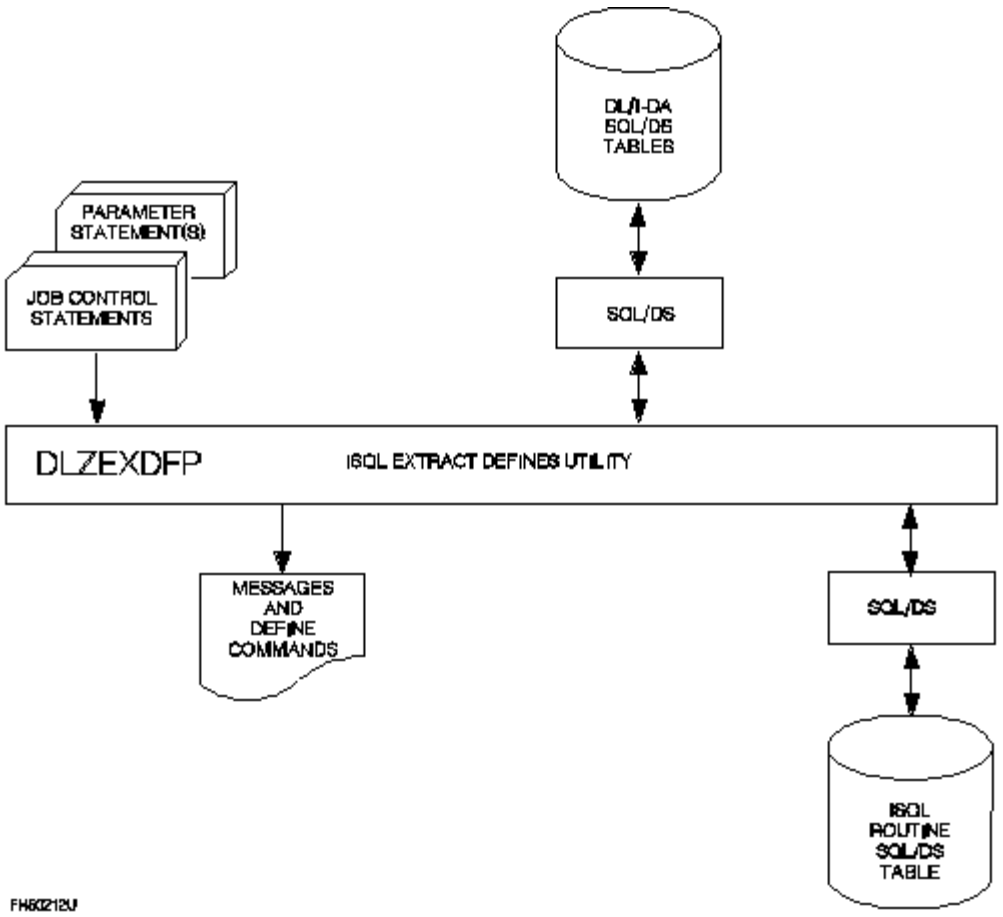



Figure 81. ISQL EXTRACT DEFINEs Utility

---



---

# 11.1.2 Job Control Statement Requirements

The ISQL EXTRACT DEFINES utility is executed as a batch SQL/DS application program. SQL/DS may be running in either single or multi-partition mode.

The following are the job control statements for execution in multiple partition mode:

| | | |
|---|---|---|
| // EXEC | PGM=DLZEXDFP,<br>SIZE=AUTO | This statement identifies the program name of the ISQL EXTRACT DEFINEs utility module and causes it to execute. It is recommended that you specify SIZE=AUTO. |
| Put your parameter statements here. | | |
| /*<br>/& | | |

The following are the job control statements for execution in single partition mode:

| | | |
|---|---|---|
| // EXEC | PROC='sqldbs' | This statement executes the procedure that contains the identification statements for the SQL/DS databases. |
| // EXEC | PGM=ARISQLDS,<br>SIZE=AUTO,<br>PARM='SYSMODE=S,<br>PROGNAME=DLZEXDFP' | This statement invokes SQL/DS and passes to SQL/DS the ISQL EXTRACT DEFINEs module name. Once the EXTRACT DEFINEs module receives control, it accesses SQL/DS in the same way as if executed in multi-partition mode. The SIZE=AUTO is required. |
| Put your parameter statements here. | | |
| /*<br>/& | | |

# 11.1.3 Parameter Statement Requirements

The format of the extract utility parameter statement is:

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

**PSBNAME={psbname }**
**{(psbname,pcbnumber)}**
identifies the PSB that is available for extract operations. psbname is the name of the PSB; pcbnumber is the number of the PCB within the PSB. If pcbnumber 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 (SQLDBA) and password assigned for SQL/DS CONNECT authority.

**REPLACE**
specifies that this ISQL routine is to replace any ISQL routine that already exists in the ISQL ROUTINE table that matches the pcbname. If REPLACE is not specified, this is the first request to create the ISQL EXTRACT DEFINEs ROUTINE in the SQL/DS ROUTINE table.

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. Characters following a blank in 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 job stream.

---

# 11.1.4 ISQL EXTRACT DEFINEs Utility Examples

As an example of the use of the ISQL EXTRACT DEFINEs utility, this section shows the commands and responses in extracting data from the DL/I sample application data base. This DL/I data base is distributed with DL/I and is described in the *DL/I DOS/VS Guide for New Users*.

The examples in this section use the Physical Customer data base created by the DL/I sample application. The physical data base structure for the Customer data base is shown in Figure 82. This data base has three access paths; Figure 83 shows the segment hierarchy for each access path.

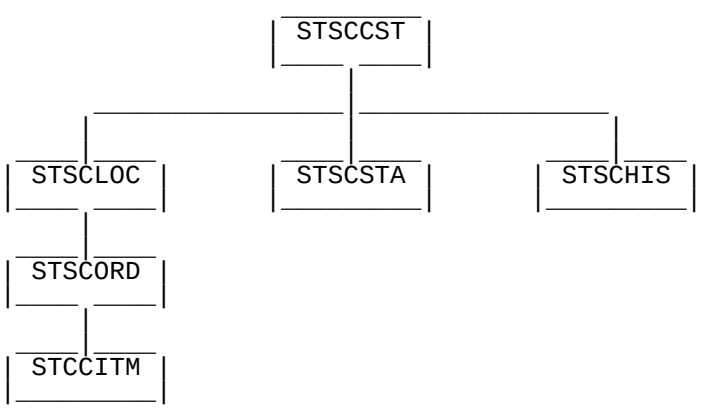



Figure 82. Customer Data Base Structure

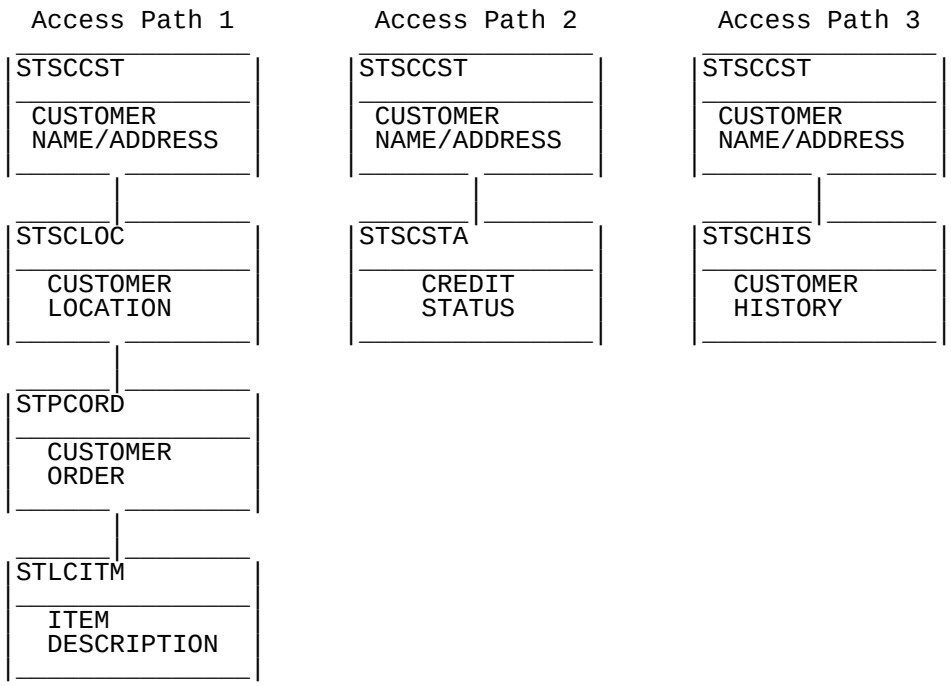


Figure 83. Segment Hierarchy for Customer Data Base Access Paths

The following examples show how to:

- Create an ISQL routine containing EXTRACT DEFINEs statements.
- Display the contents of the ISQL routine.
- Create the SQL/DS target table to receive the extracted data.
- Extract the DL/I data using the ISQL routine.
- Display the data after it has been extracted.

For additional information on the SQL/DS commands and facilities discussed in this section, refer to *SQL/Data System Planning and Administration for VSE*, *SQL/Data System Terminal User's Reference for VSE*, or other SQL/DS publications.

Subtopics:



---

# 11.1.4.1 Creating an ISQL EXTRACT DEFINEs Routine

Figure 84 shows a sample job stream for execution of the ISQL EXTRACE DEFINEs utility. When this job stream is executed, the Extract Defines program, DLZEXDFP, does the following:

- Selects information for the specified PCB (the second PCB in PSB STBICAP) from the DL/I SQL/DS tables created, during ACBGEN, by the DL/I Documentation Aid facility;

- Creates the DEFINE PCB, SEGMENT, and FIELD commands for the PCB; and

- Inserts the created commands into the ISQL ROUTINE table named STBICAP2.

```
// JOB EXTRACT DEFINES FOR STBICAP PCB 2
// LIBDEF *,SEARCH='DL/I user lib, DL/I prod lib, SQL prod lib'
// EXEC PROC='sqldbs'
// EXEC PGM=DLZEXDFP,SIZE=AUTO
EXTRACT PSBNAME=(STBICAP,2),                                          *
        PCBNAME=STBICAP2,                                             *
        DLIPROC=SAMPLE,                                               *
        USERID=SQLDBA/sqldbapw,                                       *
        REPLACE
/*
/&
```

Figure 84. ISQL EXTRACT DEFINEs Utility Job Stream

When the ISQL EXTRACT DEFINEs utility is run, a listing is generated that shows the created commands. The listing produced by the above job stream is provided in Figure 85. Note that this information contains definitions for each of the access paths for the data base. The numbers in the left column are the ISQL Routine sequence numbers.

```
      DL/I DOS/VS  DLZEXDFP  -  MESSAGES AND CONTROL STATEMENTS                    DATE: 11/14/88
TIME: 10:58:49  PAGE    1
```

```
                EXTRACT PSBNAME=(STBICAP,2),                                    *
                        PCBNAME=STBICAP2,                                       *
                        DLIPROC=SAMPLE,                                         *
                        USERID=SQLDBA/SQLDBAPW,                                 *
                        REPLACE


       00000    DEF PCB NAME=STBICAP2,PSB=(STBICAP,2  ),PROC=SAMPLE
       00010    DEF SEGMENT NAME=STSCCST ,PCB=STBICAP2,PARENT=0       ,BYTES=00106
       00020    DEF FIELD NAME=(STQCCNO ,SEQ  ),SEGM=STSCCST ,PCB=STBICAP2, -
       00030                TYPE=C,START=00001,BYTES=006
       00040    DEF FIELD NAME=(STFCCA1 ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
       00050                TYPE=C,START=00007,BYTES=025
       00060    DEF FIELD NAME=(STUCCNM ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
       00070                TYPE=C,START=00007,BYTES=025
       00080    DEF FIELD NAME=(STFCCA2 ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
       00090                TYPE=C,START=00057,BYTES=025
       00100    DEF FIELD NAME=(STFCCA3 ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
       00110                TYPE=C,START=00082,BYTES=025
       00120    DEF SEGMENT NAME=STSCLOC ,PCB=STBICAP2,PARENT=STSCCST ,BYTES=00106
       00130    DEF FIELD NAME=(STQCLNO ,SEQ  ),SEGM=STSCLOC ,PCB=STBICAP2, -
       00140                TYPE=C,START=00001,BYTES=006
       00150    DEF FIELD NAME=(STFCLNM ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
       00160                TYPE=C,START=00007,BYTES=025
       00170    DEF FIELD NAME=(STFCLA1 ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
       00180                TYPE=C,START=00032,BYTES=025
       00190    DEF FIELD NAME=(STFCLA2 ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
       00200                TYPE=C,START=00057,BYTES=025
       00210    DEF FIELD NAME=(STFCLA3 ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
       00220                TYPE=C,START=00082,BYTES=025
       00230    DEF SEGMENT NAME=STPCORD ,PCB=STBICAP2,PARENT=STSCLOC ,BYTES=00055
       00240    DEF FIELD NAME=(STQCODY ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00250                TYPE=C,START=00001,BYTES=002
       00260    DEF FIELD NAME=(STQCODT ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00270                TYPE=C,START=00001,BYTES=006
       00280    DEF FIELD NAME=(STQCODN ,SEQ  ),SEGM=STPCORD ,PCB=STBICAP2, -
       00290                TYPE=C,START=00001,BYTES=012
       00300    DEF FIELD NAME=(STQCODM ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00310                TYPE=C,START=00003,BYTES=002
       00320    DEF FIELD NAME=(STQCODD ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00330                TYPE=C,START=00005,BYTES=002
       00340    DEF FIELD NAME=(STQCONO ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00350                TYPE=C,START=00007,BYTES=006
       00360    DEF FIELD NAME=(STFCORF ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00370                TYPE=C,START=00013,BYTES=025
       00380    DEF FIELD NAME=(STFCOIC ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00390                TYPE=C,START=00038,BYTES=006
       00400    DEF FIELD NAME=(STFCOAM ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
       00410                TYPE=C,START=00044,BYTES=012
       00420    DEF SEGMENT NAME=STCCITM ,PCB=STBICAP2,PARENT=STPCORD ,BYTES=00038
       00430    DEF FIELD NAME=(STKCIIN ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
       00440                TYPE=C,START=00001,BYTES=006
       00450    DEF FIELD NAME=(STQCILI ,SEQ  ),SEGM=STCCITM ,PCB=STBICAP2, -


       DL/I DOS/VS  DLZEXDFP  -  MESSAGES AND CONTROL STATEMENTS                DATE: 11/14/88
 TIME: 10:58:49  PAGE    2


       00460                TYPE=C,START=00007,BYTES=002
       00470    DEF FIELD NAME=(STFCIQO ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
       00480                TYPE=C,START=00009,BYTES=006
       00490    DEF FIELD NAME=(STFCIQS ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
       00500                TYPE=C,START=00015,BYTES=006
       00510    DEF FIELD NAME=(STFCIQB ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
       00520                TYPE=C,START=00021,BYTES=006
       00530    DEF FIELD NAME=(STFCIAM ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
       00540                TYPE=C,START=00027,BYTES=012
       00550    DEF SEGMENT NAME=STSCSTA ,PCB=STBICAP2,PARENT=STSCCST ,BYTES=00024
       00560    DEF FIELD NAME=(STFCSCL ,NOSEQ),SEGM=STSCSTA ,PCB=STBICAP2, -
       00570                TYPE=C,START=00001,BYTES=012
       00580    DEF FIELD NAME=(STFCSBL ,NOSEQ),SEGM=STSCSTA ,PCB=STBICAP2, -
       00590                TYPE=C,START=00013,BYTES=012
       00600    DEF SEGMENT NAME=STSCHIS ,PCB=STBICAP2,PARENT=STSCCST ,BYTES=00130
       00610    DEF FIELD NAME=(STGCSL  ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
       00620                TYPE=X,START=00001,BYTES=002
       00630    DEF FIELD NAME=(STQCHDY ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
       00640                TYPE=C,START=00003,BYTES=002
       00650    DEF FIELD NAME=(STQCHDT ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
       00660                TYPE=C,START=00003,BYTES=006
```

```
00670   DEF FIELD NAME=(STQCHDN ,SEQ  ),SEGM=STSCHIS ,PCB=STBICAP2, -
00680             TYPE=C,START=00003,BYTES=012
00690   DEF FIELD NAME=(STQCHDM ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00700             TYPE=C,START=00005,BYTES=002
00710   DEF FIELD NAME=(STQCHDD ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00720             TYPE=C,START=00007,BYTES=002
00730   DEF FIELD NAME=(STQCHNO ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00740             TYPE=C,START=00009,BYTES=006
00750   DEF FIELD NAME=(STFCHRF ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00760             TYPE=C,START=00015,BYTES=025
00770   DEF FIELD NAME=(STFCHIC ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00780             TYPE=C,START=00040,BYTES=002
00790   DEF FIELD NAME=(STFCHAM ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00800             TYPE=C,START=00042,BYTES=012
00810   DEF FIELD NAME=(STFCLOS ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
00820             TYPE=C,START=00054,BYTES=077


DLZ500I ROUTINE CREATED FOR PCB STBICAP2
```

---

Figure 85. ISQL EXTRACT DEFINEs Utility Listing

---

---

# 11.1.4.2 Displaying the ISQL Routine

An SQL/DS user can retrieve information about an ISQL routine and display it on a display terminal or print it on a work station printer. This is done through an SQL/DS query. For the sample routine created by the ISQL EXTRACT DEFINEs utility in the previous section, the following query can be used:

```
SELECT NAME,SEQNO,COMMAND FROM SQLDBA.ROUTINE WHERE NAME='STBICAP2'
```

This query selects the name, seqno, and command columns from the ISQL routine table for the routine called STBICAP2.

The printed output for this query result is shown in Figure 86. Note that each row of the ROUTINE table contains the routine name, a sequence number, a command, and a space for comments. The NAME column identifies the rows that belong to a particular routine (STBICAP2, in this example). SEQNO specifies the sequence that the commands in the routine are executed. The COMMAND column is where the SQL and ISQL commands are placed. The REMARKS column can be used to record notes about the routine or a particular command in the routine.

---

```
    11/24/88                        SELECT NAME,SEQNO,COMMAND FROM SQLDBA.ROUTINE WHERE
NAME='STBICAP2'                         PAGE    1


     NAME             SEQNO  COMMAND
     --------  ------------  ------------------------------------------------------------------------
----------
     STBICAP2             0  DEF PCB NAME=STBICAP2,PSB=(STBICAP,2  ),PROC=SAMPLE
     STBICAP2            10  DEF SEGMENT NAME=STSCCST ,PCB=STBICAP2,PARENT=0       ,BYTES=00106
     STBICAP2            20  DEF FIELD NAME=(STQCCNO ,SEQ  ),SEGM=STSCCST ,PCB=STBICAP2, -
     STBICAP2            30            TYPE=C,START=00001,BYTES=006
     STBICAP2            40  DEF FIELD NAME=(STFCCA1 ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
     STBICAP2            50            TYPE=C,START=00007,BYTES=025
     STBICAP2            60  DEF FIELD NAME=(STUCCNM ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
     STBICAP2            70            TYPE=C,START=00007,BYTES=025
     STBICAP2            80  DEF FIELD NAME=(STFCCA2 ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
     STBICAP2            90            TYPE=C,START=00057,BYTES=025
     STBICAP2           100  DEF FIELD NAME=(STFCCA3 ,NOSEQ),SEGM=STSCCST ,PCB=STBICAP2, -
     STBICAP2           110            TYPE=C,START=00082,BYTES=025
     STBICAP2           120  DEF SEGMENT NAME=STSCLOC ,PCB=STBICAP2,PARENT=STSCCST ,BYTES=00106
     STBICAP2           130  DEF FIELD NAME=(STQCLNO ,SEQ  ),SEGM=STSCLOC ,PCB=STBICAP2, -
     STBICAP2           140            TYPE=C,START=00001,BYTES=006
     STBICAP2           150  DEF FIELD NAME=(STFCLNM ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
     STBICAP2           160            TYPE=C,START=00007,BYTES=025
     STBICAP2           170  DEF FIELD NAME=(STFCLA1 ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
     STBICAP2           180            TYPE=C,START=00032,BYTES=025
     STBICAP2           190  DEF FIELD NAME=(STFCLA2 ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
     STBICAP2           200            TYPE=C,START=00057,BYTES=025
     STBICAP2           210  DEF FIELD NAME=(STFCLA3 ,NOSEQ),SEGM=STSCLOC ,PCB=STBICAP2, -
     STBICAP2           220            TYPE=C,START=00082,BYTES=025
     STBICAP2           230  DEF SEGMENT NAME=STPCORD ,PCB=STBICAP2,PARENT=STSCLOC ,BYTES=00055
     STBICAP2           240  DEF FIELD NAME=(STQCODY ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
```

```
     STBICAP2           250            TYPE=C,START=00001,BYTES=002
     STBICAP2           260  DEF FIELD NAME=(STQCODT ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           270            TYPE=C,START=00001,BYTES=006
     STBICAP2           280  DEF FIELD NAME=(STQCODN ,SEQ  ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           290            TYPE=C,START=00001,BYTES=012
     STBICAP2           300  DEF FIELD NAME=(STQCODM ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           310            TYPE=C,START=00003,BYTES=002
     STBICAP2           320  DEF FIELD NAME=(STQCODD ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           330            TYPE=C,START=00005,BYTES=002
     STBICAP2           340  DEF FIELD NAME=(STQCONO ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           350            TYPE=C,START=00007,BYTES=006
     STBICAP2           360  DEF FIELD NAME=(STFCORF ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           370            TYPE=C,START=00013,BYTES=025
     STBICAP2           380  DEF FIELD NAME=(STFCOIC ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           390            TYPE=C,START=00038,BYTES=006
     STBICAP2           400  DEF FIELD NAME=(STFCOAM ,NOSEQ),SEGM=STPCORD ,PCB=STBICAP2, -
     STBICAP2           410            TYPE=C,START=00044,BYTES=012
     STBICAP2           420  DEF SEGMENT NAME=STCCITM ,PCB=STBICAP2,PARENT=STPCORD ,BYTES=00038
     STBICAP2           430  DEF FIELD NAME=(STKCIIN ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
     STBICAP2           440            TYPE=C,START=00001,BYTES=006
     STBICAP2           450  DEF FIELD NAME=(STQCILI ,SEQ  ),SEGM=STCCITM ,PCB=STBICAP2, -
     STBICAP2           460            TYPE=C,START=00007,BYTES=002
     STBICAP2           470  DEF FIELD NAME=(STFCIQO ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
     STBICAP2           480            TYPE=C,START=00009,BYTES=006
     STBICAP2           490  DEF FIELD NAME=(STFCIQS ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
     STBICAP2           500            TYPE=C,START=00015,BYTES=006
     STBICAP2           510  DEF FIELD NAME=(STFCIQB ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
     STBICAP2           520            TYPE=C,START=00021,BYTES=006
     STBICAP2           530  DEF FIELD NAME=(STFCIAM ,NOSEQ),SEGM=STCCITM ,PCB=STBICAP2, -
     STBICAP2           540            TYPE=C,START=00027,BYTES=012
     STBICAP2           550  DEF SEGMENT NAME=STSCSTA ,PCB=STBICAP2,PARENT=STSCCST ,BYTES=00024
     STBICAP2           560  DEF FIELD NAME=(STFCSCL ,NOSEQ),SEGM=STSCSTA ,PCB=STBICAP2, -
     STBICAP2           570            TYPE=C,START=00001,BYTES=012


     11/24/88                    SELECT NAME,SEQNO,COMMAND FROM SQLDBA.ROUTINE WHERE
NAME='STBICAP2'                      PAGE    2



     NAME             SEQNO  COMMAND
     --------  ------------  ------------------------------------------------------------------------
----------
     STBICAP2           580  DEF FIELD NAME=(STFCSBL ,NOSEQ),SEGM=STSCSTA ,PCB=STBICAP2, -
     STBICAP2           590            TYPE=C,START=00013,BYTES=012
     STBICAP2           600  DEF SEGMENT NAME=STSCHIS ,PCB=STBICAP2,PARENT=STSCCST ,BYTES=00130
     STBICAP2           610  DEF FIELD NAME=(STGCSL  ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           620            TYPE=X,START=00001,BYTES=002
     STBICAP2           630  DEF FIELD NAME=(STQCHDY ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           640            TYPE=C,START=00003,BYTES=002
     STBICAP2           650  DEF FIELD NAME=(STQCHDT ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           660            TYPE=C,START=00003,BYTES=006
     STBICAP2           670  DEF FIELD NAME=(STQCHDN ,SEQ  ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           680            TYPE=C,START=00003,BYTES=012
     STBICAP2           690  DEF FIELD NAME=(STQCHDM ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           700            TYPE=C,START=00005,BYTES=002
     STBICAP2           710  DEF FIELD NAME=(STQCHDD ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           720            TYPE=C,START=00007,BYTES=002
     STBICAP2           730  DEF FIELD NAME=(STQCHNO ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           740            TYPE=C,START=00009,BYTES=006
     STBICAP2           750  DEF FIELD NAME=(STFCHRF ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           760            TYPE=C,START=00015,BYTES=025
     STBICAP2           770  DEF FIELD NAME=(STFCHIC ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           780            TYPE=C,START=00040,BYTES=002
     STBICAP2           790  DEF FIELD NAME=(STFCHAM ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           800            TYPE=C,START=00042,BYTES=012
     STBICAP2           810  DEF FIELD NAME=(STFCLOS ,NOSEQ),SEGM=STSCHIS ,PCB=STBICAP2, -
     STBICAP2           820            TYPE=C,START=00054,BYTES=077
```

---

Figure 86. Routine STBICAP2

---

## 11.1.4.3 Creating the Target Table

The target table is the SQL/DS table that is to contain the data extracted from the DL/I data base. This table must be created before the SQL/DS extract facility is run.

Figure 87 shows a sample job stream to create an SQL/DS table named STDCDBP1. In this example, the STDCDBP1 table is to contain the data extracted from the DL/I data base for the first access path (see Figure 83). An entry must be identified in the SQL/DS table for each field that is extracted from the DL/I data base. This entry contains a column name (field name), data type, and size for each data item.

For this example, the DL/I field names were used as the SQL/DS column names. Because SQL/DS allows its column names to be longer than the 8-character DL/I field names, a user may specify a more descriptive name, such as "CUSTOMER NAME" for a column.

Note the SQL/DS CREATE TABLE command contained in the job stream; this is the command which is used to create the target table.

You should use the listing from the ISQL EXTRACT DEFINEs utility as a reference for creating the target table. This will make it easier to ensure that all fields have a column defined for them and that the columns are in the correct order.

```
// JOB CREATE SAMPLE SQL/DS TABLE
* ------------------------------------------------------------
* THIS JOB CREATES THE SQL/DS TABLES IN ORDER TO EXTRACT
* DATA FROM THE DL/I SAMPLE DATA BASE
* ------------------------------------------------------------
// LIBDEF *,SEARCH='SQL prod lib'
// EXEC PROC='sqldbs'
// EXEC PGM=ARIDBS,SIZE=AUTO
   CONNECT SQLDBA IDENTIFIED BY 'sqldbapw'
   COMMENT 'THIS TABLE IS FOR THE FOLLOWING PATH IN DATA BASE STDCDBP:'
   COMMENT 'SEGMENT = STSCCST  CUSTOMER NAME AND ADDRESS'
   COMMENT '              |   '
   COMMENT 'SEGMENT = STSCLOC  CUSTOMER LOCATION'
   COMMENT '                |  '
   COMMENT 'SEGMENT = STSCORD  CUSTOMER ORDER'
   COMMENT '              |   '
   COMMENT 'SEGMENT = STSCITM  ITEM DESCRIPTION'
   CREATE  TABLE STDCDBP1
          (STQCCNO CHAR(06),
           STUCCNM CHAR(25),
           STFCCA1 CHAR(25),
           STFCCA2 CHAR(25),
           STFCCA3 CHAR(25),
           STQCLNO CHAR(06),
           STFCLNM CHAR(25),
           STFCLA1 CHAR(25),
           STFCLA2 CHAR(25),
           STFCLA3 CHAR(25),
           STQCODY CHAR(02),
```

```
            STQCODT CHAR(06),
            STQCODN CHAR(12),
            STQCODM CHAR(02),
            STQCODD CHAR(02),
            STQCONO CHAR(06),
            STFCORF CHAR(25),
            STFCOIC CHAR(06),
            STFCOAM CHAR(12),
            STKCIIN CHAR(06),
            STQCILI CHAR(02),
            STFCIQO CHAR(06),
            STFCIQS CHAR(06),
            STFCIQB CHAR(06),
            STFCIAM CHAR(12))
            IN PUBLIC.ISQL;
/*
/&
```

Figure 87. Sample Job Stream to Create SQL/DS Table for Extract

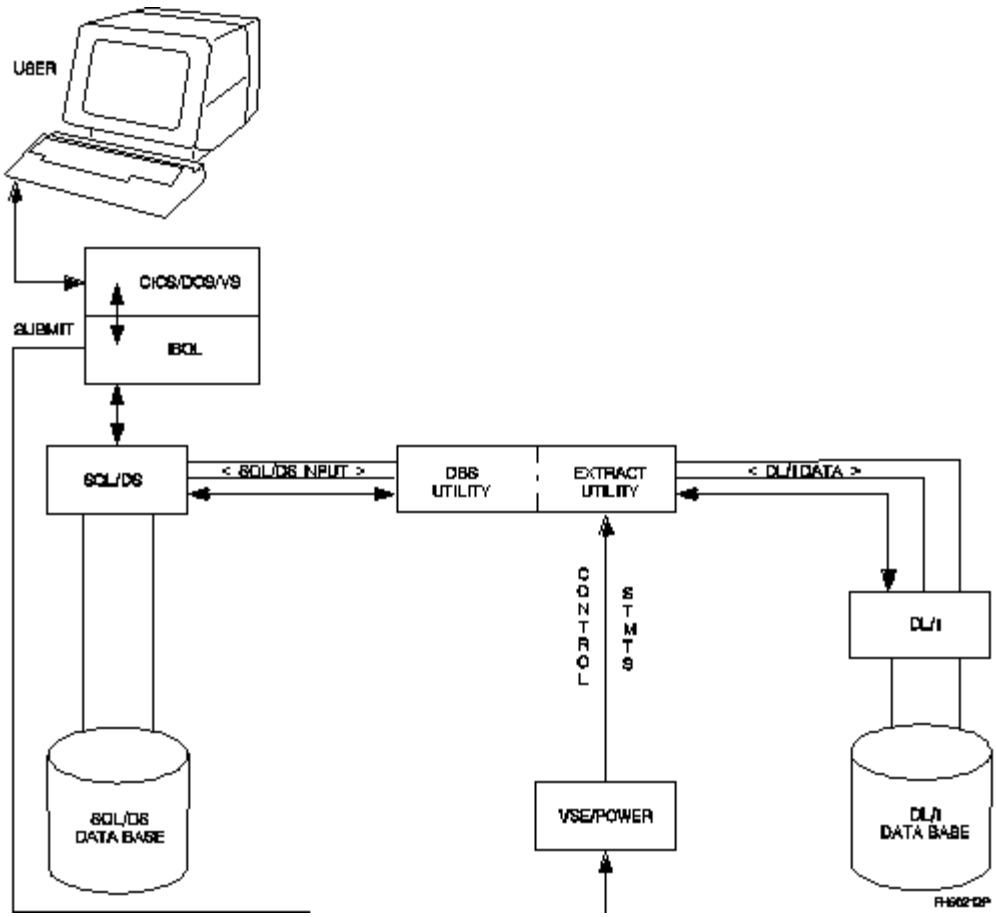



Figure 88. Extract Facility Overview

# 11.1.4.4 Extracting the DL/I Data

The following procedures are necessary to extract data from the DL/I data base:

- The Extract Defines routine must be invoked to define the structure and organization of the source DL/I data base.
- An ISQL EXTRACT command must be issued to request retrieval of data from the DL/I data base.
- The extract requests on the extract queue must be submitted for execution. During execution, the EXTRACT utility retrieves data from DL/I and passes it to SQL/DS. This data is inserted by the DBS utility into the target SQL/DS tables.

Subtopics:



---

## 11.1.4.4.1 Invoking the Extract Defines Routine

```
The ISQL routine that contains the Extract Defines commands is invoked via
the ISQL RUN command.  SQL/DS DBA authority is required to run this
routine for a userid other than SQLDBA.  For our example, the following
command is issued:  RUN SQLDBA.STBICAP2


When this command is invoked, the structure and organization of the source
DL/I data base is defined for the specified PCB and the logical
representation of the DL/I path is placed in an External Data Table (EDT).
The EDT will subsequently be used to map the extracted DL/I data to their
corresponding fields in the SQL/DS tables.
```

---

 *© Copyright IBM Corp. 1981, 1989*

---

## 11.1.4.4.2 Retrieving Data from the DL/I Data Base

An ISQL EXTRACT command must be issued to request retrieval of data from the DL/I data base. When the EXTRACT command is issued, the request is placed in the extract queue; the user is then free to perform other operations or to exit ISQL and return to see the results in a later session.

To use the EXTRACT command, a user must be the definer of, or have been granted extract authority on, the EDT and have SQL/DS insert authority on the target table.

The EXTRACT command specifies which data is to be selected from a named EDT and identifies a target table into which the data is to be placed.

For this example, the following ISQL EXTRACT command can be used:

```
EXTRACT INTO SQLDBA.STDCDBP1 SELECT * FROM STBICAP2
```

The above command requests retrieval of data from the DL/I data base for the first access path (STCCITM) for the PCB named STBICAP2 into the target table named STDCDBP1.

## 11.1.4.4.3 Loading Data into the SQL/DS Tables

The SQL/DS Extract Facility command, SUBMIT, is used to prepare the extract requests on the extract queue for execution. When the SUBMIT command is issued, the extract requests are combined and a job stream consisting of job control, DBS utility control cards, and extract utility control cards is generated and submitted to VSE/POWER for execution.

The SUBMIT command allows a user to generate a specific EXTRACT request; generate all extracts for a particular PCB; or generate all extract requests on the extract queue.

For this example, the following command is issued to generate all extracts for the PCB named STBICAP2; SUBMIT STBICAP2 When the jobs submitted to VSE/POWER have successfully completed processing, the DL/I data from the DL/I data bases is transferred into the designated SQL/DS target table.

# 11.1.4.5 Displaying the SQL/DS Table Data

After the DL/I data has been extracted and stored in the SQL/DS table, the user can display it by entering the following select command:

```
SELECT * FROM SQLDBA.STDCDBP1
```

Figure 89 shows what the data looks like in the SQL/DS table.

```
SELECT * FROM SQLDBA.STDCDBP1
STQCCNO  STUCCNM                   STFCCA1                    STFCCA2
-------  ------------------------  -------------------------  ----------------
000001   COMPANY X INC.            10 MAIN STREET             NEW YORK, NY 100
000001   COMPANY X INC.            10 MAIN STREET             NEW YORK, NY 100
000001   COMPANY X INC.            10 MAIN STREET             NEW YORK, NY 100
000002   COMPANY Y INC.            4 PENN AVENUE              CHICAGO, ILL 200
000003   COMPANY Z INC.            6 HYDE STREET              SAN FRANCISCO, C
000004   COMPANY N INC.            32 CAMP SITE               JACKSONHOLE, WY
000005   COMPANY L INC.            7 CARD DRAW                LAS VEGAS, NEVAD
000006   COMPANY K INCORPORATED    7000 LUMBER PILE           SEATTLE, WASHING
* END OF RESULT ************** 8 ROWS DISPLAYED ****** COST ESTIMATE IS 1    **

RIGHT 3

STFCCA2                   STFCCA3                    STQCLNO  STFCLNM
------------------------  -------------------------  -------  ----------------
NEW YORK, NY 10010         MR. JOHN SMITH            000010   EASTERN REGION
NEW YORK, NY 10010         MR. JOHN SMITH            000020   WESTERN REGION
NEW YORK, NY 10010         MR. JOHN SMITH            000020   WESTERN REGION
CHICAGO, ILL 20010         MR. JOHN SMITH            000010   NORTHEASTERN REG
SAN FRANCISCO, CA 62562    MR. JOHN SMITH            000010   SOUTHEASTERN REG
JACKSONHOLE, WY 86421      MR. JOHN SMITH            000010   SOUTHEASTERN REG
LAS VEGAS, NEVADA 74491    MS. JANE DOE              000010   EASTERN REGION
SEATTLE, WASHINGTON 98513  MR. JOHN DOE              000010   SOUTHEASTERN REG
* END OF RESULT ************** 8 ROWS DISPLAYED ****** COST ESTIMATE IS  1   **

RIGHT 3

STFCLNM                   STFCLA1                    STFCLA2
------------------------  -------------------------  ------------------------
EASTERN REGION            69 BROAD STREET            PHILADELPHIA, PA 11020
WESTERN REGION            5296 BATTLESHIP BLVD.      SAN DIEGO, CA 93210
WESTERN REGION            5296 BATTLESHIP BLVD.      SAN DIEGO, CA 93210
NORTHEASTERN REGION       1776 REVERE STREET         BOSTON, MA 01526
SOUTHEASTERN REGION       715 ROUNDTRIP PLACE        ATLANTA, GA 10724
SOUTHEASTERN REGION       99 SEASIDE DRIVE           MIAMI, FLA 11526
EASTERN REGION            59 POKER PLACE             ATLANTIC CITY, NJ 14066
SOUTHEASTERN REGION       222 PINETREE STREET        CHARLESTON, SC 24567
* END OF RESULT ************** 8 ROWS DISPLAYED ****** COST ESTIMATE IS  1   **

RIGHT 3
```

```
STFCLA3                    STQCODY  STQCODT  STQCODN       STQCODM  STQCODD  ST
-------------------------  -------  -------  ------------  -------  -------  --
MR. JOHN DOE               77       770129   770129100500  01       29       10
MR. JOHN SMITH             77       770510   770510102050  05       10       10
MR. JOHN SMITH             77       770510   770510102050  05       10       10
MR. JOHN DOE               77       770310   770310100600  03       10       10
MR. JOHN DOE               77       770920   770920100700  09       20       10
MS. JANE DOE               77       770510   770510100610  05       10       10
MR. JOHN SMITH             77       771020   771020100705  10       20       10
MR. JOHN SMITH             77       770320   770320100722  03       20       10
* END OF RESULT ************** 8 ROWS DISPLAYED ****** COST ESTIMATE IS  1   **


RIGHT 6


STQCONO  STFCORF                    STFCOIC  STFCOAM       STKCIIN  STQCILI  ST
-------  -------------------------  -------  ------------  -------  -------  --
100500   FIRST 1977 ORDER           000040   000000000400  000100   01       00
102050   SECOND 1977 ORDER          000035   000000000088  000200   01       00
102050   SECOND 1977 ORDER          000035   000000000088  000300   02       00
100600   SECOND 1977 ORDER          000090   000000000270  000200   01       00
100700   THIRD 1977 ORDER           000080   000000000160  000300   01       00
100610   INITIAL CUSTOMER CONTACT   000050   000000000750  000400   01       00
100705   THIRD 1977 ORDER           001000   000000001000  000500   01       00
100722   THIRD 1977 ORDER           001500   000000012000  000600   01       00
* END OF RESULT ************** 8 ROWS DISPLAYED ****** COST ESTIMATE IS  1  **


RIGHT 6


STFCIQO  STFCIQS  STFCIQB  STFCIAM
-------  -------  -------  ------------
000040   000040   000000   000000000400
000018   000008   000010   000000000054
000017   000017   000000   000000000034
000090   000090   000000   000000000270
000080   000040   000040   000000000160
000050   000050   000000   000000000750
001000   001000   000000   000000001000
001500   001500   000000   000000012000
* END OF RESULT ************** 8 ROWS DISPLAYED ****** COST ESTIMATE IS  1   **
```

---

Figure 89. Display of Extracted Data in Table SQLDBA.STDCDBP1

---

 *© Copyright IBM Corp. 1981, 1989*

---

# APPENDIX1 Part 12. Appendix

Subtopics:



 *© Copyright IBM Corp. 1981, 1989*

# APPENDIX1.1 Appendix A. Directory of Programming Interfaces for Customers

Identified here are those interfaces provided by DL/I DOS/VS by which a customer-written program is to request or receive functions or services of DL/I DOS/VS.

Subtopics:



IBM Library Server

# APPENDIX1.1.1 Interfaces

Subtopics:



# APPENDIX1.1.1.1 Application Programming Interfaces

This section lists where, in the DL/I DOS/VS publications, the programming interfaces for DL/I DOS/VS are described.

*DL/I DOS/VS Release Guide*, (SC33-6211)

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

*DL/I DOS/VS Low-Level-Code/Continuity Check in Data Language/I DOS/VS, Program Reference and Operations Manual, (SH20-9046)*

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

*DL/I DOS/VS Guide for New Users, (SH24-5001)*

*DL/I DOS/VS Reference Summary - Call and RQDLI Interfaces, (SX24-5103)*

*DL/I DOS/VS Reference Summary - High Level Programming Interface (SX24-5120)*

# APPENDIX1.1.1.2 System Programming Interfaces (Resource Definition)

*This section lists where, in the DL/I DOS/VS Publications, the interfaces for the DL/I resource definition are described.*

*DL/I DOS/VS Guide for New Users, (SH24-5001)*

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

*DL/I DOS/VS Diagnostic Guide, (SH24-5002)*

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

*DL/I DOS/VS Recovery and Restart Guide, (SH24-5030)*

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

*DL/I DOS/VS Reference Summary - System Programming, (SX24-5104)*

# APPENDIX1.1.1.3 Messages and Codes

*The following manual contains the return and error codes of the DL/I DOS/VS programming interfaces:*

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

# APPENDIX1.1.2 Macros

*This section describes the macros which are intended to be used as, or as part of the programming interface for DL/I DOS/VS:*

° *List of executable macros*

*The following executable macros are distributed as part of DL/I DOS/VS:*

- *ACCESS*
- *DATASET*
- *CALLDLI*
- *DBD*
- *DBDGEN*
- *DLZACT*
- *DLZNN*
- *DLZNNICT*
- *DLZSLC*
- *DLZTRACE*
- *FIELD*
- *FINISH*
- *LCHILD*
- *PCB*
- *PSBGEN*
- *SEGM*
- *SENFLD*
- *SENSEG*
- *VIRFLD*
- *XDFLD*

° *List of Mapping Macros*

- *DIB*
- *DLZUIB*
- *DLZUIBC*
- *DLZUIBP*
- *DLZUIBR*
- *DLZSTBF*

 *© Copyright IBM Corp. 1981, 1989*

# Index

## A



## B

# C

# D

# E

# F



# H

# I



# J



# K

# L



# M

# N



# O



# P

# R

# S

# T

# U



# V



# W



# X

# COMMENTS Readers' Comments -- We'd Like to Hear from You

*IBM Data Language/I*
*Disk Operating System/Virtual Storage*
*(DL/I DOS/VS)*
*Resource Definition and Utilities*

*Publication No. SH24-5021-02*

*We would appreciate any comments you may have about this publication, 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. Feel free to comment on technical accuracy, retrievability, clarity, or overall structure of the publication.*

*If you wish a reply, please give your name and address below. Customers in the USA should use the first address. Elsewhere, please use the address in Germany.*

*IBM Corporation*
*Attn: Dept ACV - BP*
*1001 WT Harris Blvd*
*Charlotte, NC 28257,*
*U.S.A.*

*or to:*

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

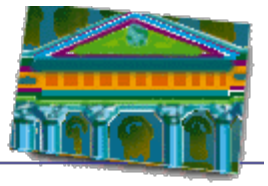
*You may also respond via* **FAX** *to: Department 3248, Country Code 49, (0)7031-16-3456*

*Name . . . . . .* _____
*Company or Organization* _____
*Address . . . . . .* _____
_____
_____
*Phone No. . . . . .* _____

---



---





# IBM Library Server Library

Welcome to IBM's Library Server--your electronic library of books on the World Wide Web. Using Library Server, you can easily manage and display electronic books grouped into catalog collections, shelves and bookcases. Library Server's high-performance, morphological searching capabilities allow you to search books and entire shelves for the information you need.

Find books in the catalog with titles, names, or doc numbers containing:

Find

Browse Bookcases

Browse Shelves

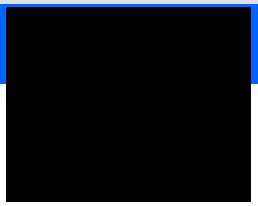
Administration

Help

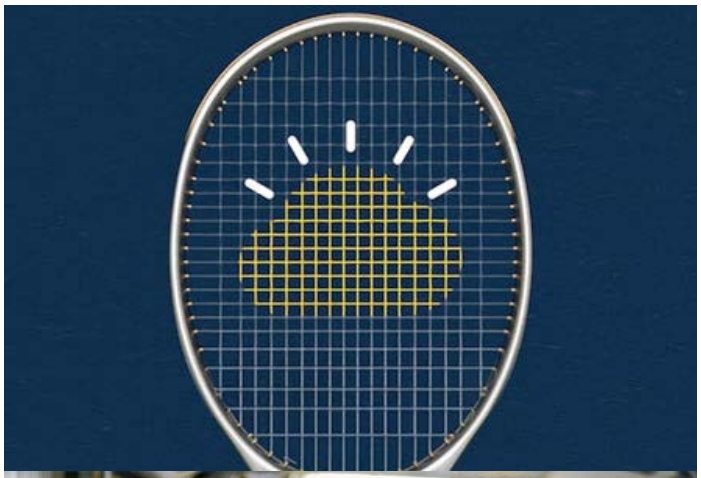
Getting Started | Product Information

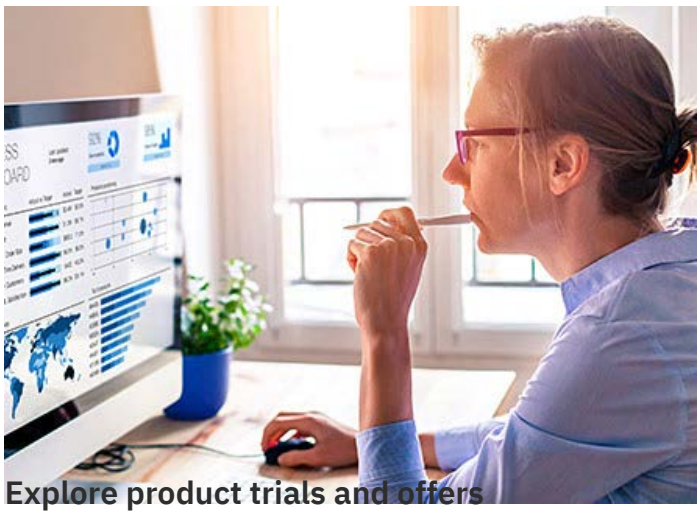
Clip the page

# This is artificial intelligence for real work



## This week at IBM





Explore product trials and offers

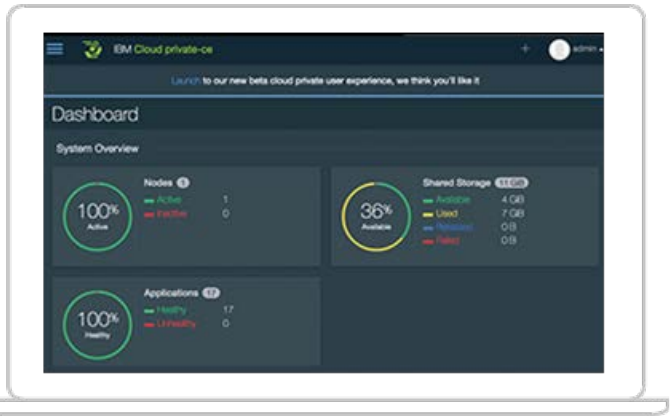
**Become a data scientist, no PhD required**

See current deals →

You bring basic computer skills and a passion for learning, we'll teach you the rest

See more products →

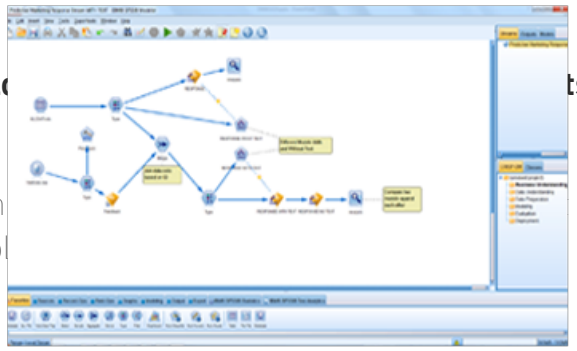
Earn your IBM Data Science Professional Certificate →




**Build ... tson AI**

With ... and resol...

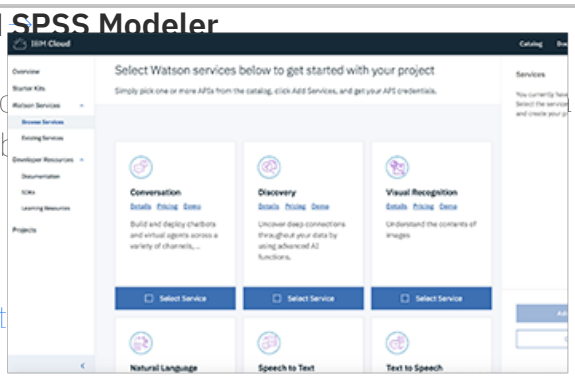
Try the Supply Chain Business Network demo



**IBM SPSS Modeler**

Get ... flexib...

Start ...




**Inside IBM**

Visit the IBM News Room →

Try a free edition now →


**Get your first look at the Think 2019 conference**

**Find the upside to information overload**

**By 2020, Earth will be home to 2.5 billion ...**

**Let's put smart to work™**

Technologies like AI, cloud and blockchain have to be more than smart

See how we're engaging new technologies to put smart to work →

**For developers** | For business

developerWorks →

Redbooks →

Support →

Marketplace →

**Trending in tech**

SPSS →

Careers →

Blockchain →

AI →

Open source →

Swift →

Cloud →

Serverless computing →

**Commit to the cause**


Your software can help save lives

Answer the Call for Code →

*Case study: OpenSponsorship*

## What athlete should endorse your product? Ask AI.

Sponsors expect more than good game stats from the sports figures who promote their products. In this social media era, reach is also key: stats about followers, retweets and engagement also make up the metrics that help sponsors choose a

spokesperson.

OpenSponsorship wanted brands of all sizes to find the right influencers. They chose IBM Watson services to analyze relevant social data and help place athletes like 'fastest woman alive' Carmelita Jeter in some unexpected but successful deals.

Read the whole story →

Explore Watson products and services →