

IMS



# Application Programming: EXEC DLI Commands for CICS and IMS

*Version 9*



IMS



# Application Programming: EXEC DLI Commands for CICS and IMS

*Version 9*

**Note**

Before using this information and the product it supports, read the information in "Notices" on page 107.

This edition applies to IMS Version 9 (program number 5655-J38) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC18-7811-02.

© **Copyright International Business Machines Corporation 1974, 2007.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

Figures . . . . .	v
-------------------	---

Tables . . . . .	vii
------------------	-----

## About This Book. . . . . ix

Summary of Contents . . . . .	ix
Prerequisite Knowledge . . . . .	ix
IBM Product Names Used in This Information . . . . .	ix
How to Read Syntax Diagrams . . . . .	xi
Accessibility Features for IMS Version 9 . . . . .	xii
Accessibility Features . . . . .	xii
Keyboard Navigation . . . . .	xiii
Related Accessibility Information . . . . .	xiii
IBM and Accessibility . . . . .	xiii
How to Send Your Comments . . . . .	xiii

## Summary of Changes . . . . . xv

Changes to the Current Edition of This Book for IMS Version 9 . . . . .	xv
Changes to This Book for IMS Version 9 . . . . .	xv
Library Changes for IMS Version 9 . . . . .	xv
New and Revised Titles . . . . .	xv
Organizational Changes . . . . .	xv
Terminology Changes . . . . .	xvi

## Chapter 1. How EXEC DLI Application Programs Work with IMS . . . . . 1

Getting Started with EXEC DLI . . . . .	1
A Database Hierarchy Example . . . . .	2

## Chapter 2. Defining Application Program Elements . . . . . 5

Specifying an Application Interface Block (AIB) . . . . .	5
AIB Mask . . . . .	5
CICS Restrictions with AIB Support . . . . .	5
Specifying the DL/I Interface Block (DIB) . . . . .	5
Defining a Key Feedback Area . . . . .	9
Defining I/O Areas . . . . .	9
COBOL I/O Area . . . . .	9
PL/I I/O Area . . . . .	10
Assembler Language I/O Area . . . . .	10

## Chapter 3. Writing an Application Program . . . . . 11

Programming Guidelines . . . . .	11
Coding a Program in Assembler Language . . . . .	12
Coding a Program in COBOL . . . . .	16
Coding a Program in PL/I . . . . .	19
Coding a Program in C . . . . .	23
Preparing Your EXEC DLI Program for Execution . . . . .	29
Translator Options Required for EXEC DLI . . . . .	29
Compiler Options Required for EXEC DLI . . . . .	29
Linkage Editor Options Required for EXEC DLI . . . . .	29

## Chapter 4. EXEC DLI Commands for an Application Program . . . . . 31

PCBs and PSB . . . . .	31
I/O PCB . . . . .	31
Alternate PCB . . . . .	31
DB PCB . . . . .	31
GSAM PCB . . . . .	31
PCB Summary . . . . .	32
Format of a PSB . . . . .	33
EXEC DLI Commands . . . . .	33
Summary of EXEC DLI Commands . . . . .	34
DLET Command . . . . .	35
GN Command . . . . .	36
GNP Command . . . . .	41
GU Command . . . . .	47
ISRT Command . . . . .	52
POS Command . . . . .	58
REPL Command . . . . .	59
RETRIEVE Command . . . . .	63
SCHD Command . . . . .	64
TERM Command . . . . .	65
System Service Commands . . . . .	66
ACCEPT Command . . . . .	66
CHKP Command . . . . .	67
DEQ Command . . . . .	68
LOAD Command . . . . .	69
LOG Command . . . . .	70
QUERY Command . . . . .	70
REFRESH Command . . . . .	71
ROLB Command . . . . .	72
ROLL Command . . . . .	73
ROLS Command . . . . .	73
SETS Command . . . . .	75
SETU Command . . . . .	76
STAT Command . . . . .	77
SYMCHKP Command . . . . .	78
XRST Command . . . . .	79

## Chapter 5. Recovering Databases and Maintaining Database Integrity . . . . . 83

Issuing Checkpoints in a Batch or BMP Program . . . . .	83
Issuing the CHKP Command . . . . .	84
Issuing the SYMCHKP Command . . . . .	84
Restarting Your Program and Checking for Position . . . . .	84
Backing Out Database Updates Dynamically: The ROLL and ROLB Commands . . . . .	84
Using Intermediate Backout Points: The SETS and ROLS Commands . . . . .	85

## Chapter 6. Processing Fast Path Databases . . . . . 87

Processing DEDBs with Subset Pointers . . . . .	87
Preparing to Use Subset Pointers . . . . .	89
Designating Subset Pointers . . . . .	90

Subset Pointer Options . . . . .	90
Subset Pointer Status Codes . . . . .	97
The POS Command . . . . .	97
Locating a Specific Sequential Dependent	
Segment . . . . .	98
Locating the Last Inserted Sequential Dependent	
Segment . . . . .	98
Identifying Free Space with the POS Command	99
The P Processing Option . . . . .	99

**Chapter 7. Comparing  
Command-Level and Call-Level  
Programs . . . . . 101**

DL/I Calls for IMS and CICS . . . . .	101
Comparing EXEC DLI Commands and DL/I Calls	102
Comparing Command Codes and Options . . . . .	103

**Chapter 8. Data Availability  
Enhancements . . . . . 105**

Accepting Database Availability Status Codes . . . . .	105
Obtaining Information about Database Availability	105

**Notices . . . . . 107**

Programming Interface Information . . . . .	109
Trademarks . . . . .	109

**Bibliography . . . . . 111**

IMS Version 9 Library . . . . .	111
Supplementary Publications . . . . .	111
Publication Collections . . . . .	111
Accessibility Titles Cited in This Library . . . . .	112

**Index . . . . . 113**

---

## Figures

1. The Structure of a Command-Level Batch or BMP Program . . . . . 1
2. Medical Hierarchy . . . . . 2
3. Processing a Long Chain of Segment Occurrences with Subset Pointers . . . . . 88
4. Examples of Setting Multiple Subset Pointers 88
5. More Examples of Setting Subset Pointers 89
6. How Subset Pointers Divide a Chain into Subsets . . . . . 89
7. Processing Performed for the Sample Passbook Example when the Passbook is Unavailable. . . 91
8. Processing Performed for the Sample Passbook Example when the Passbook is Available. . . 91
9. Retrieving the First Segment in a Chain of Segments . . . . . 92
10. Moving the Subset Pointer to the Next Segment after Your Current Position . . . . 94
11. Unconditionally Setting the Subset Pointer to Your Current Position . . . . . 95
12. Conditionally Setting the Subset Pointer to Your Current Position . . . . . 96





---

## Tables

1. Licensed Program Full Names and Short Names . . . . .	ix	8. Summary of PCB Information . . . . .	32
2. PATIENT Segment . . . . .	3	9. Summary of EXEC DLI Commands . . . . .	34
3. ILLNESS Segment. . . . .	3	10. DL/I Calls Available to IMS and CICS Command-Level Application Programs . . . . .	101
4. TREATMNT Segment . . . . .	4	11. Comparing Call-Level and Command-Level Programs: Commands and Calls . . . . .	102
5. BILLING Segment. . . . .	4	12. Comparing Call-Level and Command-Level Programs: Command Codes and Options . . . . .	103
6. PAYMENT Segment . . . . .	4		
7. HOUSEHOLD Segment . . . . .	4		



---

## About This Book

This information is available as part of the Information Management Software for z/OS® Solutions Information Center at <http://publib.boulder.ibm.com/infocenter/imzic>. A PDF version of this information is available in the information center.

This information is for CICS® application programmers whose programs use EXEC DLI commands in an IMS™ environment. This book lists and describes the EXEC DLI commands, and explains the procedures for writing application programs. For information on using databases (such as, position in the database, using multiple positioning, and using secondary indexing and logical relationships), see *IMS Version 9: Application Programming: Database Manager*.

---

## Summary of Contents

This book explains the basics of writing the DL/I part of your application program with EXEC DLI commands. It also contains reference information about the parts of an IMS command-level application program such as EXEC DLI commands, system service calls, qualification statements, EXEC DLI options, the DIB (DL/I Interface Block), I/O areas, and status codes. These chapters are for experienced programmers who understand IMS application programming and need only to look up a fact such as the meaning of a particular status code.

---

## Prerequisite Knowledge

IBM® offers a wide variety of classroom and self-study courses to help you learn IMS. For a complete list, see the IMS home page on the World Wide Web at: [www.ibm.com/ims](http://www.ibm.com/ims)

This book assumes you are a CICS programmer familiar with the functions, facilities, hardware, and software in *CICS Family: General Information* and from the Library page of the IMS home page on the Web: [www.ibm.com/ims](http://www.ibm.com/ims).

This book also assumes that, if you plan to write a CICS program, you are familiar with the principles covered in *CICS Transaction Server for z/OS CICS Application Programming* and in other CICS documentation.

---

## IBM Product Names Used in This Information

In this information, the licensed programs shown in Table 1 are referred to by their short names.

Table 1. Licensed Program Full Names and Short Names

Licensed program full name	Licensed program short name
IBM Application Recovery Tool for IMS and DB2®	Application Recovery Tool
IBM CICS Transaction Server for OS/390®	CICS
IBM CICS Transaction Server for z/OS	CICS
IBM DB2 Universal Database™	DB2 Universal Database
IBM DB2 Universal Database for z/OS	DB2 UDB for z/OS

*Table 1. Licensed Program Full Names and Short Names (continued)*

<b>Licensed program full name</b>	<b>Licensed program short name</b>
IBM Enterprise COBOL for z/OS	Enterprise COBOL
IBM Enterprise COBOL for z/OS and OS/390	Enterprise COBOL
IBM Enterprise PL/I for z/OS and OS/390	Enterprise PL/I
IBM High Level Assembler for MVS & VM & VSE	High Level Assembler
IBM IMS Advanced ACB Generator	IMS Advanced ACB Generator
IBM IMS Batch Backout Manager	IMS Batch Backout Manager
IBM IMS Batch Terminal Simulator	IMS Batch Terminal Simulator
IBM IMS Buffer Pool Analyzer	IMS Buffer Pool Analyzer
IBM IMS Command Control Facility for z/OS	IMS Command Control Facility
IBM IMS Connect for z/OS	IMS Connect
IBM IMS Connector for Java™	IMS Connector for Java
IBM IMS Database Control Suite	IMS Database Control Suite
IBM IMS Database Recovery Facility for z/OS	IMS Database Recovery Facility
IBM IMS Database Repair Facility	IMS Database Repair Facility
IBM IMS DataPropagator for z/OS	IMS DataPropagator
IBM IMS DEDB Fast Recovery	IMS DEDB Fast Recovery
IBM IMS Extended Terminal Option Support	IMS ETO Support
IBM IMS Fast Path Basic Tools	IMS Fast Path Basic Tools
IBM IMS Fast Path Online Tools	IMS Fast Path Online Tools
IBM IMS Hardware Data Compression-Extended	IMS Hardware Data Compression-Extended
IBM IMS High Availability Large Database (HALDB) Conversion Aid for z/OS	IBM IMS HALDB Conversion Aid
IBM IMS High Performance Change Accumulation Utility for z/OS	IMS High Performance Change Accumulation Utility
IBM IMS High Performance Load for z/OS	IMS HP Load
IBM IMS High Performance Pointer Checker for OS/390	IMS HP Pointer Checker
IBM IMS High Performance Prefix Resolution for z/OS	IMS HP Prefix Resolution
IBM Tivoli® NetView® for z/OS	Tivoli NetView for z/OS
IBM WebSphere® Application Server for z/OS and OS/390	WebSphere Application Server for z/OS
IBM WebSphere MQ for z/OS	WebSphere MQ
IBM WebSphere Studio Application Developer Integration Edition	WebSphere Studio
IBM z/OS	z/OS

Additionally, this information might contain references to the following IBM product names:

- "IBM C/C++ for MVS" or "IBM C/C++ for MVS/ESA" is referred to as either "C/MVS" or "C++/MVS."
- "IBM CICS for MVS" is referred to as "CICS."
- "IBM DataAtlas for OS/2" is referred to as "DataAtlas."
- "IBM Language Environment for MVS & VM" or "IBM z/OS Language Environment" is referred to as "Language Environment."
- "IBM PL/I for MVS & VM" or "IBM PL/I for OS/390 & VM" is referred to as "PL/I."

---

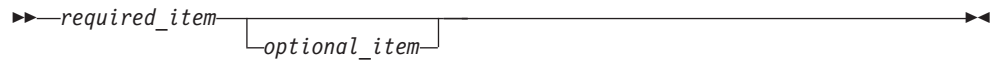
## How to Read Syntax Diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).



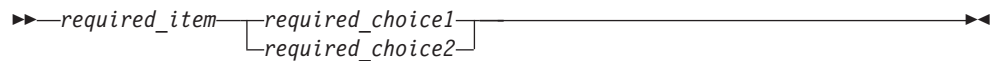
- Optional items appear below the main path.



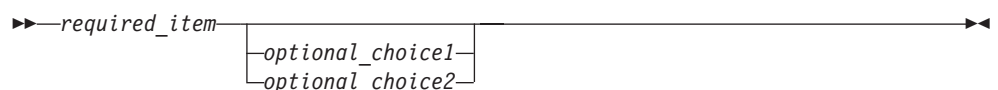
If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.



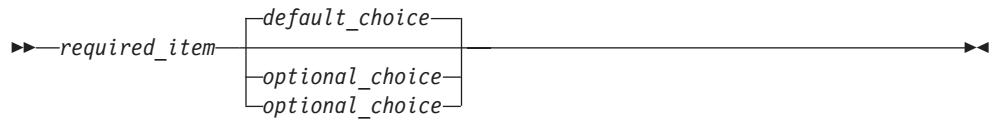
- If you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.



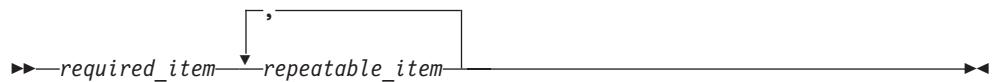
If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



If the repeat arrow contains a comma, you must separate repeated items with a comma.

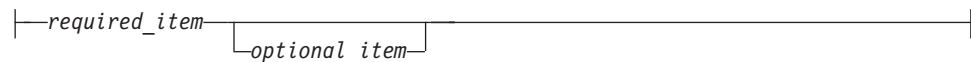


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



**fragment-name:**



- In IMS, a b symbol indicates one blank position.
- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown. Variables appear in all lowercase italic letters (for example, *column-name*). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

---

## Accessibility Features for IMS Version 9

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

### Accessibility Features

The following list includes the major accessibility features in z/OS products, including IMS Version 9. These features support:

- Keyboard-only operation.

- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size.

**Note:** The Information Management Software for z/OS Solutions Information Center (which includes information for IMS Version 9) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features by using the keyboard instead of the mouse.

## Keyboard Navigation

You can access IMS Version 9 ISPF panel functions by using a keyboard or keyboard shortcut keys.

For information about navigating the IMS Version 9 ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

## Related Accessibility Information

Online documentation for IMS Version 9 is available in the Information Management Software for z/OS Solutions Information Center.

## IBM and Accessibility

See the *IBM Accessibility Center* at [www.ibm.com/able](http://www.ibm.com/able) for more information about the commitment that IBM has to accessibility.

---

## How to Send Your Comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this or any other IMS information, you can take one of the following actions:

- Click the Feedback link located at the bottom of every page in the Information Management Software for z/OS Solutions Information Center. The information center can be found at <http://publib.boulder.ibm.com/infocenter/imzic>.
- Go to the IMS Library page at [www.ibm.com/software/data/ims/library.html](http://www.ibm.com/software/data/ims/library.html) and click the Library Feedback link, where you can enter and submit comments.
- Send your comments by e-mail to [imspubs@us.ibm.com](mailto:imspubs@us.ibm.com). Be sure to include the title, the part number of the title, the version of IMS, and, if applicable, the specific location of the text on which you are commenting (for example, a page number in the PDF or a heading in the Information Center).





---

## Summary of Changes

---

### Changes to the Current Edition of This Book for IMS Version 9

This edition includes technical and editorial changes.

---

### Changes to This Book for IMS Version 9

This book contains IMS Version 9 editorial changes.

---

### Library Changes for IMS Version 9

Changes to the IMS Library for IMS Version 9 include the addition of one title, a change of one title, organizational changes, and a major terminology change. Changes are indicated by a vertical bar (|) to the left of the changed text.

The IMS Version 9 information is now available in the Information Management Software for z/OS Solutions Information Center, which is available at <http://publib.boulder.ibm.com/infocenter/imzic>. The Information Management Software for z/OS Solutions Information Center provides a graphical user interface for centralized access to the product information for IMS, IMS Tools, DB2 Universal Database (UDB) for z/OS, DB2 Tools, and DB2 Query Management Facility (QMF™).

### New and Revised Titles

The following list details the major changes to the IMS Version 9 library:

- *IMS Version 9: IMS Connect Guide and Reference*

The library includes new information: *IMS Version 9: IMS Connect Guide and Reference*. This information is available in softcopy format only, as part of the Information Management Software for z/OS Solutions Information Center, and in PDF and BookManager® formats.

IMS Version 9 provides an integrated IMS Connect function, which offers a functional replacement for the IMS Connect tool (program number 5655-K52). In this information, the term *IMS Connect* refers to the integrated IMS Connect function that is part of IMS Version 9, unless otherwise indicated.

- The information formerly titled *IMS Version 8: IMS Java User's Guide* is now titled *IMS Version 9: IMS Java Guide and Reference*. This information is available in softcopy format only, as part of the Information Management Software for z/OS Solutions Information Center, and in PDF and BookManager formats.
- To complement the IMS Version 9 library, a retail book, *An Introduction to IMS* by Dean H. Meltz, Rick Long, Mark Harrington, Robert Hain, and Geoff Nicholls (ISBN # 0-13-185671-5), is available from IBM Press. Go to the IMS Web site at [www.ibm.com/ims](http://www.ibm.com/ims) for details.

### Organizational Changes

Organization changes to the IMS Version 9 library include changes to:

- *IMS Version 9: Customization Guide*
- *IMS Version 9: IMS Connect Guide and Reference*
- *IMS Version 9: IMS Java Guide and Reference*
- *IMS Version 9: Messages and Codes, Volume 1*

- *IMS Version 9: Utilities Reference: System*

A new appendix has been added to the *IMS Version 9: Customization Guide* that describes the contents of the ADFSSMPL (also known as SDFSSMPL) data set.

The IMS Connect messages that were in *IMS Version 9: IMS Connect Guide and Reference* have moved to *IMS Version 9: Messages and Codes, Volume 1*.

The IMS Connect commands that were in *IMS Version 9: IMS Connect Guide and Reference* have moved to *IMS Version 9: Command Reference*.

The chapter titled "DLIModel Utility" has moved from *IMS Version 9: IMS Java Guide and Reference* to *IMS Version 9: Utilities Reference: System*.

The DLIModel utility messages that were in *IMS Version 9: IMS Java Guide and Reference* have moved to *IMS Version 9: Messages and Codes, Volume 1*.

To ease the transition of your security support from the Security Maintenance Utility (SMU) to RACF, new SMU to RACF conversion utilities have been introduced. These utilities are documented in a new part in the *IMS Version 9: Utilities Reference: System*.

## Terminology Changes

IMS Version 9 introduces new terminology for IMS commands:

### **type-1 command**

A command, generally preceded by a leading slash character, that can be entered from any valid IMS command source. In IMS Version 8, these commands were called *classic* commands.

### **type-2 command**

A command that is entered only through the OM API. Type-2 commands are more flexible than type-1 commands and can have a broader scope. In IMS Version 8, these commands were called *IMSplex* commands or *enhanced* commands.

---

# Chapter 1. How EXEC DLI Application Programs Work with IMS

This chapter describes the components of your CICS program. It also describes the sample hierarchy used in the examples.

Your EXEC DLI application uses EXEC DLI commands to read and update DL/I databases. These applications can execute as pure batch, as a BMP program running with DBCTL or DB/DC, or as an online CICS program using DBCTL. Your EXEC DLI program can also issue system service commands when using DBCTL.

IMS DB/DC can provide the same services as DBCTL.

The following topics provide additional information:

- “Getting Started with EXEC DLI”
- “A Database Hierarchy Example” on page 2

---

## Getting Started with EXEC DLI

Figure 1 shows the main elements of programs that use EXEC DLI commands to access DL/I databases. The main differences between a CICS program and a command-level batch or BMP program (represented by Figure 1) are that you do not schedule a PSB for a batch program, and that you do not issue checkpoints for a CICS program. The numbers to the left of the figure correspond to the notes that follow Figure 1.

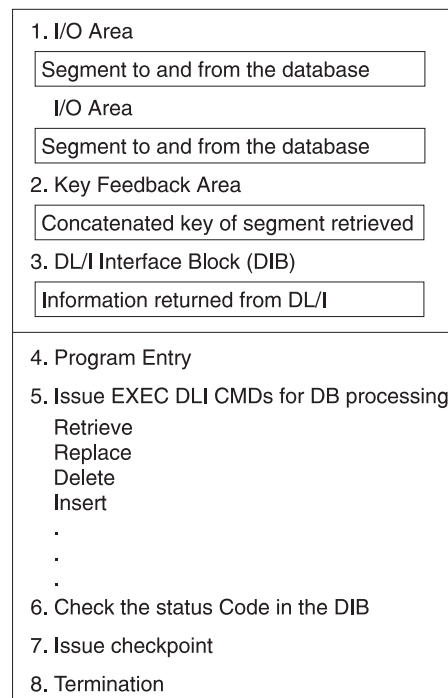


Figure 1. The Structure of a Command-Level Batch or BMP Program

## Getting Started with EXEC DLI

### Notes to Figure 1 on page 1:

- 1** I/O areas. DL/I passes segments to and from the program in the I/O areas. You may use a separate I/O area for each segment.
  - 2** Key feedback area. DL/I passes, on request, the concatenated key of the lowest-level segment retrieved to the key feedback area.
  - 3** DL/I Interface Block (DIB). DL/I and CICS place the results of each command in the DIB. The DIB contains most of the same information returned in the DB PCB for programs using the call-level interface.
- Note:** The horizontal line between **3** and **4** represents the end of the declarations section and the start of the executable code section of the program.
- 4** Program entry. Control is passed to your program during program entry.
  - 5** Issue EXEC DLI commands. Commands read and update information in the database.
  - 6** Check the status code. To find out the results of each command you issue, you should check the status code in the DIB after issuing an EXEC DLI command for database processing and after issuing a checkpoint command.
  - 7** Issue checkpoint. Issue checkpoints as needed to establish places from which to restart. Issuing a checkpoint commits database changes and releases resources.
  - 8** Terminate. This returns control to the operating system, commits database changes, and releases resources.

**Requirement:** CICS Transaction Server for z/OS runs with this version of IMS. Unless a distinction needs to be made, all supported versions are referred to as CICS. For a complete list of supported software, see the *IMS Version 9: Release Planning Guide*.

---

## A Database Hierarchy Example

Many of the examples use the medical hierarchy shown in Figure 2. The database contains information that a medical clinic might keep about its patients. To understand the examples, you should be familiar with the hierarchy and the segments it contains.

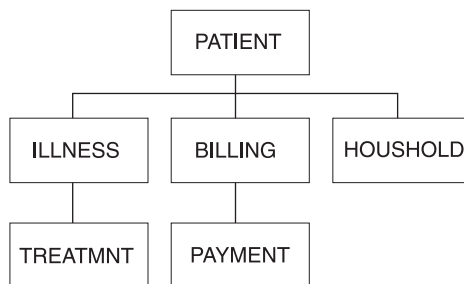


Figure 2. Medical Hierarchy

The tables that follow show the layouts of each segment in the hierarchy. The segment's field names are in the first row of each table. The number below each field name is the length in bytes that has been defined for that field.

- **PATIENT Segment**

Table 2 on page 3 shows the PATIENT segment.

It has three fields:

- The patient’s number (PATNO)
- The patient’s name (NAME)
- The patient’s address (ADDR)

PATIENT has a unique key field: PATNO. PATIENT segments are stored in ascending order based on the patient number. The lowest patient number in the database is 00001 and the highest is 10500.

*Table 2. PATIENT Segment*

Field Name	Field Length
PATNO	5
NAME	10
ADDR	30

- **ILLNESS Segment**

Table 3 shows the ILLNESS segment.

It has two fields:

- The date when the patient came to the clinic with the illness (ILLDATE)
- The name of the illness (ILLNAME)

The key field is ILLDATE. Because it is possible for a patient to come to the clinic with more than one illness on the same date, this key field is non unique, that is, there may be more than one ILLNESS segment with the same (an equal) key field value.

Usually during installation, the database administrator (DBA) decides the order in which to place the database segments with equal or no keys. The DBA can use the RULES keyword of the SEGM statement of the DBD to specify the order of the segments.

For segments with equal keys or no keys, RULES determines where the segment is inserted. Where RULES=LAST, ILLNESS segments that have equal keys are stored on a first-in-first-out basis among those with equal keys. ILLNESS segments with unique keys are stored in ascending order on the date field, regardless of RULES. ILLDATE is specified in the format YYYYMMDD.

*Table 3. ILLNESS Segment*

Field Name	Field Length
ILLDATE	8
ILLNAME	10

- **TREATMNT Segment**

Table 4 on page 4 shows the TREATMNT segment.

It contains four fields:

- The date of the treatment (DATE)
- The medicine that was given to the patient (MEDICINE)
- The quantity of the medicine that the patient received (QUANTITY)
- The name of the doctor who prescribed the treatment (DOCTOR)

The TREATMNT segment’s key field is DATE. Because a patient may receive more than one treatment on the same date, DATE is a non unique key field. TREATMNT, like ILLNESS, has been specified as having RULES=LAST. TREATMNT segments are also stored on a first-in-first-out basis. DATE is specified in the same format as ILLDATE—YYYYMMDD.

## A Database Hierarchy Example

Table 4. TREATMNT Segment

Field Name	Field Length
DATE	8
MEDICINE	10
QUANTITY	4
DOCTOR	10

- **BILLING Segment**

Table 5 shows the BILLING segment. It has only one field: the amount of the current bill. BILLING has no key field.

Table 5. BILLING Segment

Field Name	Field Length
BILLING	6

- **PAYMENT Segment**

Table 6 shows the PAYMENT segment. It has only one field: the amount of payments for the month. The PAYMENT segment has no key field.

Table 6. PAYMENT Segment

Field Name	Field Length
PAYMENT	6

- **HOUSHOLD Segment**

Table 7 shows the HOUSHOLD segment.

It contains two fields:

- The names of the members of the patient's household (RELNAME)
- How each member of the household is related to the patient (RELATN)

The HOUSEHOLD segment's key field is RELNAME.

Table 7. HOUSEHOLD Segment

Field Name	Field Length
RELNAME	10
RELATN	8

---

## Chapter 2. Defining Application Program Elements

This chapter describes how to use EXEC DLI commands with the application interface block (AIB) and DL/I interface block (DIB). Defining feedback and I/O areas is also discussed.

The following topics provide additional information:

- “Specifying an Application Interface Block (AIB)”
- “Specifying the DL/I Interface Block (DIB)”
- “Defining a Key Feedback Area” on page 9
- “Defining I/O Areas” on page 9

---

### Specifying an Application Interface Block (AIB)

EXEC DLI commands can use the AIB interface. For example, using the AIB interface, the format for the GU command would be EXEC DLI GU AIB(aib), instead of EXEC DLI GU USING PCB(n) using the PCB format.

With CICS Transaction Server 1.1 or later, the EXEC DLI commands in Chapter 4, “EXEC DLI Commands for an Application Program,” on page 31 are supported in the AIB format (as well as the PCB format).

With CICS Transaction Server 1.1 or later, and IMS/ESA® Version 5, the AIB-only commands, ICMD, RCMD, and GMSG are supported by using the EXEC DLI interface.

The CICS EDF (execution diagnostic facility) debugging transaction supports AIB EXEC DLI requests, just as it handles PCB type requests.

#### AIB Mask

The AIB mask must be supplied by the application and referenced in the EXEC call instead of the PCB number (for example, EXEC DLI GU AIB(aib)).

The DIBSTAT field is set with a valid STATUS code when AIBRETRN = X'00000000' or X'00000900'. Applications should test AIBRETRN for any other values and respond accordingly.

#### CICS Restrictions with AIB Support

Restrictions due to function shipping include:

- The AIBLEN field must be between 128 and 256 bytes. 128 is recommended.
- LIST=NO must **not** be specified on any PCBs in the PSB.

---

### Specifying the DL/I Interface Block (DIB)

Each time your program executes a DL/I command, DL/I returns a status code and other information to your program through the DL/I interface block (DIB), which is a subset of IMS PCB. Your program should check the status code to make sure the command executed successfully.

## Specifying the DL/I Interface Block (DIB)

Each program's working storage contains its own DIB. The contents of the DIB reflect the status of the last DL/I command executed in that program. If the information in your program's DIB is required by another program used by your transaction, you must pass the information to that program.

To access fields in the DIB, use labels that are automatically generated in your program by the translator.

**Restriction:** These labels are reserved; you must not redefine them.

In your COBOL, PL/I, assembler language, and C programs, some variable names are mandatory.

For a COBOL program:

```
DIBVER    PICTURE X(2)
DIBSTAT   PICTURE X(2)
DIBSEGM   PICTURE X(8)
DIBSEGLV  PICTURE X(2)
DIBKFBL   PICTURE S9(4) COMPUTATIONAL
DIBDBDNM  PICTURE X(8)
DIBDBORG  PICTURE X(8)

DIBVER    CHAR(2)
DIBSTAT   CHAR(2)
DIBSEGM   CHAR(8)
DIBSEGLV  CHAR(2)
DIBKFBL   FIXED BINARY (15,0)
DIBDBDNM  CHAR(8)
DIBDBORG  CHAR(8)
```

For an assembler language program:

```
DIBVER    CL2
DIBSTAT   CL2
DIBSEGM   CL8
DIBSEGLV  CL2
DIBKFBL   H
DIBDBDNM  CL8
DIBDBORG  CL8
```

For a C program:

```
unsigned char    dibver    {2} ;
unsigned char    dibstat   {2} ;
unsigned char    dibsegm   {8} ;
unsigned char    dibfic01  ;
unsigned char    dibfic02  ;
unsigned char    dibseglv  {2} ;
signed short int dibkfb1   ;
unsigned char    dibdbdnm  {8} ;
unsigned char    dibdborg  {8} ;
unsigned char    dibfic03  {6} ;
```

The following notes explain the contents of each variable name. The name in parenthesis is the label used to access the contents.

### 1. Translator Version (DIBVER)

This is the version of the DIB format your program is using. (DIBVER is used for documentation and problem determination.)

### 2. Status Codes (DIBSTAT)

DL/I places a 2-character status code in this field after executing each DL/I command. This code describes the results of the command.



## Specifying the DL/I Interface Block (DIB)

After processing a DL/I command, DL/I returns control to your program at the next sequential instruction following the command. The first thing your program should do after each command is to test the status code field and take appropriate action. If the command was completely successful, this field contains blanks.

The status codes that can be returned to this field (they are the only status codes returned to your program) are:

- bb** (Blanks) The command was completely successful.
- BA** For GU, GN, GNP, DLET, REPL, and ISRT commands. Data was unavailable.
- BC** For DLET, REPL, and ISRT commands. A deadlock was detected.
- FH** For GU, GN, GNP, DLET, REPL, ISRT, POS, CHKP, and SYMCHKP commands. The DEDB was inaccessible.
- FW** For GU, GN, GNP, DLET, REPL, ISRT, and POS commands. More buffer space is required than normally allowed.
- GA** For unqualified GN and GNP commands. DL/I returned a segment, but the segment is at a higher level in the hierarchy than the last segment that was returned.
- GB** For GN commands. DL/I reached the end of the database trying to satisfy your GN command and did not return a segment to your program's I/O area.
- GD** For ISRT commands. The program issued an ISRT command that did not have SEGMENT options for all levels above that of the segment being inserted.
- GE** For GU, GN, GNP, ISRT, and STAT commands. DL/I was unable to find the segment you requested, or one or more of the parents of the segment you are trying to insert.
- GG** For Get commands. DL/I returns a GG status code to a program with a processing option of GOT or GON when the segment that the program is trying to retrieve contains an invalid pointer.
- GK** For unqualified GN and GNP commands. DL/I returned a segment that satisfies an unqualified GN or GNP request, but the segment is of a different segment type (but at the same level) than the last segment returned.
- II** For ISRT commands. The segment you are trying to insert already exists in the database. This code can also be returned if you have not established a path for the segment before trying to insert it. The segment you are trying to insert might match a segment with the same key in another hierarchy or database record.
- LB** For load programs only after issuing a LOAD command. The segment you are trying to load already exists in the database. DL/I returns this status code only for segments with key fields.
- NI** For ISRT and REPL commands. The segment you are trying to insert or replace requires a duplicate entry to be inserted in a secondary index that does not allow duplicate entries. This status code is returned for batch programs that write log records to direct access storage. If a CICS program that does not log to disk encounters this condition, the program (transaction) is abnormally terminated.

## Specifying the DL/I Interface Block (DIB)

**TG** For TERM commands. The program tried to terminate a PSB when one was not scheduled.

These DIBSTAT status codes indicate exceptional conditions, and are the only status codes returned to your program. All other status codes indicate error conditions and cause your transaction or batch program to abnormally terminate. If you want to pass control to an error routine from your CICS program, you can use the CICS HANDLE ABEND command; the last 2 bytes of the abend code are the IMS status code that caused the abnormal termination. For more information on the HANDLE ABEND command, see the application programming reference manual for your version of CICS. Batch BMP programs abend with abend 1041.

### 3. Segment Name (DIBSEGM)

This is the name of the lowest-level segment successfully accessed. When a retrieval is successful, this field contains the name of the retrieved segment. If the retrieval is unsuccessful, this field contains the last segment, along the path to the requested segment, that satisfies the command.

After issuing an XRST command, this field is either set to blanks (indicating a successful normal start), or a checkpoint ID (indicating the checkpoint ID from which the program was restarted).

You should test this field after issuing any of the following commands:

- GN
- GNP
- GU
- ISRT
- LOAD
- RETRIEVE
- XRST

### 4. Segment Level Number (DIBSEGLV)

This is the hierarchic level of the lowest-level segment retrieved. When IMS DB retrieves the segment you have requested, IMS DB places, in character format, the level number of that segment in this field. If you are issuing a path command, IMS DB places the number of the lowest-level segment retrieved. If IMS DB is unable to find the segment you have requested, it gives the level number of the last segment it encountered that satisfied your command. This is the lowest segment on the last path that IMS DB encountered while searching for the segment you requested.

You should test this field after issuing any of the listed commands:

- GN
- GNP
- GU
- ISRT
- LOAD
- RETRIEVE

### 5. Key Feedback Length (DIBKFBL)

This is a halfword field that contains the length of the concatenated key when you use the KEYFEEDBACK option with get commands. If your key feedback area is not long enough to contain the concatenated key, the key is truncated, and this area indicates the actual length of the full concatenated key.

### 6. Database Description Name (DIBDBDNM)

This is the fullword field that contains the name of the DBD. The DBD is the DL/I control block that contains all information used to describe a database. The DIBDBDNM field is returned only on a QUERY command.

### 7. Database Organization (DIBDBORG)

This is the fullword field that names the type of database organization (HDAM, HIDAM, HISAM, HSAM, GSAM, SHSAM, INDEX, or DEDB) padded to the right with blanks. The DIBDBORG field is returned only on a QUERY command.

---

## Defining a Key Feedback Area

To retrieve the concatenated key of a segment, you must define an area into which the key is placed. The concatenated key returned is that of the lowest-level segment retrieved. (The segment retrieved is indicated in the DIB by the DIBSEGM and DIBSEGLV fields.)

Specify the name of the area using the KEYFEEDBACK option on a GET command.

A concatenated key is made up of the key of a segment, plus the keys for all of its parents. For example, say you requested the concatenated key of the ILLNESS segment for January 2, 1988, for patient number 05142. **0514219880102** would be returned to your key feedback field. This number includes the key field of the ILLNESS segment, ILLDATE, concatenated to the key field of the PATIENT segment, PATNO.

If you define an area that is not long enough to contain the entire concatenated key, the key is truncated.

---

## Defining I/O Areas

You use I/O areas to pass segments back and forth between your program and the database. What an I/O area contains depends on the kind of command you are issuing:

- When you retrieve a segment, DL/I places the segment you requested in the I/O area.
- When you add a new segment, you build the new segment in the I/O area before issuing an ISRT command.
- Before you modify a segment, you first retrieve the segment into the then issue the DLET or REPL command.

**Restriction:** The I/O area must be long enough to contain the longest segment you retrieve from or add to the database. (Otherwise, you might experience storage overlap.) If you are retrieving, adding, or replacing multiple segments in one command, you must define an I/O area for each segment.

As an example of what a segment looks like in your I/O area, say that you retrieved the ILLNESS segment for Robert James, who came to the clinic on March 3, 1988. He was treated for strep throat. The data returned to your I/O area would look like this:

```
19880303STREPTHROA
```

## COBOL I/O Area

The I/O area in a COBOL program should be defined as a 01 level working storage entry. You can further define the area with 02 entries.

## Defining I/O Areas

```
IDENTIFICATION DIVISION.  
:  
:  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 INPUT-AREA.  
   02 KEY PICTURE X(6).  
   02 FIELD PICTURE X(84).
```

### PL/I I/O Area

In PL/I, the name for the I/O area used in the DL/I call can be the name of a fixed-length character string, a major structure, a connected array, or an adjustable character string. **Restriction:** The PL/I I/O area *cannot* be the name of a minor structure or a character string with the attribute VARYING. If you want to define it as a minor structure, you can use a pointer to the minor structure as the parameter.

Your program should define the I/O area as a fixed-length character string and pass the name of that string, or define it in one of the other ways described previously and then pass the pointer variable that points to that definition. If you want to use substructures or elements of an array, use the DEFINED or BASED attribute.

```
DECLARE   1 INPUT_AREA,  
         2 KEY CHAR(6),  
         2 FIELD CHAR(84);
```

### Assembler Language I/O Area

The I/O area in an assembler language program is formatted as follows:

```
IOAREA DS 0CL90  
KEY DS CL6  
FIELD DS CL84
```

---

## Chapter 3. Writing an Application Program

This chapter provides programming guidelines and information on preparing programs for execution using EXEC DLI commands. It also contains skeleton programs in assembler language, COBOL, PL/I, C, and C++.

The following topics provide additional information:

- “Programming Guidelines”
- “Preparing Your EXEC DLI Program for Execution” on page 29

---

### Programming Guidelines

This description provides some guidelines for writing efficient and error-free programs

The number, type, and sequence of the DL/I requests your program issues affect the efficiency of your program. A program that is poorly designed runs if it is coded correctly. The suggestions that follow can help you develop the most efficient design possible for your application program. Inefficiently designed programs can adversely affect performance and are hard to change. Being aware of how certain combinations of commands or calls affects performance helps you to avoid these problems and design a more efficient program.

After you have a general sequence of calls mapped out for your program, use these guidelines to improve the sequence. Usually an efficient sequence of requests causes efficient internal DL/I processing.

- Use the simplest call. Qualify your requests to narrow the search for DL/I, but do not use more qualification than required.
- Use the request or sequence of requests that gives DL/I the shortest path to the segment you want.
- Use the fewest number of requests possible in your program. Each DL/I request your program issues uses system time and resources. You may be able to eliminate unnecessary calls by:
  - Using path requests if you are replacing, retrieving, or inserting more than one segment in the same path. If you are using more than one request to do this, you are issuing unnecessary requests.
  - Changing the sequence so that your program saves the segment in a separate I/O area, and then gets it from that I/O area the second time it needs the segment. If your program retrieves the same segment more than once during program execution, you are issuing an unnecessary request.
  - Anticipating and eliminating needless and nonproductive requests, such as requests that result in GB, GE, and II status codes. For example, if you are issuing GNs for a particular segment type and you know how many occurrences of that segment type exist, do not issue the GN that results in a GE status code. You can keep track of the number of occurrences your program retrieves, and then continue with other processing when you know you have retrieved all the occurrences of that segment type.
  - Issuing an insert request with a qualification for each parent instead of issuing Get requests for the parents to make sure that they exist. When you

## Programming Guidelines

are inserting segments, you cannot insert dependents unless the parents exist. If DL/I returns a GE status code, at least one of the parents does not exist.

- Keep the main section of the program logic together. For example, branch to conditional routines, such as error and print routines, in other parts of the program, instead of having to branch around them to continue normal processing.
- Use call sequences that make good use of the physical placement of the data. Access segments in hierarchical sequence as much as possible. Avoid moving backward in the hierarchy.
- Process database records in order of the key field of the root segments. (For HDAM databases, this order depends on the randomizing routine that is used. Check with your DBA for this information.)
- Try to avoid constructing the logic of the program and the structure of commands or calls in a way that depends heavily on the database structure. Depending on the current structure of the hierarchy reduces the program's flexibility.

## Coding a Program in Assembler Language

The following sample is a CICS online program that is written in assembler language. It shows how the different parts of a command-level program fit together and how the EXEC DLI commands are coded.

Except for a few commands, this program applies to batch, BMP, and CICS programs. Any differences are highlighted in the notes for the sample assembler code. The numbering on the right of the sample code references these notes.

```
*ASM XOPTS(CICS,DLI)
*
R2      EQU  2
R3      EQU  3
R4      EQU  4
R11     EQU 11
R12     EQU 12
R13     EQU 13
DFHEISTG DSECT
SEGKEYA DS  CL4
SEGKEYB DS  CL4
SEGKEYC DS  CL4
SEGKEY1 DS  CL4
SEGKEY2 DS  CL4
CONKEYB DS  CL8
SEGNAME DS  CL8
SEGLEN  DS  H
PCBNUM  DS  H
AREAA   DS  CL80
AREAB   DS  CL80
AREAC   DS  CL80
AREAG   DS  CL250
AREASTAT DS CL360
*      COPY  MAPSET
*
*****
*  INITIALIZATION
*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE
*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND ROUTINE
*  RECEIVE INPUT MESSAGE
*****
SAMPLE  DFHEIENT CODEREG=(R2,R3),DATAREG=(R13,R12),EIBREG=R11
*
EXEC CICS HANDLE CONDITION ERROR(ERRORS)
```

## Coding a Program in Assembler Language

```

*
      EXEC CICS HANDLE ABEND LABEL(ABENDS)
*
      EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET')
*
      ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
*
*****
*   SCHEDULE PSB NAMED 'SAMPLE1'
*****
*
      EXEC DLI SCHD PSB(SAMPLE1)
      BAL  R4,TESTDIB          CHECK STATUS
*
*****
*   RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDENTS
*****
*
      MVC  SEGKEYA,=C'A300'
      EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
      SEGLENGTH(80) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)
      BAL  R4,TESTDIB          CHECK STATUS
GNPLOOP EQU  *
      EXEC DLI GNP USING PCB(1) INTO(AREAG) SEGLENGTH(250)
      CLC  DIBSTAT,=C'GE'      LOOK FOR END
      BE   LOOPDONE            DONE AT 'GE'
      BAL  R4,TESTDIB          CHECK STATUS
      B    GNPLOOP
LOOPDONE EQU  *
*
*****
*   INSERT NEW ROOT SEGMENT
*****
*
      MVC  AREAA,=CL80'DATA FOR NEW SEGMENT INCLUDING KEY'
      EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA)
      SEGLENGTH(80)
      BAL  R4,TESTDIB          CHECK STATUS
*
*****
*   RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
*****
*
      MVC  SEGKEYA,=C'A200'
      MVC  SEGKEYB,=C'B240'
      MVC  SEGKEYC,=C'C241'
      EXEC DLI GU USING PCB(1)
      SEGMENT(SEGA) WHERE(KEYA=SEGKEYA)
      FIELDLENGTH(4)
      INTO(AREAA)
      SEGLENGTH(80)
      SEGMENT(SEGB) WHERE(KEYB=SEGKEYB) FIELDLENGTH(4)
      INTO(AREAB)
      SEGLENGTH(80)
      SEGMENT(SEGC) WHERE(KEYC=SEGKEYC) FIELDLENGTH(4)
      INTO(AREAC)
      SEGLENGTH(80)
      BAL  R4,TESTDIB
*
      UPDATE FIELDS IN THE 3 SEGMENTS
      EXEC DLI REPL USING PCB(1)
      SEGMENT(SEGA) FROM(AREAA) SEGLENGTH(80)
      SEGMENT(SEGB) FROM(AREAB) SEGLENGTH(80)
      SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
      BAL  R4,TESTDIB          CHECK STATUS
*
*****
*   INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT
*****

```

## Coding a Program in Assembler Language

```

*
MVC AREAC,=CL80'DATA FOR NEW SEGMENT INCLUDING KEY'
MVC CONKEYB,=C'A200B240'
EXEC DLI ISRT USING PCB(1) X
      SEGMENT(SEGB) KEYS(CONKEYB) KEYLENGTH(8) X
      SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
BAL R4,TESTDIB CHECK STATUS
*
*****
* RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
* AND THEN DELETE IT AND ITS DEPENDENTS
*****
*
MVC CONKEYB,=C'A200B230'
EXEC DLI GU USING PCB(1) X
      SEGMENT(SEGB) X
      KEYS(CONKEYB) KEYLENGTH(8) X
      INTO(AREAB) SEGLENGTH(80)
BAL R4,TESTDIB CHECK STATUS
EXEC DLI DLET USING PCB(1) X
      SEGMENT(SEGB) SEGLENGTH(80) FROM(AREAB)
BAL R4,TESTDIB CHECK STATUS
*
*****
* RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,
* OBJECT SEGMENT WITH WHERE OPTION USING A LITERAL,
* AND THEN SET PARENTAGE
*
* USE VARIABLES FOR PCB INDEX, SEGMENT NAME, AND SEGMENT LENGTH
*****
*
MVC CONKEYB,=C'A200B230'
MVC SEGNAME,=CL8'SEGA'
MVC SEGLEN,=H'80'
MVC PCBNUM,=H'1'
EXEC DLI GU USING PCB(PCBNUM) X
      SEGMENT((SEGNAME)) X
      KEYS(CONKEYB) KEYLENGTH(8) SETPARENT X
      SEGMENT(SEGC) INTO(AREAC) SEGLENGTH(SEGLEN) X
      WHERE(KEYC='C520')
BAL R4,TESTDIB CHECK STATUS
*
*****
* RETRIEVE DATABASE STATISTICS
*****
*
EXEC DLI STAT USING PCB(1) INTO(AREASTAT) X
      VSAM FORMATTED LENGTH(360)
BAL R4,TESTDIB CHECK STATUS
*
*****
* RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS
*****
*
MVC SEGKEY1,=C'A050'
MVC SEGKEY2,=C'A150'
EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA) X
      SEGLENGTH(80) FIELDLENGTH(4,4,4,4) X
      WHERE(KEYA > SEGKEY1 AND KEYA < SEGKEY2
      KEYA > 'A275' AND KEYA < 'A350')
BAL R4,TESTDIB CHECK STATUS
*
*****
* TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED
*****
*
EXEC DLI TERM

```



```

*
*****
* SEND OUTPUT MESSAGE
*****
*
      EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET')           6
      EXEC CICS WAIT TERMINAL

*
*****
* COMPLETE TRANSACTION AND RETURN TO CICS
*****
*
      EXEC CICS RETURN                                           12

*
*****
* CHECK STATUS IN DIB
*****
*
TESTDIB EQU *
        CLC  DIBSTAT,=C' '      IS STATUS BLANK                13
        BER  R4                  YES - RETURN
*
        HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS
*
        BR   R4                  RETURN
ERRORS  EQU *
*
        HANDLE ERROR CONDITIONS
*
ABENDS  EQU *
*
        HANDLE ABENDS INCLUDING DLI ERROR STATUS CODES
*
      END

```

### Notes for the sample assembler code:

- 1** For a CICS online program containing EXEC DLI commands, you must specify the DLI and CICS options. For a batch or BMP program containing EXEC DLI, you must specify only the DLI option.
- 2** For reentry, define each of the areas the program uses—I/O areas, key feedback areas, and segment name areas in DFHEISTG.
- 3** Define an I/O area for each segment you retrieve, add, or replace (in a single command).
- 4** For a batch or BMP program containing EXEC DLI, you must save registers on entry and restore registers on exit according to z/OS register-saving conventions.
- 5** In a batch or BMP program, a DFHEIENT saves the registers on entry. Do not specify the EIBREG parameter in a batch program.
- 6** Do not code EXEC CICS commands in a batch or BMP program.
- 7** In a CICS online program, use the SCHD PSB command to obtain a PSB for the use of your program. Do *not* schedule a PSB in a batch or BMP program.
- 8** This GU command retrieves the first occurrence of SEGA with a key of A300. You do not have to provide the KEYLENGTH or SEGLENGTH options in an assembler language program.
- 9** This GNP command retrieves all dependents under segment SEGA. The GE status code indicates that no more dependents exist.
- 10** This GU command is an example of a path command. Use a separate I/O area for each segment you retrieve.
- 11** In a CICS online program, the TERM command terminates the PSB scheduled earlier. You do *not* terminate the PSB in a batch or BMP program.

## Coding a Program in Assembler Language

**12** For a batch or BMP program, code DFHEIRET with an optional RCREG parameter instead of EXEC CICS RETURN. The RCREG parameter identifies a register containing the return code.

**13** After issuing each command, you should check the status code in the DIB.

## Coding a Program in COBOL

The following sample program is written in COBOL. It shows how the different parts of a command-level program fit together, and how the EXEC DLI commands are coded. The sample program applies to the COBOL V4 compiler (5734-CB2), the OS/VS COBOL compiler (5740-CB1), IBM COBOL for z/OS & VM (5688-197), and the VS COBOL II compiler (5668-958 and 5668-940).

Except for a few commands, this program applies to batch, BMP, and CICS programs. Any differences are highlighted in the notes for the sample COBOL code. The numbering on the right of the sample code references the notes.

```
CBL LIB,APOST,XOPTS(CICS,DLI)          IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 SEGKEYA          PIC X(4).
77 SEGKEYB          PIC X(4).
77 SEGKEYC          PIC X(4).
77 SEGKEY1          PIC X(4).
77 SEGKEY2          PIC X(4).
77 SEGKEY3          PIC X(4).
77 SEGKEY4          PIC X(4).
77 CONKEYB          PIC X(8).
77 SEGNAME          PIC X(8).
77 SEGLEN           COMP PIC S9(4).
77 PCBNUM           COMP PIC S9(4).
01 AREAA            PIC X(80).
* DEFINE SEGMENT I/O AREA
01 AREAB            PIC X(80).
01 AREAC            PIC X(80).
01 AREAG            PIC X(250).
01 AREASTAT        PIC X(360).
* COPY MAPSET.
PROCEDURE DIVISION.
*
* *****
*  INITIALIZATION
*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE
*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND ROUTINE
*  RECEIVE INPUT MESSAGE
* *****
*
* EXEC CICS HANDLE CONDITION ERROR(ERRORS) END-EXEC.
*
* EXEC CICS HANDLE ABEND LABEL(ABENDS) END-EXEC.
*
* EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET') END-EXEC.
* ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
*
* *****
*  SCHEDULE PSB NAMED 'SAMPLE1'
* *****
*
* EXEC DLI SCHD PSB(SAMPLE1) END-EXEC.
* PERFORM TEST-DIB THRU OK.
```

```

*
* *****
* RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDENTS
* *****
*
    MOVE 'A300' TO SEGKEYA.
    EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
        SEGLENGTH(80) WHERE(KEYA=SEGKEYA)
        FIELDLENGTH(4)
    END-EXEC.
    PERFORM TEST-DIB THRU OK.
GNPLOOP.
    EXEC DLI GNP USING PCB(1) INTO(AREAG) SEGLENGTH(250)
    END-EXEC.
    IF DIBSTAT EQUAL TO 'GE' THEN GO TO LOOPDONE.
    PERFORM TEST-DIB THRU OK.
    GO TO GNPLOOP.
LOOPDONE.
*
* *****
* INSERT NEW ROOT SEGMENT
* *****
*
    MOVE 'DATA FOR NEW SEGMENT INCLUDING KEY' TO AREAA.
    EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA)
        SEGLENGTH(80) END-EXEC.
    PERFORM TEST-DIB THRU OK.
*
* *****
* RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
* *****
*
    MOVE 'A200' TO SEGKEYA.
    MOVE 'B240' TO SEGKEYB.
    MOVE 'C241' TO SEGKEYC.
    EXEC DLI GU USING PCB(1)
        SEGMENT(SEGA) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)
        INTO(AREAA)
        SEGLENGTH(80)
        SEGMENT(SEGB) WHERE(KEYB=SEGKEYB) FIELDLENGTH(4)
        INTO(AREAB)
        SEGLENGTH(80)
        SEGMENT(SEGC) WHERE(KEYC=SEGKEYC) FIELDLENGTH(4)
        INTO(AREAC)
        SEGLENGTH(80)
    END-EXEC.
    PERFORM TEST-DIB THRU OK.
*
    UPDATE FIELDS IN THE 3 SEGMENTS
    EXEC DLI REPL USING PCB(1)
        SEGMENT(SEGA) FROM(AREAA) SEGLENGTH(80)
        SEGMENT(SEGB) FROM(AREAB) SEGLENGTH(80)
        SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
    END-EXEC.
    PERFORM TEST-DIB THRU OK.
*
* *****
* INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT
* *****
*
    MOVE 'DATA FOR NEW SEGMENT INCLUDING KEY' TO AREAC.
    MOVE 'A200B240' TO CONKEYB.
    EXEC DLI ISRT USING PCB(1)
        SEGMENT(SEGB) KEYS(CONKEYB) KEYLENGTH(8)
        SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
    END-EXEC.
    PERFORM TEST-DIB THRU OK.
*

```

6

7

## Coding a Program in COBOL

```
* *****
* RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
* AND THEN DELETE IT AND ITS DEPENDENTS
* *****
*
  MOVE 'A200B230' TO CONKEYB.
  EXEC DLI GU USING PCB(1)
    SEGMENT(SEGB)
      KEYS(CONKEYB) KEYLENGTH(8)
      INTO(AREAB) SEGLENGTH(80)
  END-EXEC.
  PERFORM TEST-DIB THRU OK.
  EXEC DLI DLET USING PCB(1)
    SEGMENT(SEGB) SEGLENGTH(80) FROM(AREAB) END-EXEC.
  PERFORM TEST-DIB THRU OK.
*
* *****
* RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,
* OBJECT SEGMENT WITH WHERE OPTION,
* AND THEN SET PARENTAGE
*
* USE VARIABLES FOR PCB INDEX, SEGMENT NAME, AND SEGMENT LENGTH
* *****
*
  MOVE 'A200B230' TO CONKEYB.
  MOVE 'C520' TO SEGKEYC.
  MOVE 'SEGA' TO SEGNAME.
  MOVE 80 TO SEGLEN.
  MOVE 1 TO PCBNUM.
  EXEC DLI GU USING PCB(PCBNUM)
    SEGMENT((SEGNAME))
      KEYS(CONKEYB) KEYLENGTH(8) SETPARENT
      SEGMENT(SEGC) INTO(AREAC) SEGLENGTH(SEGLEN)
      WHERE(KEYC=SEGKEYC) FIELDLENGTH(4) END-EXEC.
  PERFORM TEST-DIB THRU OK.
*
* *****
* RETRIEVE DATABASE STATISTICS
* *****
*
  EXEC DLI STAT USING PCB(1) INTO(AREASTAT)
    VSAM FORMATTED LENGTH(360) END-EXEC.
  PERFORM TEST-DIB THRU OK.
*
* *****
* RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS
* *****
*
  MOVE 'A050' TO SEGKEY1.
  MOVE 'A150' TO SEGKEY2.
  MOVE 'A275' TO SEGKEY3.
  MOVE 'A350' TO SEGKEY4.
  EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
    SEGLENGTH(80) FIELDLENGTH(4,4,4,4)
    WHERE(KEYA > SEGKEY1 AND KEYA < SEGKEY2 OR
      KEYA > SEGKEY3 AND KEYA < SEGKEY4)
  END-EXEC.
  PERFORM TEST-DIB THRU OK.
*
* *****
* TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED
* *****
*
  EXEC DLI TERM END-EXEC.
```

```

* SEND OUTPUT MESSAGE
* *****
*
* EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET') END-EXEC.
* EXEC CICS WAIT TERMINAL END-EXEC.
*
* *****
* COMPLETE TRANSACTION AND RETURN TO CICS
* *****
*
* EXEC CICS RETURN END-EXEC.
*
* *****
* CHECK STATUS IN DIB
* *****
*
* TEST-DIB.
* IF DIBSTAT EQUAL TO ' ' THEN GO TO OK.
* OK.
* ERRORS.
* HANDLE ERROR CONDITIONS
* ABENDS.
* HANDLE ABENDS INCLUDING DLI ERROR STATUS CODES

```

**9**

### Notes for the sample COBOL code:

- 1** For a CICS online program containing EXEC DLI commands, you must specify the DLI and CICS options. For a batch or BMP program containing EXEC DLI, you must specify only the DLI option.
- 2** Define each of the areas the program uses—I/O areas, key feedback areas, and segment name areas—as 77- or 01-level working storage entries.
- 3** Define an I/O area for each segment you retrieve, add, or replace (in a single command).
- 4** Do not code EXEC CICS commands in a batch or BMP program.
- 5** For CICS online programs, you use a SCHD PSB command to obtain a PSB. You do *not* schedule a PSB in a batch or BMP program.
- 6** This GU command retrieves the first occurrence of SEGA with a key of A300. KEYLENGTH and SEGLENGTH are optional for IBM COBOL for z/OS & VM (and VS COBOL II). For COBOL V4 and OS/VS COBOL, KEYLENGTH and SEGLENGTH are required.
- 7** This GU command is an example of a path command. You must use a separate I/O area for each segment you retrieve.
- 8** For a CICS online program, the TERM command terminates the PSB scheduled earlier. You do *not* terminate the PSB in a batch or BMP program.
- 9** After issuing each command, you should check the status code in the DIB.

## Coding a Program in PL/I

The following sample program is written in PL/I. It shows how the different parts of a command-level program fit together and how the EXEC DLI commands are coded.

Except for a few commands, this program applies to batch, BMP, and CICS programs. Any differences are highlighted in the notes for the sample PL/I code. The numbering on the right of the sample code references those notes.

```

*PROCESS INCLUDE,GN,XOPTS(CICS,DLI);
SAMPLE: PROCEDURE OPTIONS(MAIN);
DCL SEGKEYA CHAR (4);
DCL SEGKEYB CHAR (4);

```

**1**

**2**

## Coding a Program in PL/I

```

DCL      SEGKEYC                CHAR (4);
DCL      SEGKEY1                CHAR (4);
DCL      SEGKEY2                CHAR (4);
DCL      SEGKEY3                CHAR (4);
DCL      SEGKEY4                CHAR (4);
DCL      CONKEYB                CHAR (8);
DCL      SEGNAME                CHAR (8);
DCL      PCBNUM                 FIXED BIN (15);
DCL      AREAA                  CHAR (80);
/*      DEFINE SEGMENT I/O AREA
DCL      AREAB                  CHAR (80);
DCL      AREAC                  CHAR (80);
DCL      AREAG                  CHAR (250);
DCL      AREASTAT               CHAR (360);
%INCLUDE MAPSET
/*
/*
/* *****
/*  INITIALIZATION
/*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE
/*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND PROGRAM
/*  RECEIVE INPUT MESSAGE
/* *****
/*
EXEC CICS HANDLE CONDITION ERROR(ERRORS);
/*
EXEC CICS HANDLE ABEND PROGRAM('ABENDS');
/*
EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET');
/*  ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
/*
/* *****
/*  SCHEDULE PSB NAMED 'SAMPLE1'
/* *****
/*
EXEC DLI SCHD PSB(SAMPLE1);
CALL TEST_DIB;

/* *****
/*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDENTS
/* *****
/*
SEGKEYA = 'A300';
EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
WHERE(KEYA=SEGKEYA);
CALL TEST_DIB;
GNPLOOP:
EXEC DLI GNP USING PCB(1) INTO(AREAG);
IF DIBSTAT = 'GE' THEN GO TO LOOPDONE;
CALL TEST_DIB;
GO TO GNPLOOP;
LOOPDONE:
/*
/* *****
/*  INSERT NEW ROOT SEGMENT
/* *****
/*
AREAA = 'DATA FOR NEW SEGMENT INCLUDING KEY';
EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA);
CALL TEST_DIB;
/*
/* *****
/*  RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
/* *****
/*
SEGKEYA = 'A200';
SEGKEYB = 'B240';

```

```

SEGKEYC = 'C241';
EXEC DLI GU USING PCB(1)
  SEGMENT(SEGA) WHERE(KEYA=SEGKEYA)
  INTO(AREAA)
  SEGMENT(SEGB) WHERE(KEYB=SEGKEYB)
  INTO(AREAB)
  SEGMENT(SEGC) WHERE(KEYC=SEGKEYC)
  INTO(AREAC);
CALL TEST_DIB;
/* UPDATE FIELDS IN THE 3 SEGMENTS */
EXEC DLI REPL USING PCB(1)
  SEGMENT(SEGA) FROM(AREAA)
  SEGMENT(SEGB) FROM(AREAB)
  SEGMENT(SEGC) FROM(AREAC);
CALL TEST_DIB;
/*
/* ***** */
/* INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT */
/* ***** */
/*
AREAC = 'DATA FOR NEW SEGMENT INCLUDING KEY';
CONKEYB = 'A200B240';
EXEC DLI ISRT USING PCB(1)
  SEGMENT(SEGB) KEYS(CONKEYB)
  SEGMENT(SEGC) FROM(AREAC);
CALL TEST_DIB;
/*
/* ***** */
/* RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY */
/* AND THEN DELETE IT AND ITS DEPENDENTS */
/* ***** */
/*
CONKEYB = 'A200B230';
EXEC DLI GU USING PCB(1)
  SEGMENT(SEGB)
  KEYS(CONKEYB)
  INTO(AREAB);
CALL TEST_DIB;
EXEC DLI DLET USING PCB(1)
  SEGMENT(SEGB) FROM(AREAB);
CALL TEST_DIB;
/*
/* ***** */
/* RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY, */
/* OBJECT SEGMENT WITH WHERE OPTION */
/* AND THEN SET PARENTAGE */
/*
/* USE VARIABLES FOR PCB INDEX, SEGMENT NAME */
/* ***** */
/*
CONKEYB = 'A200B230';
SEGNAME = 'SEGA';
SEGKEYC = 'C520';
PCBNUM = 1;
EXEC DLI GU USING PCB(PCBNUM)
  SEGMENT((SEGNAME))
  KEYS(CONKEYB) SETPARENT
  SEGMENT(SEGC) INTO(AREAC)
  WHERE(KEYC=SEGKEYC);
CALL TEST_DIB;
/*
/* ***** */
/* RETRIEVE DATABASE STATISTICS */
/* ***** */
/*
EXEC DLI STAT USING PCB(1) INTO(AREASTAT) VSAM FORMATTED;
CALL TEST_DIB;

```

## Coding a Program in PL/I

```
/* */
/* ***** */
/* RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS */
/* ***** */
/* */
SEGKEY1 = 'A050';
SEGKEY2 = 'A150';
SEGKEY3 = 'A275';
SEGKEY4 = 'A350';
EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
      WHERE(KEYA > SEGKEY1 AND KEYA < SEGKEY2 OR
            KEYA > SEGKEY3 AND KEYA < SEGKEY4);
CALL TEST_DIB;
/* */
/* ***** */
/* TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED */
/* ***** */
/* */

EXEC DLI TERM; 9

/* */
/* ***** */
/* SEND OUTPUT MESSAGE */
/* ***** */
/* */
EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET'); 4
EXEC CICS WAIT TERMINAL;
/* */
/* ***** */
/* COMPLETE TRANSACTION AND RETURN TO CICS */
/* ***** */
/* */
EXEC CICS RETURN; 4
/* */
/* ***** */
/* CHECK STATUS IN DIB */
/* ***** */
/* */
TEST_DIB: PROCEDURE; 10
  IF DIBSTAT = ' ' RETURN;

  /* HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS */
  /* */
OK:
END TEST_DB;
ERRORS:
  /* HANDLE ERROR CONDITIONS */
  /* */
END SAMPLE;
```

### Notes to the sample PL/I code:

- 1** For a CICS online program containing EXEC DLI commands, you must specify the DLI and CICS options. For a batch or BMP program containing EXEC DLI, you must specify only the DLI option.
- 2** Define, in automatic storage, each of the areas; I/O areas, key feedback areas, and segment name areas.
- 3** Define an I/O area for each segment you retrieve, add, or replace in a single command.
- 4** Do not code EXEC CICS commands in a batch or BMP program.
- 5** For CICS online programs, you use a SCHD PSB command to obtain a PSB. You do *not* schedule a PSB in a batch or BMP program.



**6** This GU command retrieves the first occurrence of SEGA with a key of A300. Notice that you do not need to include the KEYLENGTH and SEGLENGTH options.

**7** This GNP command retrieves all dependents under segment SEGA. The GE status code indicates that no more dependents exist.

**8** This GU command is an example of a path command. You must use a separate I/O area for each segment you retrieve.

**9** For a CICS online program, the TERM command terminates the PSB scheduled earlier. You do *not* terminate the PSB in a batch or BMP program.

**10** After issuing each command, you should check the status code in the DIB.

## Coding a Program in C

The following sample program is written in C. It shows how the different parts of a command-level program fit together and how the EXEC DLI commands are coded.

Except for a few commands, this program applies to batch, BMP, and CICS programs. Any differences are highlighted in the notes for the sample C code. The numbering on the right of the sample code references those notes.

```
#include < string.h> 1
#include < stdio.h > 2

char DIVIDER[120] = "-----\
-----";
char BLANK[120] = " \
 \0";
char BLAN2[110] = " \0";

char SCHED[120] = "Schedule PSB(PC3COCHD) " 3
char GN1[120] = "GN using PCB(2) Segment(SE2ORDER) check dibstat \
is blank";
char GNP1[120] = "GNP using PCB(2) check dibstat = GK or blank \
(or GE for last GNP)";
char GU1[120] = "GU using PCB(2) Segment(SE2ORDER) where(\
FE20GREF=000000') check dibstat blank";
char GU2[120] = "GU using PCB(2) Segment(SE2ORDER) where(\
FE20GREF=000999') check dibstat blank";
char REP1[120] = "REPLACE using PCB(2) Segment(SE2ORDER) check \
dibstat is blank";
char DEL1[120] = "DELETE using PCB(2) Segment(SE2ORDER) check \
dibstat is blank";
char INS1[120] = "INSERT using PCB(2) Segment(SE2ORDER) where\
(FE20GREF='000999') check dibstat is blank";
char TERM[120] = "TERM - check dibstat is blank";
char STAT[120] = "STAT USING PCB(2) VSAM FORMATTED";
char DATAB[6] = "000999";
char DATAC[114] = " REGRUN TEST INSERT N01.";
char START[120] = "PROGXIV STARTING";
char OKMSG[120] = "PROGXIV COMPLETE";
int TLINE = 120;
int L11 = 11;
int L360 = 11;
struct {
    char NEWSEGB[6];
    char NEWSEGC[54];
} NEWSEG;
char OUTLINE[120]; 4
struct {
    char OUTLINA[9];
    char OUTLINB[111];
} OUTLIN2;
```

## Coding a Program in C

```
struct {
    char OUTLINX[9];
    char OUTLINY[6];
    char OUTLINZ[105];
} OUTLIN3;
char GUIOA[60];
char GNIOA[60];
struct {
    char ISRT1[6];
    char ISRT2[54];
} ISRTIOA;
struct {
    char REPLIO1[6];
    char REPLIO2[54];
} REPLIOA;
struct {
    char DLET1[6];
    char DLET2[54];
} DLETIOA;
struct {
    char STATA1[120];
    char STATA2[120];
    char STATA3[120];
} STATAAREA;
struct {
    char DHPART[2];
    char RETCODE[2]
} DHABCODE;

main()
{
    EXEC CICS ADDRESS EIB(dfheiptr);
    strcpy(OUTLINE,DIVIDER);
    SENDLINE();
    strcpy(OUTLINE,START);
    SENDLINE();
    /*
    /* SCHEDULE PSB
    /*
    /*
    strcpy(OUTLINE,SCHED);
    SENDLINE();
    EXEC DLI SCHEDULE PSB(PC3COCHD);
    SENDSTAT();
    TESTDIB();
    /*
    /* ISSUE GU REQUEST
    /*
    /*
    strcpy(OUTLINE,GU1);
    SENDLINE();
    EXEC DLI GET UNIQUE USING PCB(2)
    SEGMENT(SE2ORDER)
    WHERE(FE20GREF>="000000")
    INTO(&GUIOA) SEGLLENGTH(60);
    strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
    strcpy(OUTLIN2.OUTLINB,GUIOA);
    SENDLIN2();
    SENDSTAT();
    TESTDIB();
    /*
    /* ISSUE GNP REQUEST
    /*
    /*
do {
    strcpy(OUTLINE,GNP1);
    SENDLINE();
    EXEC DLI GET NEXT IN PARENT USING PCB(2)
    INTO(&GNIOA) SEGLLENGTH(60);
    strcpy(OUTLIN2.OUTLINA,"SEGMENT=");
```

```

        strcpy(OUTLIN2.OUTLINB,GNIOA);
        SENDLIN2();
        SENDSTAT();
        if (strncmp(dibptr->dibstat,"GE",2) != 0)
            TESTDIB();
} while (strncmp(dibptr->dibstat,"GE",2) != 0);
/*
/* ISSUE GN REQUEST
/*
        strcpy(OUTLINE,GN1);
        SENDLINE();
        EXEC DLI GET NEXT USING PCB(2)
            SEGMENT(SE2ORDER)
            INTO(&GNIOA) SEGLLENGTH(60);
        strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
        strcpy(OUTLIN2.OUTLINB,GNIOA);
        SENDLIN2();
        SENDSTAT();
        TESTDIB();
/*
/* INSERT SEGMENT
/*
        strcpy(OUTLINE,INS1);
        SENDLINE();
        strcpy(NEWSEG.NEWSEGB,DATAB);
        strcpy(NEWSEG.NEWSEGC,DATAC);
        strcpy(ISRTIOA.ISRT1,NEWSEG.NEWSEGB);
        strcpy(ISRTIOA.ISRT2,NEWSEG.NEWSEGC);
        strcpy(OUTLIN3.OUTLINX,"ISRT SEG=");
        strcpy(OUTLIN3.OUTLINY,ISRTIOA.ISRT1);
        strcpy(OUTLIN3.OUTLINZ,ISRTIOA.ISRT2);
        SENDLIN3();
        EXEC DLI ISRT USING PCB(2)
            SEGMENT(SE2ORDER)
            FROM(&ISRTIOA) SEGLLENGTH(60);
        SENDSTAT();
        if (strncmp(dibptr->dibstat,"II",2) == 0)
            strncpy(dibptr->dibstat," ",2);
        TESTDIB();
/*
/* ISSUE GN REQUEST
/*
        strcpy(OUTLINE,GN1);
        SENDLINE();
        EXEC DLI GET NEXT USING PCB(2)
            SEGMENT(SE2ORDER)
            INTO(&GNIOA) SEGLLENGTH(60);
        strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
        strcpy(OUTLIN2.OUTLINB,GNIOA);
        SENDLIN2();
        SENDSTAT();
        TESTDIB();
/*
/* GET INSERTED SEGMENT TO BE REPLACED
/*
        strcpy(OUTLINE,GU2);
        SENDLINE();
        EXEC DLI GET UNIQUE USING PCB(2)
            SEGMENT(SE2ORDER)
            WHERE(FE20GREF="000999")
            INTO(&ISRTIOA) SEGLLENGTH(60);
        strcpy(OUTLIN3.OUTLINX,"ISRT SEG=");
        strcpy(OUTLIN3.OUTLINY,ISRTIOA.ISRT1);
        strcpy(OUTLIN3.OUTLINZ,ISRTIOA.ISRT2);
        SENDLIN3();
        SENDSTAT();
        TESTDIB();

```

## Coding a Program in C

```
/*                                     */
/* REPLACE SEGMENT                     */
/*                                     */
    strcpy(OUTLINE,REP1);
    SENDLINE();
    strcpy(REPLIOA.REPLIO1,DATAB);
    strcpy(REPLIOA.REPLIO2,"REGRUN REPLACED SEGMENT NO1.");
    strcpy(OUTLIN3.OUTLINX,"REPL SEG=");
    strcpy(OUTLIN3.OUTLINY,REPLIOA.REPLIO1);
    strcpy(OUTLIN3.OUTLINZ,REPLIOA.REPLIO2);
    SENDLIN3();
    EXEC DLI REPLACE USING PCB(2)
        SEGMENT(SE2ORDER)
        FROM(&REPLIOA) SEGLLENGTH(60);
    SENDSTAT();
    TESTDIB();
/*                                     */
/* ISSUE GN REQUEST                     */
/*                                     */
    strcpy(OUTLINE,GN1);
    SENDLINE();
    EXEC DLI GET NEXT USING PCB(2)
        SEGMENT(SE2ORDER)
        INTO(&GNIOA) SEGLLENGTH(60);
    strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
    strcpy(OUTLIN2.OUTLINB,GNIOA);
    SENDLIN2();
    SENDSTAT();
    TESTDIB();
/*                                     */
/* GET REPLACED SEGMENT                 */
/*                                     */
    strcpy(OUTLINE,GU2);
    SENDLINE();
    EXEC DLI GET UNIQUE USING PCB(2)
        SEGMENT(SE2ORDER)
        WHERE(FE20GREF="000999")
        INTO(&REPLIOA) SEGLLENGTH(60);
    strcpy(OUTLIN3.OUTLINX,"REPL SEG=");
    strcpy(OUTLIN3.OUTLINY,REPLIOA.REPLIO1);
    strcpy(OUTLIN3.OUTLINZ,REPLIOA.REPLIO2);
    SENDLIN3();
    SENDSTAT();
    TESTDIB();
/*                                     */
/* ISSUE DELETE REQUEST                 */
/*                                     */
    strcpy(OUTLINE,DEL1);
    SENDLINE();
    strcpy(DLETIOA.DLET1,REPLIOA.REPLIO1);
    strcpy(DLETIOA.DLET2,REPLIOA.REPLIO2);
    strcpy(OUTLIN3.OUTLINX,"DLET SEG=");
    strcpy(OUTLIN3.OUTLINY,DLETIOA.DLET1);
    strcpy(OUTLIN3.OUTLINZ,DLETIOA.DLET2);
    SENDLIN3();
    EXEC DLI DELETE USING PCB(2)
        SEGMENT(SE2ORDER)
        FROM(&DLETIOA) SEGLLENGTH(60);
    SENDSTAT();
    TESTDIB();
/*                                     */
/* ISSUE STAT REQUEST                   */
/*                                     */
    strcpy(OUTLINE,STAT);
    SENDLINE();
    EXEC DLI STAT USING PCB(2)
        VSAM FORMATTED
```

```

        INTO(&STATAREA);
        SENDSTT2();
        TESTDIB();
/*                                     */
/* ISSUE TERM REQUEST                 */
/*                                     */
        strcpy(OUTLINE,TERM);
        SENDLINE();
        EXEC DLI TERM;
        SENDSTAT();
        TESTDIB();
        strcpy(OUTLINE,DIVIDER);
        SENDLINE();
        SENDOK();
/*                                     */
/* RETURN TO CICS                     */
/*                                     */
        EXEC CICS RETURN;
}
/*                                     */
/*                                     */
/*                                     */
SENDLINE()
{
        EXEC CICS SEND FROM(OUTLINE) LENGTH(120);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLINE) LENGTH(TLINE);
        strcpy(OUTLINE,BLANK);
        return;
}
SENDLIN2()
{
        EXEC CICS SEND FROM(OUTLIN2) LENGTH(120);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLIN2) LENGTH(TLINE);
        strcpy(OUTLIN2.OUTLINA,BLANK,9);
        strcpy(OUTLIN2.OUTLINB,BLANK,111);
        return;
}
SENDLIN3()
{
        EXEC CICS SEND FROM(OUTLIN3) LENGTH(120);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLIN3) LENGTH(TLINE);
        strcpy(OUTLIN3.OUTLINX,BLANK,9);
        strcpy(OUTLIN3.OUTLINY,BLANK,6);
        strcpy(OUTLIN3.OUTLINZ,BLANK,105);
        return;
}
SENDSTAT()
{
        strncpy(OUTLIN2.OUTLINA,BLANK,9);
        strncpy(OUTLIN2.OUTLINB,BLANK,110);
        strcpy(OUTLIN2.OUTLINA," DIBSTAT=");
        strcpy(OUTLIN2.OUTLINB,dibptr->dibstat);
        EXEC CICS SEND FROM(OUTLIN2) LENGTH(11);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLIN2) LENGTH(L11);
        strcpy(OUTLINE,DIVIDER);
        SENDLINE();
        return;
}
SENDSTT2()
{
        strncpy(OUTLIN2.OUTLINA,BLANK,9);
        strncpy(OUTLIN2.OUTLINB,BLANK,110);
        strcpy(OUTLIN2.OUTLINA," DIBSTAT=");

```

19

20

## Coding a Program in C

```
        strcpy(OUTLIN2.OUTLINB,dibptr->dibstat);
        EXEC CICS SEND FROM(STATAREA) LENGTH(360);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(STATAREA)
            LENGTH(L360);
        return;
    }

SENDOK()
{
    EXEC CICS SEND FROM(OKMSG) LENGTH(120);
    EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OKMSG) LENGTH(TLINE);
    return;
}

TESTDIB() 21
{
    if (strncmp(dibptr->dibstat," ",2) == 0)
        return;
    else if (strncmp(dibptr->dibstat,"GK",2) == 0)
        return;
    else if (strncmp(dibptr->dibstat,"GB",2) == 0)
        return;
    else if (strncmp(dibptr->dibstat,"GE",2) == 0)
        return;
    else
    {
        EXEC CICS ABEND ABCODE("PETE"); 22
        EXEC CICS RETURN;
    }
    return;
}
```

### Notes for the sample C code:

- 1** You must include a standard header file `string.h` to gain access to string manipulation facilities.
- 2** You must include standard header file `stdio.h` to access the standard I/O library.
- 3** Define DL/I messages.
- 4** Define the I/O areas.
- 5** Program start.
- 6** Define PSB PC3COCHD.
- 7** Issue the first command. Retrieves the first occurrence of segment SE2ORDER and puts it into array OUTLIN2.
- 8** Issue the GNP command to get the next segment and put it into array OUTLIN2.
- 9** GE status codes indicate no more segments to get.
- 10** Get next segment SE2ORDER and put it into the array OUTLIN2.
- 11** Insert segment into array OUTLIN3.
- 12** Issue GN to retrieve next segment and put it into array OUTLIN2.
- 13** Get next segment that will be replaced and put it into OUTLIN3.
- 14** Replace the segment and put it into array OUTLIN3.
- 15** Get next segment and put it into array OUTLIN2.
- 16** Get the replaced segment and put it into array OUTLIN3.
- 17** Issue DELETE command after putting content of segment into array OUTLIN3.
- 18** Issue STAT REQUEST command.

- 19** Issue TERM command.
- 20** Output processing.
- 21** Check return code.
- 22** Do not code EXEC CICS commands in a batch or BMP program.

---

## Preparing Your EXEC DLI Program for Execution

The steps for preparing your program for execution are as follows:

1. Run the CICS command language translator to translate the EXEC DLI and EXEC CICS commands. COBOL, PL/I, and assembler language programs have separate translators.
2. Compile your program.
3. Link-edit:
  - An online program with the appropriate CICS interface module
  - A batch or BMP program with the IMS interface module.

You can use CICS-supplied procedures to translate, compile, and link-edit your program. The procedure you use depends on the type of program (batch, BMP, or CICS online) and the language it is written in (COBOL, PL/I, or assembler language).

## Translator Options Required for EXEC DLI

Even when you use the CICS-supplied procedures for preparing your program, you must supply certain translator options.

For a CICS online program containing EXEC DLI commands, you must specify the DLI and CICS options. For a batch or BMP program containing EXEC DLI commands, you must specify the DLI option..

You can also specify the options on the EXEC job control statement that invokes the translator; if you use both methods, the CBL and \*PROCESS statement overrides those in the EXEC statement. For more information on the translator options, see *CICS Transaction Server for CICS Application Programming Guide*.

You must ensure that the translator options you use in a COBOL program do not conflict with the COBOL compiler options. When you translate an IBM COBOL for z/OS & VM program, you must use the COBOL for z/OS & VM translator option. Similarly, when you translate a VS COBOL II program, you must use the COBOL II translator option.

## Compiler Options Required for EXEC DLI

If you want to compile your batch COBOL program with COBOL for z/OS & VM and then execute it AMODE(31) on z/OS, you must use the compiler option RENT. If you want to compile your batch COBOL program with VS COBOL II and then execute it AMODE(31) on z/OS, you must use the compiler options RES and RENT. For information on which compiler options should be used for a CICS program, see *CICS Application Programming Reference*.

## Linkage Editor Options Required for EXEC DLI

If the compiler being used supports it, you can link a program written with EXEC commands as AMODE(31) RMODE(ANY).

## Preparing your Program for Execution



---

## Chapter 4. EXEC DLI Commands for an Application Program

This chapter explains the program specification block (PSB) and the different kinds of program communication blocks (PCBs). It also lists and describes the EXEC DLI commands. For each command, a syntax diagram is provided, along with information on options, restrictions, usage, and examples illustrating that usage.

The following topics provide additional information:

- “PCBs and PSB”
- “EXEC DLI Commands” on page 33

---

### PCBs and PSB

A program specification block (PSB) used in a DBCTL environment can contain:

- I/O PCB
- Alternate PCBs
- DB PCBs
- GSAM PCBs

#### I/O PCB

In a DBCTL environment, an I/O PCB is needed to issue DBCTL service requests. Unlike the other types of PCB, it is not defined with PSB generation, but if the application program is using an I/O PCB, this has to be indicated in the PSB scheduling request.

#### Alternate PCB

An alternate PCB defines a logical terminal and can be used instead of the I/O PCB when it is necessary to direct a response to a terminal. Alternate PCBs appear in PSBs used in a CICS-DBCTL environment, but are used only in an IMS DC environment. CICS applications using DBCTL cannot successfully issue commands that specify an alternate PCB, an MSDB PCB, or a GSAM PCB. However, a PSB that contains PCBs of these types can be scheduled successfully in a CICS-DBCTL environment.

Alternate PCBs are included in the PCB address list returned to a call level application program. In an EXEC DLI application program, the existence of alternate PCBs in the PSB affects the PCB number used in the PCB keyword.

#### DB PCB

A database PCB (DB PCB) is the PCB that defines an application program's interface to a database. One DB PCB is needed for each database view used by the application program. It can be a full-function PCB, a DEDB PCB, or an MSDB PCB.

#### GSAM PCB

A GSAM PCB defines an application program's interface for GSAM operations.

When using DBCTL, a CICS program receives, by default, a DB PCB as the first PCB in the parameter list passed to it after scheduling. However, when your application program can handle an I/O PCB, you indicate this using the

SYSSERVE keyword on the SCHED command. The I/O PCB is then the first PCB in the parameter address list passed back to your application program.

## PCB Summary

This topic summarizes the information concerning I/O PCBs and alternate PCBs in various types of application programs.

**Recommendation:** You should read this topic if you intend to issue system service requests.

### DB Batch Programs

Alternate PCBs are always included in the list of PCBs supplied to the program by DL/I irrespective of whether you have specified CMPAT=Y. The I/O PCB is returned depending on the CMPAT option.

If you specify CMPAT=Y, the PCB list contains the address of the I/O PCB, followed by the addresses of any alternate PCBs, followed by the addresses of any DB PCBs.

If you do not specify CMPAT=Y, the PCB list contains the addresses of any alternate PCBs followed by the addresses of the DB PCBs.

### BMP Programs, MPPs, and IFPs

I/O PCBs and alternate PCBs are always passed to BMP programs. I/O PCBs and alternate PCBs are also always passed to MPPs and to IFP application programs.

The PCB list contains the address of the I/O PCB, followed by the addresses of any alternate PCBs, followed by the addresses of the DB PCBs.

### CICS Programs with DBCTL

The first PCB always refers to the first DB PCB whether you specify the SYSSERVE keyword.

Table 8 summarizes the I/O PCB and alternate PCB information. The first column lists different DB environments, the second and third column specify if the I/O PCB or alternate PCB, respectively, is valid in the specified environment.

Table 8. Summary of PCB Information

Environment	EXEC DLI	
	I/O PCB count included in PCB(n)	Alternate PCB count included in PCB(n)
CICS DBCTL <sup>1</sup>	No	No
CICS DBCTL <sup>2</sup>	No	No
BMP	Yes	Yes
Batch <sup>3</sup>	No	Yes
Batch <sup>4</sup>	Yes	Yes

**Notes:**

1. SCHED command issued without the SYSSERVE option.
2. SCHED command issued with the SYSSERVE option for a CICS DBCTL command or for a function-shipped command which is satisfied by a remote CICS system using DBCTL.
3. CMPAT=N specified on the PSBGEN statement.
4. CMPAT=Y specified on the PSBGEN statement.

## Format of a PSB

The following is the format of a PSB.

```
[IOPCB]
[Alternate PCB ... Alternate PCB]
[DBPCB ... DBPCB]
[GSAMPCB ... GSAMPCB]
```

Each PSB must contain at least one PCB. The I/O PCB must be addressable in order to issue a system service command. An alternate PCB is used only for IMS online programs, which can run only with the Transaction Manager. Alternate PCBs can be present even though your program does not run under the Transaction Manager. A DB PCB can be a full-function PCB, a DEDB PCB, or an MSDB PCB.

---

## EXEC DLI Commands

The EXEC DLI commands are the only commands allowed for EXEC DLI. They can be used to read and update DL/I databases with a batch program, a BMP region (running DBCTL or DB/DC), or a CICS program using DBCTL.

The examples in this chapter use the PL/I delimiter. Code the commands in free form: Where keywords, operands, and parameters are shown separated by commas, no blanks can appear immediately before or after the comma. Where keywords, operands, and parameters are shown separated by blanks, you can include as many blanks as you wish. The format of the commands is the same for users of COBOL, PL/I, assembler language, C/370™, and C++/370.

Table 9 on page 34 provides a summary of the commands available to batch, BMP, and online programs. The descriptions in the following topics illustrate the general syntax of the EXEC DLI commands and the information supplied by each parameter and variable:

- “Summary of EXEC DLI Commands” on page 34
- “DLET Command” on page 35
- “GN Command” on page 36
- “GNP Command” on page 41
- “GU Command” on page 47
- “ISRT Command” on page 52
- “POS Command” on page 58
- “REPL Command” on page 59
- “RETRIEVE Command” on page 63
- “SCHD Command” on page 64
- “TERM Command” on page 65
- “System Service Commands” on page 66
- “ACCEPT Command” on page 66
- “CHKP Command” on page 67
- “DEQ Command” on page 68
- “LOAD Command” on page 69
- “LOG Command” on page 70
- “QUERY Command” on page 70
- “REFRESH Command” on page 71
- “ROLB Command” on page 72

## EXEC DLI Commands

- “ROLL Command” on page 73
- “ROLS Command” on page 73
- “SETS Command” on page 75
- “SETU Command” on page 76
- “STAT Command” on page 77
- “SYMCHKP Command” on page 78
- “XRST Command” on page 79

The examples included with each command refer to the “A Database Hierarchy Example” on page 2.

## Summary of EXEC DLI Commands

A summary of all the EXEC DLI commands is provided in Table 9. The table lists the EXEC DLI commands and specifies if they are valid in the Batch, Batch-Oriented BMP, or CICS with DBCTL environment.

Table 9. Summary of EXEC DLI Commands

Request Type	Program Characteristics		
	Batch	Batch- Oriented BMP	CICS with DBCTL <sup>1</sup>
ACCEPT command <sup>4</sup>	Yes	Yes	Yes
CHKP command <sup>4</sup>	Yes	Yes	No
DEQ command <sup>4</sup>	Yes	Yes	Yes
DLET command <sup>4</sup>	Yes	Yes	Yes
Get commands (GU, GHU, GN, GHN, GNP, GHNP) <sup>4</sup>	Yes	Yes	Yes
GMSG command <sup>5</sup>	No	Yes	Yes
ICMD command <sup>5</sup>	No	Yes	Yes
ISRT command <sup>4</sup>	Yes	Yes	Yes
LOAD command	Yes	No	No
LOG command <sup>4</sup>	Yes	Yes	Yes
POS command <sup>4</sup>	No	Yes	Yes
QUERY command <sup>4</sup>	Yes	Yes	Yes
RCMD command <sup>5</sup>	No	Yes	Yes
REFRESH command <sup>4</sup>	Yes	Yes	Yes
REPL command <sup>4</sup>	Yes	Yes	Yes
RETRIEVE command	Yes	Yes	No
ROLB command	Yes	Yes	No
ROLL command	Yes	Yes	No
ROLS command <sup>2,4</sup>	Yes	Yes	Yes
SCHD command	No	No	Yes
SETS command <sup>2,4</sup>	Yes	Yes	Yes
SETU command	Yes	Yes	No
STAT command <sup>3,4</sup>	Yes	Yes	Yes
SYMCHKP command	Yes	Yes	No
TERM command	No	No	Yes

Table 9. Summary of EXEC DLI Commands (continued)

Request Type	Program Characteristics		
	Batch	Batch- Oriented BMP	CICS with DBCTL <sup>1</sup>
XRST command	Yes	Yes	No

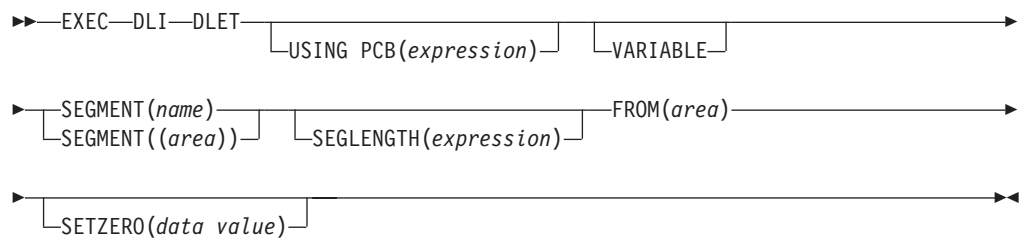
**Notes:**

1. In a CICS remote DL/I environment, commands in the CICS with DBCTL column are supported if you are shipping a function to a remote CICS that uses DBCTL.
2. ROLS and SETS commands are not valid when the PSB contains a DEDB.
3. STAT is a Product-sensitive programming interface.
4. Are supported in the AIB format.
5. Are described in the AOI documentation within the *IMS Version 9: Operations Guide*.

## DLET Command

The Delete (DLET) command is used to remove a segment and its dependents from the database.

### Format



### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### VARIABLE

Indicates that a segment is variable-length.

#### SEGMENT(name)

Qualifies the command, specifying the name of the segment type you want to retrieve, insert, delete, or replace.

#### SEGMENT((area))

Is a reference to an area in your program containing the name of the segment type. You can specify an area instead of specifying the name of the segment in the command.

#### SEGLENGTH(expression)

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

## DLET Command

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

### FROM(area)

Specifies an area containing the segment to be added, replaced, or deleted. Use FROM to insert one or more segments with one command.

### SETZERO(data\_value)

Specifies setting a subset pointer to zero.

## Usage

You use the DLET command to delete a segment and its dependents from the database. You must first retrieve segments you want to delete, just as if you were replacing segments. The DLET command deletes the retrieved segment *and* its dependents, if any, from the database.

## Example

“Evelyn Parker has moved away from this area. Her patient number is 10450. Delete her record from the database.”

**Explanation:** You want to delete all the information about Evelyn Parker from the database. To do this, you must delete the PATIENT segment. When you do this, DL/I deletes all the dependents of that segment. This is exactly what you want DL/I to do—there is no reason to keep such segments as ILLNESS and TREATMNT for Evelyn Parker if she is no longer one of the clinic’s patients.

Before you can delete the patient segment, you have to retrieve it:

```
EXEC DLI GU  
    SEGMENT(PATIENT) INTO(PATAREA) WHERE (PATNO=PATN01);
```

To delete this patient’s database record, you issue a DLET command and use the FROM option to give the name of the I/O area that contains the segment you want deleted:

```
EXEC DLI DLET SEGMENT(PATIENT) FROM(PATAREA);
```

When you issue this command, the PATIENT segment, and its dependents—the ILLNESS, TREATMNT, BILLING, PAYMENT, and HOUSHOLD segments—are deleted.

## Restrictions

You cannot issue any commands using the same PCB between the retrieval command and the DLET command, and you can issue only one DLET command for each GET command.

## GN Command

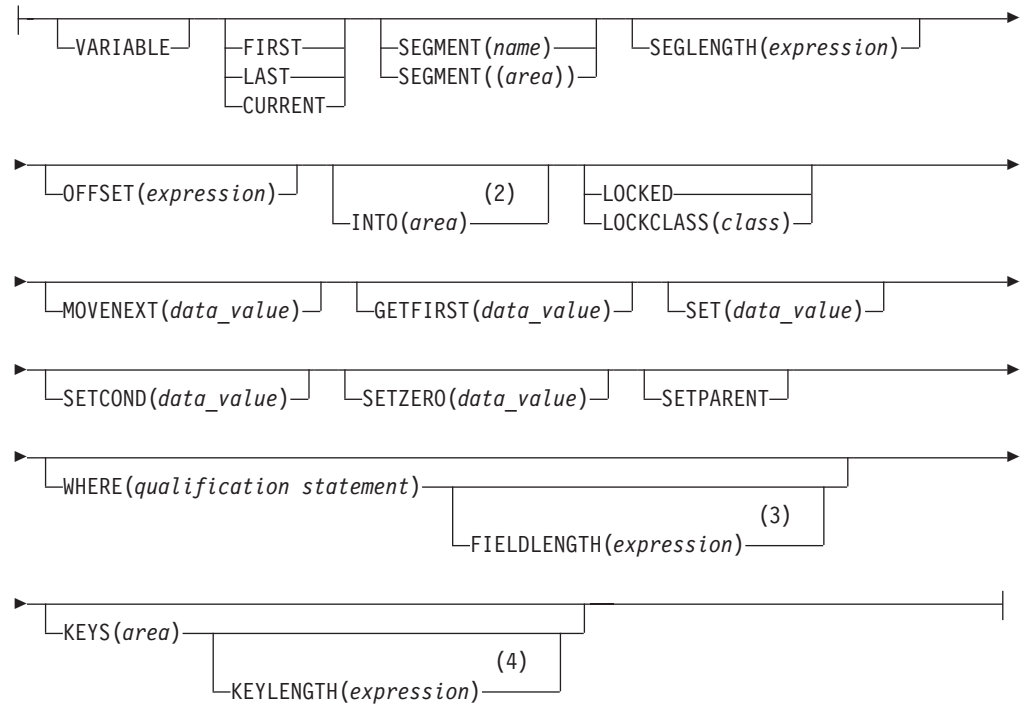
The Get Next (GN) command is used to retrieve segments sequentially.

### Format

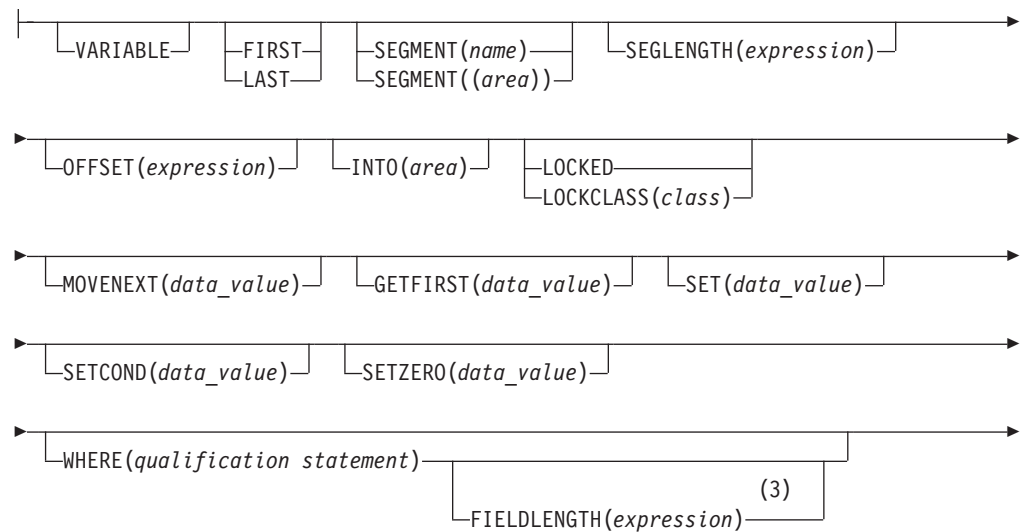
```
▶▶—EXEC—DLI—GET NEXT  
    └──GN──┘ └──USING PCB(expression)──┘
```

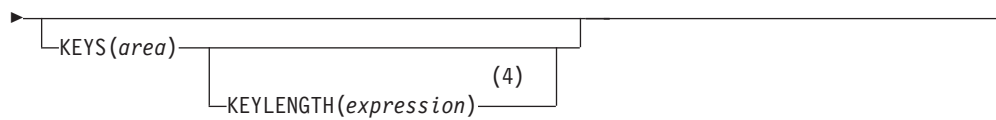


**A For each parent segment (optional):**



**B For the object segment (optional):**





### Notes:

- 1 If you leave out the SEGMENT option, specify the INTO option as shown.
- 2 Specify INTO on parent segments for a path command.
- 3 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 4 You can use either the KEYS option or the WHERE option, but not both on one segment level.

### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### KEYFEEDBACK(area)

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

#### FEEDBACKLEN(expression)

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

#### INTO(area)

Specifies an area into which the segment is read.

#### VARIABLE

Indicates that a segment is variable-length.

#### FIRST

Specifies that you want to retrieve the first segment occurrence of a segment type, or that you want to insert a segment as the first occurrence.

#### LAST

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence.

#### CURRENT

Qualifies the command, and specifies that you want to use the level of and levels above the current position as qualifications for this segment.

#### SEGMENT(name), SEGMENT((area))

Qualifies the command, specifying the name of the segment type or the area of your program containing the name of the segment type that you want to retrieve.

You can have as many levels of qualification for a GN command as there are levels in the database's hierarchy. Using fully qualified commands with the WHERE or KEYS option clearly identifies the hierarchical path and the



segment you want, and is useful in documenting the command. However, you do not need to qualify a GN command, because you can specify a GN command without the SEGMENT option.

Once you have established position in the database record, issuing a GN command without a SEGMENT option retrieves the next segment occurrence in sequential order.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GN command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from the DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Omitting the SEGMENT option for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest-level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

#### **SEGLENGTH(expression)**

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

#### **OFFSET(expression)**

Specifies the offset to the destination parent. It can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. Use OFFSET when you process concatenated segments in logical relationships. OFFSET is required whenever the destination parent is a variable-length segment.

#### **LOCKED**

Specifies that you want to retrieve a segment for the exclusive use of your program, until a checkpoint or sync point is reached. This option performs the same function as the Q command code, and it applies to both Fast Path and full function. A 1-byte alphabetic character of 'A' is automatically appended as the class for the Q command code.

#### **LOCKCLASS(class)**

Specifies that you want to retrieve a segment for the exclusive use of your program until a DEQ command is issued or until a checkpoint or sync point is reached. (DEQ commands are not supported for Fast Path.) Class is a 1-byte alphabetic character (B-J), representing the lock class of the retrieved segment.

For full-function code, the LOCKCLASS option followed by a letter (B-J) designates the class of the lock for the segment. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

## GN Command

Fast Path does not support LOCKCLASS but, for consistency between full function and Fast Path, you must specify LOCKCLASS('x'), where x is a letter in the range B to J. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

### **MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

### **GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

### **SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

### **SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

### **SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

### **SETPARENT**

Sets parentage at the level you want.

### **FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

### **KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. It can be any expression in the host language that converts to the integer data type; if it is a variable, it must be declared as a binary halfword value. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or the VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

### **KEYS(area)**

Qualifies the command with the segment's concatenated key. You can use either KEYS or WHERE for a segment level, but not both.

"Area" specifies an area in your program containing the segment's concatenated key.

### **WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

## **Usage**

Use the GN command to sequentially retrieve segments from the database. Each time you issue a GN command, IMS DB retrieves the next segment, as described by the options you include in the command. Before issuing a GN command, you should establish position in the database record by issuing a GU command.

You do not have to use a segment option with a GN command. However, you should qualify your GN commands as much as possible with the KEYS or WHERE options after the SEGMENT option.

## Examples

**Example 1:** “We need a list of all patients who have been to this clinic.”

*Explanation:* To answer this request, your program would issue a command qualified with the segment name PATIENT until DL/I returned a GB status code to the program. (GB means that DL/I reached the end of the database before being able to satisfy your command). This command looks like this:

```
EXEC DLI GN
  SEGMENT(PATIENT) INTO(PATAREA);
```

Each time your program issued this command, the current position moves forward to the next database record.

**Example 2:** “What are the names of the patients we have seen since the beginning of this month?”

*Explanation:* A GN command that includes one or more WHERE or KEYS options retrieves the next occurrence of the specified segment type that satisfies the command. To answer this request, the program issues the following GN command until DL/I returned a GB status code. The example shows the command you use at the end of April, 1988 (assuming ILLDATE1 contains 198804010):

```
EXEC DLI GN
  SEGMENT(PATIENT) INTO(PATAREA)
  SEGMENT(ILLNESS) INTO(ILLAREA) WHERE(ILLDATE>=ILLDATE1);
```

**Example 3:**

```
EXEC DLI GN INTO(PATAREA);
```

*Explanation:* If you just retrieved the PATIENT segment for patient 04124 and then issued this command, you retrieve the first ILLNESS segment for patient 04124.

## Restrictions

With an unqualified GN command, the retrieved segment type might not be the one expected. Therefore, specify an I/O area large enough to contain the largest segment accessible to your program.

Use either the KEYS option or the WHERE option, but not both on one segment level.

## GNP Command

The Get Next in Parent (GNP) command is used to retrieve dependent segments sequentially.

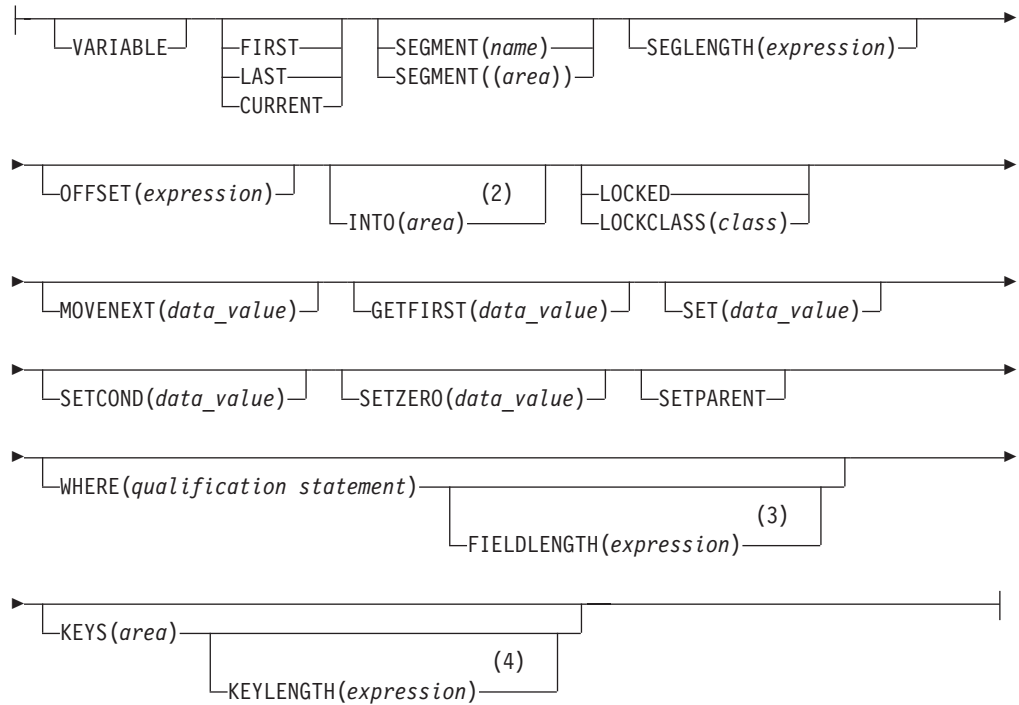
### Format

```
▶▶ EXEC DLI GET NEXT IN PARENT
  GNP USING PCB(expression)
```

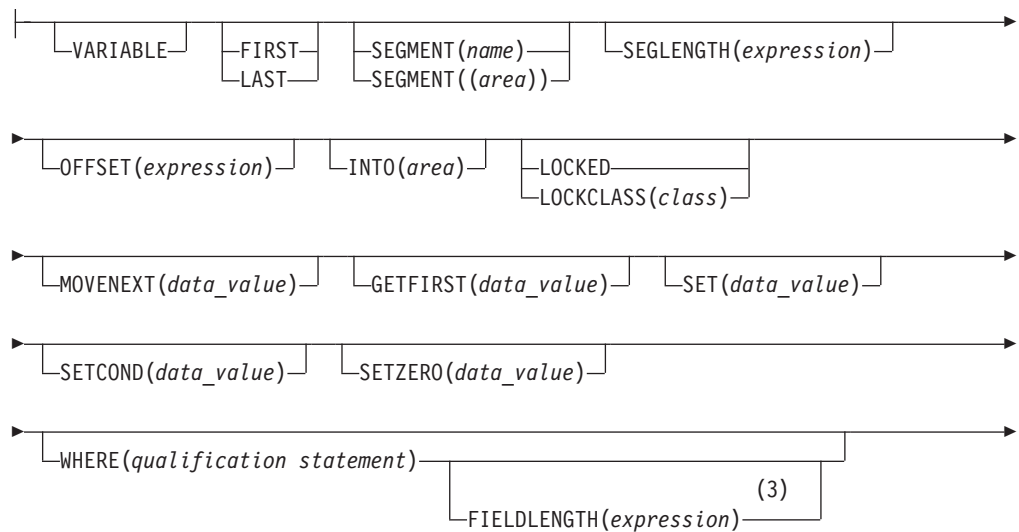
## GNP Command

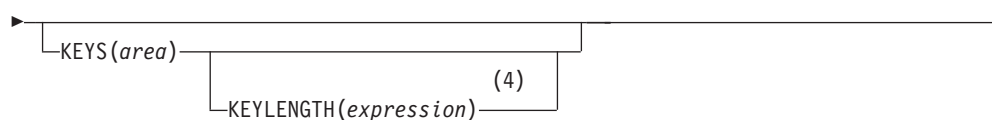


### A For each parent segment (optional):



### B For the object segment (optional):



**Notes:**

- 1 If you leave out the SEGMENT option, specify the INTO option as shown.
- 2 Specify INTO on parent segments for a path command.
- 3 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 4 You can use either the KEYS option or the WHERE option, but not both on one segment level.

**Options**

You can qualify your GNP command by using SEGMENT and WHERE options.

If you do not qualify your command, IMS DB retrieves the next sequential segment under the established parent. If you include a SEGMENT option, IMS DB retrieves the first occurrence of that segment type that it finds by searching forward under the established parent.

You can have as many levels of qualification for a GNP command as there are levels in the database's hierarchy. However, you should not qualify your command in a way that causes DL/I to move off of the segment type you have established as a parent for the command.

**USING PCB(expression)**

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**KEYFEEDBACK(area)**

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated. Use this to retrieve a segment's concatenated key.

**FEEDBACKLEN(expression)**

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**INTO(area)**

Specifies an area into which the segment is read. Use this to retrieve one or more segments with one command.

**VARIABLE**

Indicates that a segment is variable-length.

**FIRST**

Specifies that you want to retrieve the first segment occurrence of a segment type, or that you want to insert a segment as the first occurrence. Use this to retrieve the first segment occurrence of a segment type.

**LAST**

Specifies that you want to retrieve the last segment occurrence of a segment

type, or that you want to insert a segment as the last segment occurrence. Use this to retrieve the last segment occurrence of a segment type.

### **CURRENT**

Qualifies the command, and specifies that you want to use the level of and levels above the current position as qualifications for this segment. Use this to retrieve a segment based on your current position.

### **SEGLENGTH(expression)**

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (SEGLENGTH is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

### **OFFSET(expression)**

Specifies the offset to the destination parent. The argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. Use OFFSET when you process concatenated segments in logical relationships. OFFSET is required whenever the destination parent is a variable-length segment.

### **LOCKED**

Specifies that you want to retrieve a segment for the exclusive use of your program, until a checkpoint or sync point is reached. Use this to reserve a segment for the exclusive use of your program. This option performs the same function as the Q command code, and it applies to both Fast Path and full function. A 1-byte alphabetic character of 'A' is automatically appended as the class for the Q command code.

### **LOCKCLASS(class)**

Specifies that you want to retrieve a segment for the exclusive use of your program until a DEQ command is issued or until a checkpoint or sync point is reached. (DEQ commands are not supported for Fast Path.) Class is a 1-byte alphabetic character (B-J), representing the lock class of the retrieved segment.

For full-function code, the LOCKCLASS option followed by a letter (B-J) designates the class of the lock for the segment. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

Fast Path does not support LOCKCLASS but, for consistency between full function and Fast Path, you must specify LOCKCLASS('x'), where x is a letter in the range B to J. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

### **MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

### **GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

### **SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

**SETPARENT**

Sets parentage at the level you want.

**WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYS(area)**

Qualifies the command with the segment's concatenated key. You can use either KEYS or WHERE for a segment level, but not both.

"Area" specifies an area in your program containing the segment's concatenated key.

**KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. It can be any expression in the host language that converts to the integer data type; if it is a variable, it must be declared as a binary halfword value. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

**SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area in your program containing the name of the segment type that you want to retrieve, insert, delete, or replace.

You can have as many levels of qualification for a GNP command as there are levels in the database's hierarchy. Using fully qualified commands with the WHERE or KEYS option clearly identifies the hierarchic path and the segment you want, and is useful in documenting the command. However, you do not need to qualify a GNP command at all, because you can specify a GNP command without the SEGMENT option.

Once you have established position in the database record, issuing a GNP command without a SEGMENT option retrieves the next segment occurrence in sequential order.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GNP command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from DIB the segment type retrieved.)



If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Omitting the SEGMENT option for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest-level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

### Usage

The Get Next in Parent (GNP) command makes it possible to limit the search for a segment; you can retrieve only the dependents of a particular parent. You must have established parentage before issuing a GNP command.

### Examples

**Example 1:** "We need the complete record for Kate Bailey. Her patient number is 09080."

*Explanation:* To satisfy this request, you want only to retrieve the dependent segments of the patient whose patient number is 09080; you do not want to retrieve all the dependents of each patient. To do this, use the GU command to establish your position and parentage on the PATIENT segment for Kate Bailey. Then continue to issue a GNP without SEGMENT or WHERE options until DL/I returns all the dependents of that PATIENT segment. (A GE status code indicates that you have retrieved all the dependent segments.) To answer this request, your program can issue these commands:

```
EXEC DLI GU
      SEGMENT(PATIENT) INTO(PATAREA)
      WHERE (PATNO=PATN01);
EXEC DLI GNP
      INTO(ILLAREA);
```

A GNP command without SEGMENT or WHERE options retrieves the first dependent segment occurrence under the current parent. If your current position is already on a dependent of the current parent, this command retrieves the next segment occurrence under the parent.

With an unqualified GNP command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a GNP command, you can find out from the DIB the segment type retrieved.)

**Example 2:** "Which doctors have been prescribing acetaminophen for headaches?"

*Explanation:* A GNP command with only a SEGMENT option sequentially retrieves the dependent segments of the segment type you have specified under the established parent. Suppose that for this example, the key of ILLNESS is ILLNAME, and the key of TREATMNT is MEDICINE. You want to retrieve each TREATMNT segment where the treatment was acetaminophen. The name of the doctor who prescribed the treatment is part of the TREATMNT segment. (Assume that data area ILLNAME1 contains HEADACHE, and MEDIC1 contains ACETAMINOP). To answer this request, you can issue these commands:



```
EXEC DLI GN
  SEGMENT(ILLNESS) WHERE (ILLNAME=ILLNAME1);
EXEC DLI GNP
  SEGMENT(TREATMNT) WHERE (MEDICINE=MEDIC1);
```

To process this, your program continues issuing the GNP command until DL/I returned a GE (not found) status code, then your program retrieves the next headache segment and retrieves the TREATMNT segments for it. Your program does this until there were no more ILLNESS segments where the ILLNAME was headache.

## Restrictions

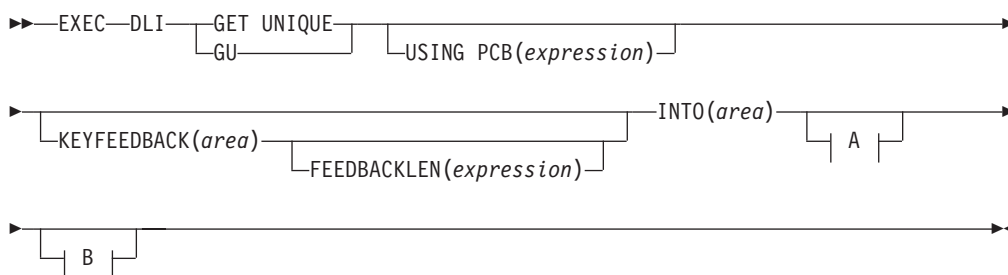
Restrictions for GNP command:

- You must have established parentage before issuing this command.
- You cannot qualify your GNP command in a way that causes DL/I to move off of the segment type you have established as the parent for the command.
- You can retrieve only the dependents of a particular parent.

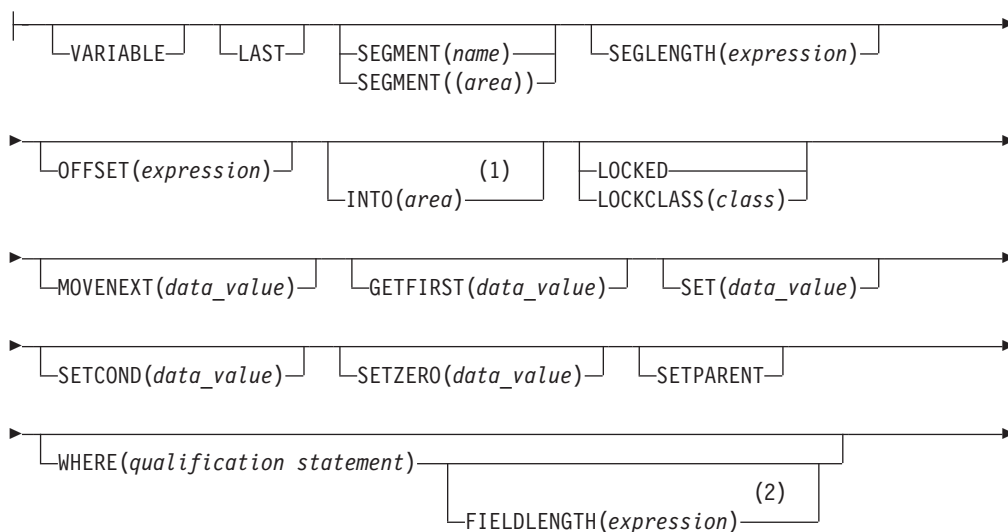
## GU Command

The Get Unique (GU) command is used to directly retrieve specific segments, and to establish a starting position in the database for sequential processing.

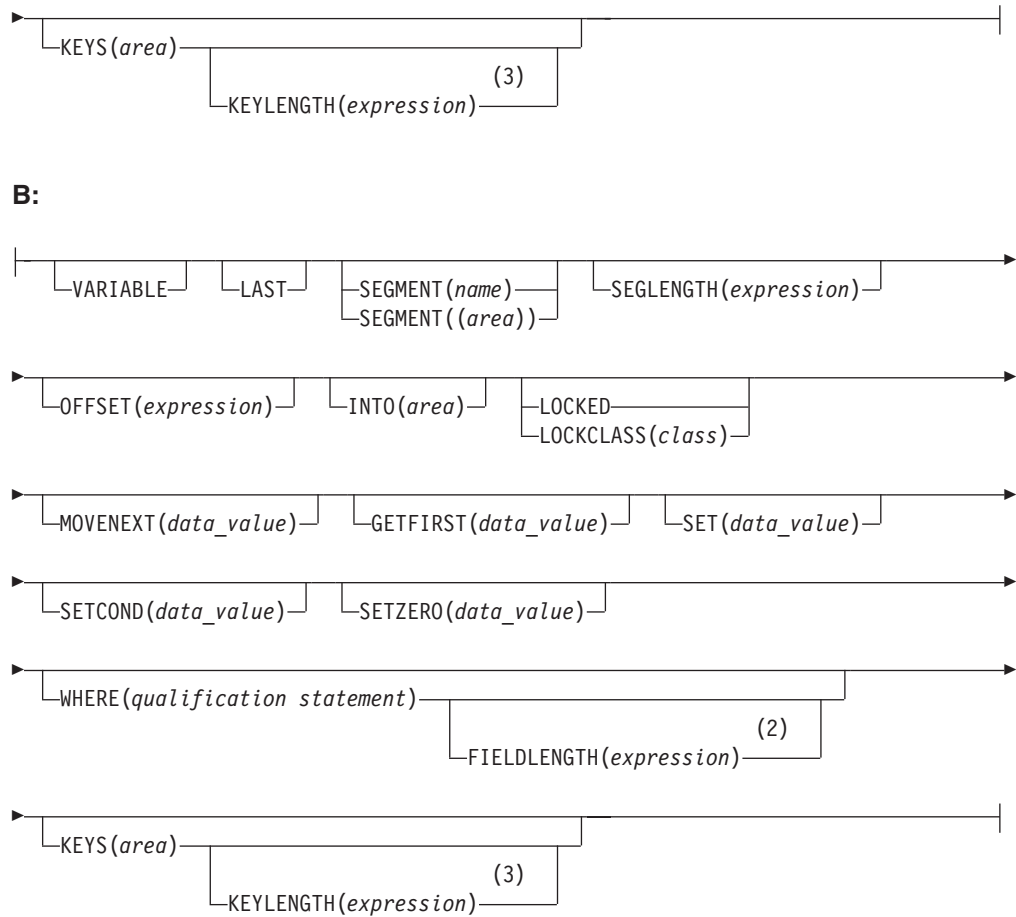
### Format



### A:



## GU Command



### Notes:

- 1 Specify INTO on parent segments for a path command.
- 2 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 3 You can use either the KEYS option or the WHERE option, but not both on one segment level.

### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### KEYFEEDBACK(area)

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

#### FEEDBACKLEN(expression)

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**INTO(area)**

Specifies an area into which the segment is read.

**VARIABLE**

Indicates that a segment is variable-length.

**LAST**

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence.

**SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area in your program containing the name of the segment type that you want to retrieve, insert, delete, or replace.

To retrieve the first occurrence of a segment type, you need only specify the SEGMENT option. You can specify as many levels of qualification as there are hierarchic levels defined by the PCB you are using.

To establish position at the beginning of the database, issue a GU command with a SEGMENT option that names the root segment type.

If you leave out SEGMENT options for one or more hierarchic levels, DL/I assumes a segment qualification for that level. The qualification that DL/I assumes depends on your current position.

- If DL/I has a position established at the missing level, DL/I uses the segment on which position is established.
- If DL/I does not have a position established at the missing level, DL/I uses the first occurrence at that level.
- If DL/I moves forward from a position established at a higher level, DL/I uses the first occurrence at the missing level that falls within the new path.
- If you leave out a SEGMENT option for the *root* level, and DL/I has position established on a root, DL/I does not move from that root when trying to satisfy the command.

You can have as many levels of qualification for a GU command as there are levels in the database's hierarchy. Using fully qualified commands with the WHERE or KEYS option clearly identifies the hierarchic path and the segment you want, and is useful in documenting the command. However, you do not need to qualify a GU command at all, because you can specify a GU command without the SEGMENT option.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GU command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Omitting the SEGMENT option for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest-level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

### **SEGLENGTH(expression)**

Specifies the length of the I/O area into which the segment is retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (SEGLENGTH is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

### **OFFSET(expression)**

Specifies the offset to the destination parent. The argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. Use OFFSET when you process concatenated segments in logical relationships. OFFSET is required whenever the destination parent is a variable-length segment.

### **LOCKED**

Specifies that you want to retrieve a segment for the exclusive use of your program, until a checkpoint or sync point is reached. This option performs the same function as the Q command code. It applies to both Fast Path and full function. A 1-byte alphabetic character of 'A' is automatically appended as the class for the Q command code.

### **LOCKCLASS(class)**

Specifies that you want to retrieve a segment for the exclusive use of your program until a DEQ command is issued or until a checkpoint or sync point is reached. (DEQ commands are not supported for Fast Path.) Class is a 1-byte alphabetic character (B-J), representing the lock class of the retrieved segment.

For full-function code, the LOCKCLASS option followed by a letter (B-J) designates the class of the lock for the segment. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

Fast Path does not support LOCKCLASS but, for consistency between full function and Fast Path, you must specify LOCKCLASS('x'), where x is a letter in the range B to J. An example is LOCKCLASS('B'). If LOCKCLASS is not followed by a letter in the range B to J, then EXECDLI sets a status code of GL and initiates an ABENDU1041.

### **MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

### **GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

### **SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

### **SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

### **SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

### **SETPARENT**

Sets parentage at the level you want.

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYLENGTH(expression)**

Specifies the length of the concatenated key when you use the KEYS option. The argument can be any expression in the host language that converts to the integer data type; a variable must be declared as a binary halfword value. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

**WHERE(qualification statement)**

Use WHERE to further qualify your GU commands after using SEGMENT. If you fully qualify a GU command, you can retrieve a segment regardless of your position in the database record.

**KEYS(area)**

Use KEYS to further qualify your GU commands and specify the segment occurrence by using its concatenated key.

If you specify a SEGMENT option without a KEYS or WHERE option, IMS DB retrieves the first occurrence of that segment type it encounters by searching forward from current position. With an unqualified GU command, the segment type you retrieve might not be the one you expected, so you should specify an I/O area large enough to contain the largest segment your program has access to. (After successfully issuing a retrieval command, you can find out from DIB the segment type retrieved.)

If you fully qualify your command with a WHERE or KEYS option, you would retrieve the next segment in sequential order, as described by the options.

Including the WHERE or KEYS options for parent segments defines the segment occurrences that are to be part of the path to the segment you want retrieved. Leaving the SEGMENT option out for a level, or including only the SEGMENT option without a WHERE option, indicates that any path to the option satisfies the command. DL/I uses only the qualified parent segments and the lowest level SEGMENT option to satisfy the command. DL/I does not assume a qualification for the missing level.

**Usage**

Use the GU command to retrieve specific segments from the database, or to establish a position in the database for sequential processing.

You must at least specify the SEGMENT option with a GU command to indicate the segment type you want to retrieve. (IMS DB retrieves the first occurrence of the segment you named in the SEGMENT argument.)

When you need to retrieve a specific occurrence of a segment type, you can further qualify the command by using the WHERE or KEYS option after the SEGMENT option.

You probably want to further qualify your GU commands with the WHERE or KEYS option, and specify a specific occurrence of a segment type. If you fully qualify a GU command, you can retrieve a segment regardless of your position in the database record.

### Examples

**Example 1:** “What illness was Robert James here for most recently? Was he given any medication on that day for that illness? His patient number is 05136.”

*Explanation:* This example requests two pieces of information. To answer the first part of the request and retrieve the most recent ILLNESS segment, issue this GU command (assuming that PATNO1 contains 05163):

```
EXEC DLI GU
  SEGMENT(PATIENT) WHERE(PATNO=PATNO1)
  SEGMENT(ILLNESS) INTO(AREA);
```

Once you had retrieved the ILLNESS segment with the date of the patient’s most recent visit to the clinic, you can issue another command to find out whether he was treated during that visit. If the date of his most recent visit was January 5, 1988, you can issue the following command to find out whether or not he was treated on that day for that illness (assuming PATNO1 contains 05163, and DATE1 contains 19880105):

```
EXEC DLI GU
  SEGMENT(PATIENT) WHERE(PATNO=PATNO1)
  SEGMENT(ILLNESS) WHERE(ILLDATE=DATE1)
  SEGMENT(TREATMNT) INTO(TRTAREA) WHERE(DATE=DATE1);
```

**Example 2:** “What is Joan Carter currently being treated for? Her patient number is 10320.”

```
EXEC DLI GU
  SEGMENT(PATIENT) WHERE(PATNO=PATNO1)
  SEGMENT(ILLNESS) INTO(ILLAREA);
```

*Explanation:* In this example you want the ILLNESS segment for the patient whose patient number is 10320.

### Example 3:

```
EXEC DLI GU
  SEGMENT(PATIENT)
  SEGMENT(ILLNESS)
  SEGMENT(TREATMNT) INTO(AREA);
```

*Explanation:* This example retrieves the first TREATMNT segment and specifies the three levels of qualification.

### Restriction

You must at least specify the SEGMENT option to indicate the segment type you want to retrieve.

## ISRT Command

The Insert (ISRT) command is used to add one or more segments to the database.

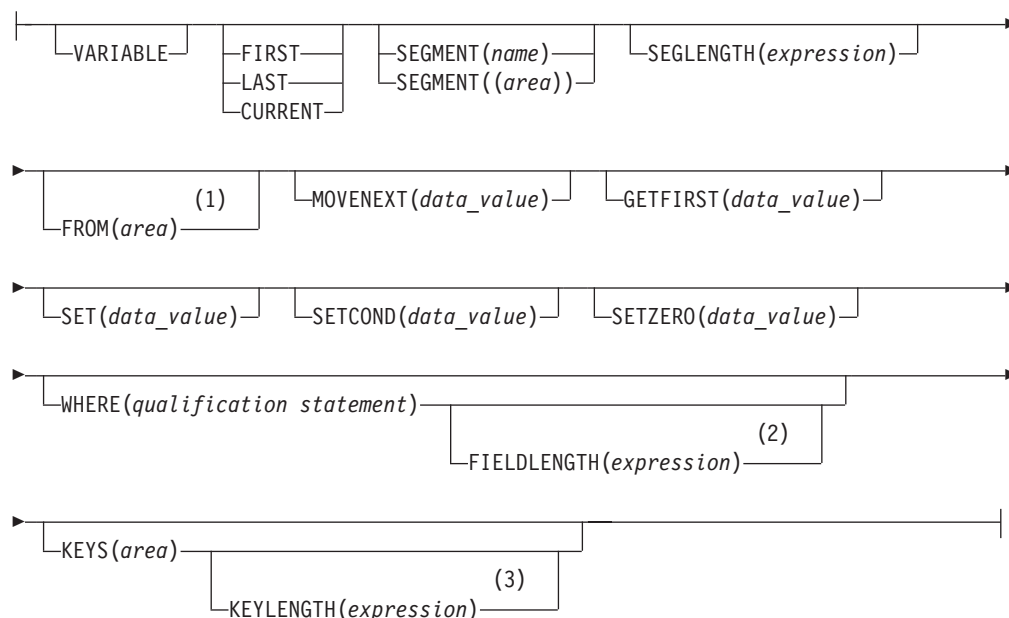
### Format

```
▶▶ EXEC—DLI—

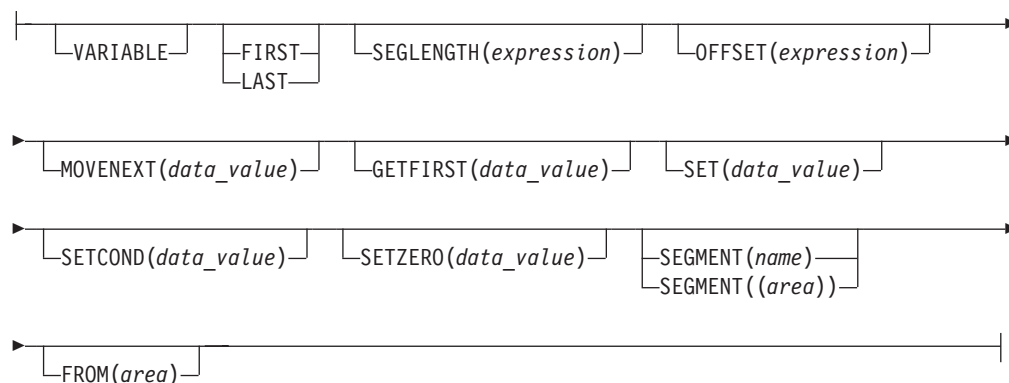
|        |                       |   |   |
|--------|-----------------------|---|---|
| INSERT |                       |   | B |
| ISRT   | USING PCB(expression) | A |   |

▶▶
```

**A For each parent segment (optional):**



### B For the object segment (required):



### Notes:

- 1 Specify FROM on parent segments for a path command.
- 2 If you use multiple qualification statements, specify a length for each, using FIELDLENGTH. For example: **FIELDLENGTH(24,8)**
- 3 You can use either the Keys option or the Where option, but not both on one segment level.

### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### VARIABLE

Indicates that a segment is variable-length.

#### FIRST

Specifies that you want to retrieve the first segment occurrence of a segment

type, or that you want to insert a segment as the first occurrence. Use `FIRST` to insert a segment as a first occurrence of a segment type.

### **LAST**

Specifies that you want to retrieve the last segment occurrence of a segment type, or that you want to insert a segment as the last segment occurrence. Use `LAST` to insert a segment as the last occurrence of a segment type.

### **CURRENT**

Qualifies the command, and specifies that you want to use the level of and levels above the current position as qualifications for this segment. Use `CURRENT` to insert a segment based on your current position.

### **SEGMENT(name), SEGMENT((area))**

Qualifies the command, specifying the name of the segment type or the area in the program containing the name of the segment type that you want to retrieve, insert, delete, or replace.

You must include at least a `SEGMENT` option for each segment you want to add to the database. Unless `ISRT` is a path command, the lowest level `SEGMENT` option specifies the segment being inserted. You cannot use a `WHERE` or `KEYS` option for this level.

If a segment has a unique key, `DL/I` inserts the segment in its key sequence. (If the segment does not have a key, or has a nonunique key, `DL/I` inserts it according to the value specified for the `RULES` parameter during `DBDGEN`.)

If you specify a `SEGMENT` option for only the lowest level segment, and do not qualify the parent segments with `SEGMENT`, `WHERE`, or `KEYS` options, you must make sure that the current position is at the correct place in the database to insert the segment. The `SEGMENT` option that `DL/I` assumes depends on your current position in the database record:

- If `DL/I` has a position established at the missing level, `DL/I` uses the segment on which position is established.
- If `DL/I` does not have a position established at the missing level, `DL/I` uses the first occurrence at that level.
- If `DL/I` moves forward from a position established at a higher level, `DL/I` uses the first occurrence at the missing level that falls within the new path.
- If you leave out a `SEGMENT` option for the *root* level, and `DL/I` has position established on a root, `DL/I` does not move from that root when trying to satisfy the command.

It is good practice to always provide qualifications for higher levels to establish the position of the segment being inserted.

If you are inserting a root segment, you need only specify a `SEGMENT` option. `DL/I` determines the correct place for its insertion in the database by the key taken from the I/O area. If the segment you are inserting is not a root segment, but you have just inserted its immediate parent, the segment can be inserted as soon as it is built in the I/O area just by using a `SEGMENT` option for it in the `ISRT` command. You need not code the parent level segments to establish your position.

When you specify multiple parent segments, you can mix segments with and without the `WHERE` option. If you include only `SEGMENT` options on parent segments, `DL/I` uses the first occurrence of each segment type to satisfy the command.

### **SEGLENGTH(expression)**

Specifies the length of the I/O area from which the segment is obtained. Its



argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

**FROM(area)**

Specifies an area containing the segment to be added, replaced, or deleted. Use FROM to insert one or more segments with one command.

**MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

**GETFIRST(data\_value)**

Specifies that you want the search to start with the first segment occurrence in a subset.

**SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

**SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

**SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

**WHERE(qualification statement)**

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

WHERE establishes position on the parents of a segment when you are inserting that segment. You can do this by specifying a qualification of WHERE or KEYS for the higher level SEGMENT options.

When you specify multiple parent segments, you can mix segments with and without the WHERE option. If you include only SEGMENT options on parent segments, DL/I uses the first occurrence of each segment type to satisfy the command.

**FIELDLENGTH(expression)**

Specifies the length of the field value in a WHERE option.

**KEYS(area)**

Qualifies the command with the segment's concatenated key. You can use either KEYS or WHERE for a segment level, but not both.

KEYS can be used to qualify a parent segment. Instead of using WHERE, you can specify KEYS and use the concatenated key of the segment as qualification. You can use the KEYS option once for each command, immediately after the highest level SEGMENT option.

"Area" specifies an area in your program containing the segment's concatenated key.

### KEYLENGTH(expression)

Specifies the length of the concatenated key when you use the KEYS option. It can be any expression in the host language that converts to the integer data type; if it is a variable, it must be declared as a binary halfword value. For IBM COBOL for MVS & VM (or VS COBOL II), PL/I, or assembler language, KEYLENGTH is optional. For COBOL programs that are not compiled with the IBM COBOL for MVS & VM (or VS COBOL II) compiler, you must specify KEYLENGTH with the KEYS option.

### Usage

To add new segments to an existing database, use the ISRT command. When you issue the ISRT command, DL/I takes the data from the I/O area you have named in the FROM option and adds the segment to the database. (The initial loading of a database requires using the LOAD command, instead of the ISRT command.)

You can use ISRT to add new occurrences of an existing segment type to a HIDAM, HISAM, or HDAM database. For an HSAM database, you can add new segments only by reprocessing the whole database or by adding the new segments to the end of the database.

Before you can issue the ISRT command to add a segment to the database, your program must build the segment to be inserted in an I/O area. If the segment has a key, you must place the correct key in the correct location in the I/O area. If field sensitivity is used, the fields must be in the order defined by the PSB for the application's view of the segment.

If you are adding a root segment occurrence, DL/I places it in the correct sequence in the database by using the key you supply in the I/O area. If the segment you are inserting is not a root, but you have just inserted its parent, you can insert the child segment by issuing an insert request qualified with only the segment name. You must build the new segment in your I/O area before you issue the ISRT request. You also qualify insert requests with the segment name when you add a new root segment occurrence. When you are adding new segment occurrences to an existing database, the segment *type* must have been defined in the DBD. You can add new segment occurrences directly or sequentially after you have built them in the program's I/O area.

If the segment type you are inserting has a unique key field, the location where DL/I adds the new segment occurrence depends on the value of its key field. If the segment does not have a key field, or if the key is not unique, you can control where the new segment occurrence is added by specifying either the FIRST, LAST, or HERE insert rule. Specify the rules on the RULES parameter of the SEGM statement for the database.

### Examples

**Example 1:** "Add information to the record for Chris Edwards about his visit to the clinic on February 1, 1993. His patient number is 02345. He had a sore throat."

*Explanation:* First, build the ILLNESS segment in your program's I/O area. Your I/O area for the ILLNESS segment looks like this:

```
19930201S0RETHROAT
```

Use the command to add this new segment occurrence to the database is:

```
EXEC DLI ISRT
  SEGMENT(PATIENT) WHERE (PATNO=PATN01)
  SEGMENT(ILLNESS) FROM(ILLAREA);
```

**Example 2:** “Add information about the treatment to the record for Chris Edwards, and add information about the illness.”

*Explanation:* You build the TREATMNT segment in a segment I/O area. The TREATMNT segment includes the date, the medication, amount of medication, and the doctor’s name:

```
19930201MYOCINbbb0001TRIEBbbbb&b
```

The following command adds both the ILLNESS segment and the TREATMNT segment to the database:

```
EXEC DLI ISRT
  SEGMENT(PATIENT) WHERE (PATNO=PATN01)
  SEGMENT(ILLNESS) FROM(ILLAREA)
  SEGMENT(TREATMNT) FROM(TRETAREA);
```

**Example 3:**

```
EXEC DLI ISRT
  SEGMENT(ILLNESS) KEYS(CONKEY)
  SEGMENT(TREATMNT) FROM(TRETAREA);
```

*Explanation:* Using this command is the same as having a WHERE option qualified on the key field for the ILLNESS and PATIENT segments.

## Restrictions

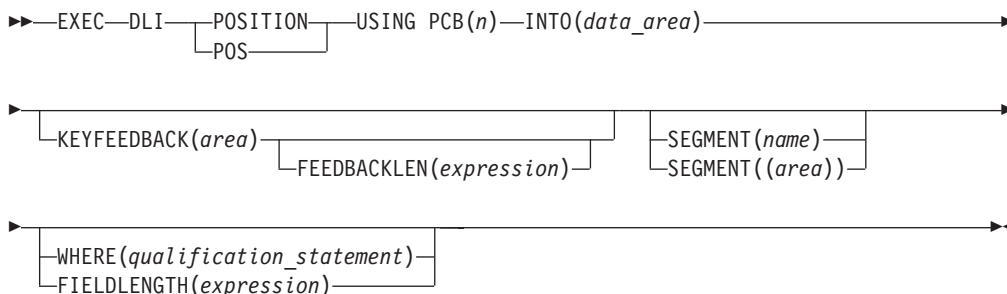
Restrictions the ISRT command:

- You cannot issue the ISRT command until you have built a new segment in the I/O area.
- You must specify at least one SEGMENT option for each segment being added to the database.
- When inserting a segment, you must have position established on the parents of the segment.
- If you specify a SEGMENT option for only the lowest level segment, and do not qualify the parent segments with SEGMENT, WHERE, or KEYS options, be sure that current position is at the correct place in the database to insert the segment.
- If you use a FROM option for a segment, you cannot qualify the segment by using the WHERE or KEYS option; DL/I uses the key field value specified in the I/O area as qualification.
- You must use a separate I/O area for each segment type you want to add.
- You cannot mix SEGMENT options with and without the FROM option. When you use a FROM option for a parent segment, you must use a FROM option for each dependent segment. (You can begin the path at any level, but you must not leave out any levels.)
- You can only use the FIRST option with segments that have either no keys or have a nonunique key with HERE specified on the RULES operand of the SEGM statement in the DBD.
- You can only use the LAST option when the segment has no key or a nonunique key, and the INSERT rule for the segment is either FIRST or HERE.

## POS Command

The Position (POS) command retrieves the location of either a dependent or the segment.

### Format



### Options

#### USING PCB(n)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### INTO(data\_area)

Specifies an area into which the segment is read.

#### KEYFEEDBACK(area)

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

#### FEEDBACKLEN(expression)

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (FEEDBACKLEN is required in COBOL programs and optional in PL/I and assembler language programs.)

#### SEGMENT(name)

Qualifies the command, specifying the name of the segment type you want to retrieve, insert, delete, or replace.

#### SEGMENT((area))

Is a reference to an area in your program containing the name of the segment type. You can specify an area instead of specifying the name of the segment in the command.

#### WHERE(qualification statement)

Qualifies the command, specifying the segment occurrence. Its argument is one or more qualification statements, each of which compares the value of a field in a segment to a value you supply.

#### FIELDLENGTH(expression)

Specifies the length of the field value in a WHERE option.

### Usage

Use the POS command to:

- Retrieve the location of a specific sequential dependent segment, including the last one inserted

- Determine the amount of unused space within each DEDB area

If the area specified by the POS command is unavailable, the I/O area is unchanged and an FH status code is returned.

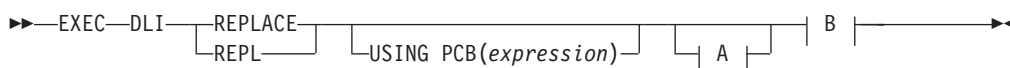
### Restriction

The POS command is for DEDBs only.

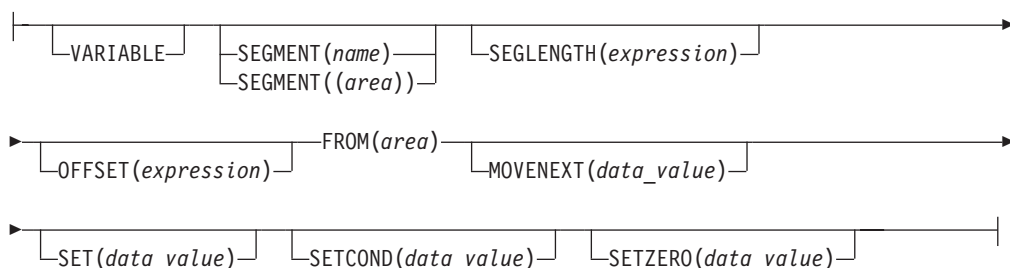
## REPL Command

The Replace (REPL) command is used to replace a segment, usually to change the values of one or more of its fields.

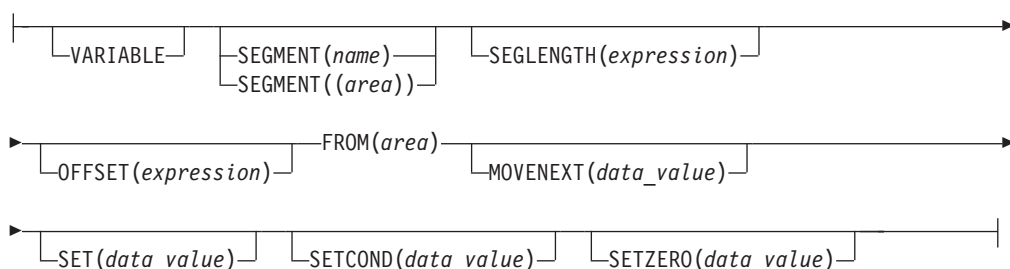
### Format



#### A For each parent segment (optional):



#### B For the object segment (required):



### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### VARIABLE

Indicates that a segment is variable-length.

#### SEGMENT(name)

Qualifies the command, specifying the name of the segment type you want to retrieve, insert, delete, or replace.

## REPL Command

### **SEGMENT((area))**

Is a reference to an area in your program containing the name of the segment type. You can specify an area instead of specifying the name of the segment in the command.

### **SEGLENGTH(expression)**

Specifies the length of the I/O area from which the segment is obtained. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLENGTH must be greater than or equal to the length of the longest segment that can be processed by this call.

### **OFFSET(expression)**

Specifies the offset to the destination parent. It can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. You use OFFSET when you process concatenated segments in logical relationships. It is required whenever the destination parent is a variable length segment.

### **FROM(area)**

Specifies an I/O area containing the segment to be added, replaced or deleted. You can replace more than the segment by including the FROM option after the corresponding SEGMENT option for each segment you want to replace. Including FROM options for one or more parent segments is called a path command.

The argument following FROM identifies an I/O area that you have defined in your program. You must use a separate I/O area for each segment type you want to replace.

### **MOVENEXT(data\_value)**

Specifies a subset pointer to be moved to the next segment occurrence after your current segment.

### **SET(data\_value)**

Specifies unconditionally setting a subset pointer to the current segment.

### **SETCOND(data\_value)**

Specifies conditionally setting a subset pointer to the current segment.

### **SETZERO(data\_value)**

Specifies setting a subset pointer to zero.

## **Usage**

You must qualify the REPL command with at least one SEGMENT and FROM option, which together indicate the retrieved segments you want replaced.

If the Get command that preceded the REPL command was a path command, and you do not want to replace all of the retrieved segments or the PSB does not have replace sensitivity for all of the retrieved segments, you can indicate which of the segments are not to be replaced by omitting the SEGMENT option.

If your program attempts to do a path replace of a segment where it does not have replace sensitivity, the data for the segment in the I/O area for the REPL command must be the same as the segment returned on the preceding GET command. If the data changes in this situation, the transaction is abended and no data is changed as a result of the Replace command.

Notice that the rules for a REPL path command differ from the rules for an ISRT path command. You cannot skip segment levels to be inserted with an ISRT command, as you can with a REPL command.

To update information in a segment, you can use the REPL command. The REPL command replaces data in a segment with data you supply in your application program. First, you must retrieve the segment into an I/O area. You then modify the information in the I/O area and replace the segment with the REPL command. For your program to successfully replace a segment, that segment must already have been defined as replace-sensitive in the PCB by specifying PROCOPT=A or PROCOPT=R on the SENSEG statement in the PCB.

You cannot issue any commands using the same PCB between a Get command and the REPL command, and you can issue only one REPL command for each Get command.

## Examples

### Example 1:

```
EXEC DLI GU SEGMENT(PATIENT) INTO(PATAREA);
EXEC DLI REPL SEGMENT(PATIENT) FROM(PATAREA);
```

*Explanation:* This example shows that you cannot issue any commands using the same PCB between the Get command and the REPL command, and you can issue only one REPL command for each Get command. If you issue this commands and wanted to modify information in the segment again, you must first reissue the GU command, before reissuing the REPL command.

**Example 2:** "We have received a payment for \$65.00 from a patient whose ID is 08642. Update the patient's billing record and payment record with this information, and print a current bill for the patient."

*Explanation:* The four parts to satisfying this processing request are:

1. Retrieve the BILLING and PAYMENT segments for the patient.
2. Calculate the new values for these segments by subtracting \$65.00 from the value in the BILLING segment, and adding \$65.00 to the value in the PAYMENT segment.
3. Replace the values in the BILLING and PAYMENT segments with the new values.
4. Print a bill for the patient, showing the patient's name, number, address, the current amount of the bill, and the amount of the payments to date.

To retrieve the BILLING and PAYMENT segments, issue a GU command. Because you also need the PATIENT segment when you print the bill, you can include INTO following the SEGMENT options for the PATIENT segment and for the BILLING segment:

```
EXEC DLI GU
    SEGMENT(PATIENT) INTO(PATAREA) WHERE (PATNO=PATNO1)
    SEGMENT(BILLING) INTO(BILLAREA)
    SEGMENT(PAYMENT) INTO(PAYAREA);
```

After you have calculated the current bill and payment, you can print the bill, then replace the billing and payment segments in the database. Before issuing the REPL command, you must change the segments in the I/O area.



## REPL Command

Because you have not changed the PATIENT segment, you do not need to replace it when you replace the BILLING and PAYMENT segments. To indicate to DL/I that you do not want to replace the PATIENT segment, you do not specify the SEGMENT option for the PATIENT segment in the REPL command.

```
EXEC DLI REPL
      SEGMENT(BILLING) FROM(BILLAREA)
      SEGMENT(PAYMENT) FROM(PAYAREA);
```

This command tells DL/I to replace the BILLING and PAYMENT segments, but not to replace the PATIENT segment.

These two examples are called *path commands*. You use a path REPL command to replace more than one segment with one command.

**Example 3:** "Steve Arons, patient number 10250, has moved to a new address in this town. His new address is 4638 Brooks Drive, Lakeside, California. Update the database with his new address."

*Explanation:* You need to retrieve the PATIENT segment for Steve Arons and replace the address portion of the segment. To retrieve the PATIENT segment, you can use this GU command (assuming PATNO1 contains 10250):

```
EXEC DLI GU
      SEGMENT(PATIENT) INTO(PATAREA) WHERE (PATNO=PATNO1);
```

Since you are not replacing the first two fields of the PATIENT segment (PATNO and NAME), you do not have to change them in the I/O area. Place the new address in the I/O area following the PATNO and NAME fields. Then you issue the REPL command:

```
EXEC DLI REPL
      SEGMENT(PATIENT) FROM(PATAREA);
```

### Example 4:

```
EXEC DLI GU SEGMENT(PATIENT) INTO(PATAREA)
      WHERE (PATNO=PATNO1)
      SEGMENT(ILLNESS) INTO(ILLAREA)
      SEGMENT(TREATMNT) INTO(TRETAREA);
EXEC DLI REPL SEGMENT(PATIENT) FROM(PATAREA)
      SEGMENT(TREATMNT) FROM(TRETAREA);
```

*Explanation:* This example assumes that you want to replace the PATIENT and TREATMNT segments for patient number 10401, but you do not want to change the ILLNESS segment. To do this issue this command (assuming PATNO1 contains 10401).

## Restrictions

Restrictions for the REPL command:

- You cannot issue any commands using the same PCB between the Get command and the REPL command.
- You can issue only one REPL command for each Get command.
- To modify information in a segment, you must first reissue the GU command before reissuing the REPL command.
- You must qualify the REPL command with at least one SEGMENT option and one FROM option.
- If you use a FROM option for a segment, you cannot qualify the segment by using the WHERE or KEYS option; DL/I uses the key field value specified in the I/O area as qualification.



## RETRIEVE Command

Use the RETRIEVE command to determine current position in the database in batch and BMP programs.

### Format

```
▶▶—EXEC—DLI—RETRIEVE—USING PCB(expression)—KEYFEEDBACK(area)—————▶▶
▶— FEEDBACKLEN(expression)—————▶▶
```

### Options

#### USING PCB(*expression*)

Specifies the DB PCB you want to use for the command. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

**expression** specifies the PCB for which you want to retrieve the concatenated key. It can be any expression in the host language that converts to the integer data type. You can specify either a number or a reference to a halfword containing a number. The value must be a positive integer not greater than the number of PCBs generated for the PSB. The first PCB in the list, the I/O PCB, is 1. The first DB PCB in the list is 2, the second is 3, and so forth.

#### KEYFEEDBACK(*area*)

Specifies an area into which the concatenated key for the segment is placed. If the area is not long enough, the key is truncated.

#### FEEDBACKLEN(*expression*)

Specifies the length of the key feedback area into which you want the concatenated key retrieved. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number. (It is required in COBOL programs and optional in PL/I and assembler language programs.)

**expression** is the length of the key feedback I/O area. It can be any expression in the host language that converts to integer data type; you can specify either a number or a reference to a halfword containing a number. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language, FEEDBACKLEN is optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, you must specify FEEDBACKLEN with the KEYFEEDBACK option.

### Usage

If your program issues symbolic checkpoint commands it must also issue the extended RESTART (XRST) command or the RETRIEVE command. The RETRIEVE command is issued once, at the start of your program. You can use the RETRIEVE command to start your program normally, or to restart it in case of an abnormal termination.

You can use the RETRIEVE command from a specific checkpoint id or a time/date stamp. Because the RETRIEVE command attempts to reposition the database, your program also needs to check for correct position.

After issuing the RETRIEVE command, the segment type and level on which the position is established is returned to the DIBSEGM and DIBSEGLV fields in the

## RETRIEVE Command

DIB. The value in DIBKFBL is set to the actual length of the concatenated key. The DIBSTAT field contains the value returned from the GU repositioning, not the XRST command.

The RESTART command attempts to reposition DL/I databases by issuing an internal GU qualified with the concatenated key. It is your responsibility to verify that your position in the database from the GU repositioning is the correct position for the checkpoint ID used in the XRST command. You can use the RETRIEVE command to retrieve the concatenated key used with the GU repositioning, and determine your current position in all the PCBs your program accesses.

### Example

```
EXEC DLI RETRIEVE USING PCB(2) KEYFEEDBACK(KEYAREA);  
EXEC DLI RETRIEVE USING PCB(5) KEYFEEDBACK(KEYAREA);
```

**Explanation:** These RETRIEVE commands retrieve the concatenated key for the first and fourth DB PCBs. (The first PCB in the list is the I/O PCB, so the first DB PCB is the second one in the list.) After issuing the first RETRIEVE command, you can determine your position in the first DB PCB by examining the concatenated key in KEYAREA, and the values returned in the DIBSEGM and DIBSEGLV fields in the DIB. After issuing the second RETRIEVE command, you can determine your position in the fourth DB PCB by examining the same fields.

### Restrictions

Restrictions for the RETRIEVE command:

- You cannot use this command in a CICS program.
- To use this command, you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command unless the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## SCHD Command

The Schedule (SCHD) command is used to schedule a PSB in a CICS online program. For information on the I/O PCB, see "PCBs and PSB" on page 31.

### Format

```
▶▶ EXEC DLI SCHEDULE PSB(name) SYSSERVE NODHABEND  
          └──┬──┘   └──┬──┘   └──┬──┘   └──┬──┘  
          SCHD   PSB((area))
```

### Options

#### PSB(name)

Specifies the name of the PSB available to your application program that you want to schedule with the SCHD command.

#### PSB((area))

Specifies an 8-byte data area in your program that contains the name of the PSB available to your program that you want to schedule with the SCHD command.

#### SYSSERVE

Specifies that the application program can handle an I/O PCB and might issue a system service request in the logical unit of work (LUW).

**NODHABEND**

Specifies that a CICS transaction does not fail with a DHxx abend.

Should a schedule fail under EXEC DLI, a status code might be returned in the DIB, causing a CICS transaction to fail with a DHxx abend. This option prevents this. Following an unsuccessful SCHD command, the control, as well as the status code in the DIB are passed back to the application program, which can then take the appropriate action.

**Usage**

Before you can access DL/I databases from a CICS program, you must notify DL/I that your program will be accessing a database by scheduling a PSB. Do this by issuing the SCHD command. When you no longer plan to use a PSB, or you want to schedule a subsequent PSB (one or more), you must terminate the previous PSB with the TERM command. (For more information on the I/O PCB and PSB, see "PCBs and PSB" on page 31)

The SCHD command can be specified two ways (see "Example").

**Example**

```
EXEC DLI SCHD PSB(psbname)SYSSERVE;
EXEC DLI SCHD PSB((AREA));
```

**Explanation:** These examples show two ways to schedule a PSB in a CICS program.

**TERM Command**

The Terminate (TERM) command is used to terminate a PSB in a CICS online program.

**Format**



**Options**

No options are allowed with the TERM command.

**Usage**

If you want to use a PSB other than the one already scheduled, use the TERM command to release the PSB.

When you issue the TERM command, all database changes are committed and cannot be backed out. Because returning to CICS also terminates the PSB and commits changes, you need not use the TERM command unless you want to schedule another PSB, or commit database changes before returning to CICS.

No options are allowed with the TERM command. If your program subsequently needs a PSB that has terminated, it must reschedule that PSB by issuing another SCHD command.

In most applications, you do not need to use the TERM command.

**Example**

```
EXEC DLI TERM
```

## TERM Command

**Explanation:** This example shows how to terminate a PSB with the TERM command.

## System Service Commands

The following system service commands require that you first issue the SCHD command with the SYSSERVE keyword:

- "ACCEPT Command"
- "DEQ Command" on page 68
- "LOG Command" on page 70
- "QUERY Command" on page 70
- "REFRESH Command" on page 71
- "ROLS Command" on page 73
- "SETS Command" on page 75
- "SETU Command" on page 76
- "STAT Command" on page 77

The following system service commands are valid in batch or BMP regions or programs without first issuing the SCHD command with the SYSSERVE keyword:

- "CHKP Command" on page 67
- "ROLB Command" on page 72
- "ROLL Command" on page 73
- "SYMCHKP Command" on page 78
- "XRST Command" on page 79

The following system service commands are valid in an online CICS program using DBCTL:

- ACCEPT
- DEQ
- LOG
- QUERY
- REFRESH
- ROLS
- SETS
- STAT

To issue system service commands, the input/output PCB (I/O PCB) is required. For detailed information on the I/O PCB, as well as the PSB, and PCB, see "PCBs and PSB" on page 31.

## ACCEPT Command

The Accept (ACCEPT) command is used to tell IMS to return a status code to your program, rather than abend the transaction.

### Format

```
▶▶—EXEC—DLI—ACCEPT STATUSGROUP('A')—▶▶  
└─ACCEPT STATUSGROUP('B')└─
```

## Options

### STATUSGROUP('A')

Informs IMS that the application is prepared to accept status codes regarding unavailability. IMS then returns a status code instead of pseudoabending if a call issued later requires access to unavailable data.

This is a required option.

### STATUSGROUP('B')

Informs IMS that the application is prepared to accept status codes regarding unavailability and deadlock occurrence. IMS returns a status code instead of pseudoabending if a call issued later requires access to unavailable data or deadlock occurrence.

## Usage

Use the ACCEPT command to tell IMS to return a status code instead of abending the program. These status codes result because PSB scheduling completed without all of the referenced databases being available.

## Example

```
EXEC DLI ACCEPT STATUSGROUP('A');
```

This example shows how to specify the ACCEPT command.

## CHKP Command

The Checkpoint (CHKP) command is used to issue a basic checkpoint and to end a logical unit of work. You cannot use this command in a CICS program.

## Format

```
▶▶ EXEC DLI CHECKPOINT ID(area)
  CHKP ID('literal') ▶▶
```

## Options

### ID(area)

Contains the checkpoint ID. Specifies the name of an area in your program containing the checkpoint ID. The area pointed to is eight bytes. If you are using PL/I, specify this option as a pointer to a major structure, an array, or a character string.

### ID('literal')

'literal' is an 8-byte checkpoint ID, enclosed in quotation marks. In CHKP commands the area pointed to is 8 bytes long.

## Usage

The two kinds of commands that allow you to make checkpoints are: the CHKP, or basic Checkpoint command, and the SYMCHKP, or Symbolic Checkpoint command.

Batch programs can use either the symbolic or the basic command.

Both checkpoint commands make it possible for you to commit your program's changes to the database and to establish places from which the program can be restarted, should it terminate abnormally.

You must not use the CHKPT=EOV parameter on any DD statement to take an IMS checkpoint.

## CHKP Command

Both commands cause a loss of database position at the time the command is issued. Position must be reestablished by a GU command or other method of establishing position.

It is not possible to re-establish position in the midst of nonunique keys or nonkeyed segments.

You can issue the basic CHKP command to commit your program's changes to the database and establish places from which your program can be restarted. When you issue a basic CHKP command, you must provide the code for restarting your program.

When you issue a CHKP command, you specify the ID for the checkpoint. You can supply either the name of a data area in your program that contains the ID, or you can supply the actual ID, enclosed in single quotes. See "Examples."

### Examples

```
EXEC DLI CHKP ID(chkpid);
EXEC DLI CHKP ID('CHKP0007');
```

**Explanation:** These examples show how to specify the CHKP command.

### Restrictions

Restrictions for the CHKP command:

- You cannot use this command in a CICS program.
- You must first define an I/O PCB for your program before you can use the CHKP command.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.

## DEQ Command

The Dequeue (DEQ) command is used to release a segment that is retrieved with the LOCKCLASS option.

### Format

```
▶▶ EXEC—DLI—DEQ—LOCKCLASS(data_value)—————▶▶
```

### Option

#### LOCKCLASS(*data\_value*)

Specifies that you want to release the lock that is being held as the result of an earlier GU, GN, or GNP command that had a LOCKCLASS option with the same *data\_value*. *Data\_value* must be a 1-byte alphabetic character in the range of B to J.

For full function, specify the LOCKCLASS option followed by the lock class of that segment (for example, LOCKCLASS('B')). If the option is not followed by a letter (B-J), EXECDLI sets a status code of GL and initiates an ABENDU1041.

DEQ commands are not supported for Fast Path.

### Usage

Use the DEQ command to release locks on segments that were retrieved using the LOCKCLASS option. Using LOCKCLASS on Get commands allows you to reserve segments for exclusive use by your transaction. No other transaction is allowed to

update these reserved segments until either your transaction reaches a sync point or the DEQ command has been issued, thereby releasing the locks on these reserved segments. The LOCKCLASS option lets your application program leave these segments and retrieve them later without any changes having been added.

### Example

Your program can use the LOCKCLASS option as follows:

```
EXEC DLI DEQ LOCKCLASS(data_value)
EXEC DLI GU SEGMENT(PARTX)
      SEGMENT(ITEM1) LOCKCLASS('B') INTO(PTAREA1);
EXEC DLI GU SEGMENT(PARTX)
      SEGMENT(ITEM2) LOCKCLASS('C') INTO(PTAREA2);
EXEC DLI DEQ LOCKCLASS('B');
```

**Explanation:** This example shows the format of the DEQ command, where *data\_value* is a 1-byte alphabetic character in the range B to J. The DEQ command releases the lock that was gotten and held with a LOCKCLASS of 'B' for the PARTX segment as a result of the first GU. The lock that was gotten with a LOCKCLASS of 'C' on the PARTX segment during the second GU remains held.

### Restriction

Restrictions for the DEQ command:

- To use this command you must first define an I/O PCB for your program.

## LOAD Command

The Load (LOAD) command is used to add a segment sequentially while loading the database.

### Format

```
▶ EXEC—DLI—LOAD —————▶
                    | USING PCB(expression) | | VARIABLE |
                    |-----| |-----|
▶ SEGMENT(name) —————▶ FROM(area)
  | SEGMENT((area)) | | SEGLENGTH(expression) |
  |-----| |-----|
```

### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### VARIABLE

Indicates that a segment is variable-length.

#### SEGMENT(name)

Specifies the name of the segment type you want to retrieve, insert, delete, or replace.

#### SEGMENT((area))

A reference to an area in your program containing the name of the segment type. You can specify an area instead of the name of the segment in the command.

#### SEGLENGTH(expression)

Specifies the length of the I/O area from which the segment is obtained. Its argument can be any expression that converts to the integer data type; you can

## LOAD Command

specify either a number or a reference to a halfword in your program containing a number. (SEGLength is required in COBOL programs for any SEGMENT level that specifies the INTO or FROM option.)

**Requirement:** The value specified for SEGLength must be greater than or equal to the length of the longest segment that can be processed by this call.

### FROM(area)

Specifies an area containing the segment to be added, replaced, or deleted.

### Usage

The LOAD command is used for database load programs, which are described in *IMS Version 9: Administration Guide: Database Manager*.

### Example

```
EXEC DLI LOAD
      SEGMENT(ILLNESS) FROM(ILLAREA);
```

## LOG Command

The Log (LOG) command is used to write information to the system log.

### Format

```
▶▶—EXEC—DLI—LOG—FROM(area)—LENGTH(expression)—▶▶
```

### Options

#### FROM(area)

Specifies an area containing the segment to be added, replaced, or deleted.

#### LENGTH(expression)

Specifies the length of an area.

### Usage

You use the LOG command to write information to the system log. For detailed information on this command, see *IMS Version 9: Application Programming: Design Guide*.

### Example

```
EXEC DLI LOG
      FROM(ILLAREA) LENGTH(18);
```

### Restriction

Restrictions for the LOG command:

- To use this command you must first define an I/O PCB for your program.

## QUERY Command

The Query (QUERY) command obtains status code and other information in the DL/I interface block (DIB), which is a subset of the IMS PCB.

### Format

```
▶▶—EXEC—DLI—QUERY—USING—PCB(expression)—▶▶
```



### Options

USING PCB(*expression*) is required. No other options are allowed with the QUERY command.

### Usage

For full-function databases, the DIB should contain NA, NU, TH or blanks. For an explanation of the codes, see *IMS Version 9: Messages and Codes, Volume 1*.

Use the QUERY command after scheduling the PSB but before making the database call. If the program has already issued a call using the DB PCB, you then use the REFRESH command to update the information in the DIB.

### example

#### Example 1:

```
EXEC DLI QUERY USING PCB(expression);
```

*Explanation:* This example shows how to specify the QUERY command. In this example, (n) specifies the PCB.

#### Example 2:

```
EXEC DLI REFRESH DBQUERY;
```

*Explanation:* If your program has already issued a call using the DB PCB name, use the REFRESH command to update the information in the DIB. The REFRESH command updates all DB PCBs. You can issue it only one time.

### Restrictions

Restrictions for the QUERY command:

- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.

## REFRESH Command

The Refresh (REFRESH) command is used to obtain the most recent information from the DIB for the most recently issued command.

### Format

▶▶—EXEC—DLI—REFRESH—DBQUERY—————▶▶

### Options

DBQUERY is required. Other options are not allowed with the REFRESH command.

### Usage

The REFRESH command is used with the QUERY command.

The QUERY command is used after scheduling the PSB but before making the first database call. If the program has already issued a call using the DB PCB, use the REFRESH command to update the information in the DIB.

The REFRESH command updates all DB PCBs. It can be issued only once.

## REFRESH Command

### Example

```
EXEC DLI REFRESH DBQUERY;
```

**Explanation:** This example shows how to specify the REFRESH command.

### Restrictions

Restrictions for the REFRESH command:

- To use this command, you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You can issue this command only one time.

## ROLB Command

The Rollback (ROLB) command is used to dynamically back out changes and return control to your program. You cannot use this command in a CICS program.

### Format

```
▶▶—EXEC—DLI—ROLB—▶▶
```

### Options

No options are allowed with the ROLB command.

### Usage

When a batch or BMP program determines that some of its processing is invalid, two commands make it possible for the program to remove the effects of its inaccurate processing. These are the rollback commands, ROLL (see “ROLL Command” on page 73) and ROLB.

The ROLB command is valid in batch programs when the system log is stored on direct access storage and dynamic backout has been specified through the use of the BKO execution parameter.

Issuing the ROLB causes IMS DB to back out any changes your program has made to the database since its last checkpoint, or since the beginning of the program if your program has not issued a checkpoint. When you issue a ROLB command, IMS DB returns control to your program after backing out the changes, so that your program can continue processing with the next statement after the ROLB command.

### Example

```
EXEC DLI ROLB;
```

**Explanation:** This example shows how to dynamically back out changes and return control to your program with the ROLB command.

### Restrictions

Restrictions for the ROLB command:

- You cannot use this command in a CICS program.
- You must first define an I/O PCB for your program before you can use this command.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.

- You cannot use this command when the system log is stored on direct access storage and dynamic backout has not been specified.

## ROLL Command

The Roll (ROLL) command is used to dynamically back out changes. You cannot use this command in a CICS program

### Format

▶▶—EXEC—DLI—ROLL—————▶▶

### Options

No options are allowed with the ROLL command.

### Usage

When a batch program determines that some of its processing is invalid, two commands make it possible for the program to remove the effects of its inaccurate processing. These are the rollback commands, ROLL and ROLB (see “ROLB Command” on page 72).

You can use ROLL in batch programs.

Issuing the ROLL causes CICS and DL/I to back out any changes your program has made to the database since its last checkpoint, or since the beginning of the program provided your program has not issued a checkpoint. When you issue a ROLL command, DL/I terminates your program after backing out the updates.

### Example

```
EXEC DLI ROLL;
```

**Explanation:** This example shows how to dynamically back out changes with the ROLL command.

If you use the ROLL command, IMS terminates the program with user abend code U0778. This type of abnormal termination does not produce a storage dump.

### Restrictions

Restrictions for the ROLL command:

- You cannot use this command in a CICS program.
- You must first define an I/O PCB for your program before you can use this command.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## ROLS Command

The Rollback to SETS or SETU (ROLS) command is used to back out to a processing point set by an earlier SETS command.

### Format

```
► EXEC—DLI—ROLS—  
└── USING PCB(expression)—  
└── TOKEN(token)—AREA(data_area)—
```

### Options

#### USING PCB(*expression*)

Specifies the DB PCB you want to use. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### TOKEN(*token*)

A 4-byte token associated with the current processing point. If you specify both TOKEN and AREA, the ROLS command backs out to the SETS or SETU you specified.

#### AREA(*data\_area*)

The name of the area to be restored to the program when a ROLS command is issued. The first 2 bytes of the data-area field contain the length of the data-area, including the length itself. The second 2 bytes must be set to X'0000'. If you specify both TOKEN and AREA, the ROLS command backs out to the SETS you specified.

The ROLS call has two formats: with TOKEN and AREA (for IOPCB only) and without TOKEN and AREA (for IOPCB or DBPCB).

### Usage

Use the SETS and ROLS commands to define multiple points at which to preserve the state of DL/I full-function databases and to return to these points later. (For example, you can use them so your program can handle situations that can occur when PSB scheduling completes without all of the referenced DL/I databases being available.)

Use of the SETS and ROLS commands apply only to DL/I full-function databases. This means that if a logical unit of work (LUW) is updating types of recoverable resources other than full-function databases, for example, VSAM files, the SETS and ROLS requests have no effect on the non-DL/I resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. It is up to you to ensure the consistency of all the resources involved.

You can use the ROLS command to backout to the state all full-function databases were in before either a specific SETS or SETU request or the most recent commit point.

### Examples

#### Example 1:

```
EXEC DLI ROLS TOKEN(token1) AREA(data_area)
```

*Explanation:* In this example (for IOPCB only), backout takes place to the corresponding TOKEN, as specified by a prior SETS call, and control returns to the application.

#### Example 2:

```
EXEC DLI ROLS USING PCB(PCB5)
```

*Explanation:* In this example, for IOPCB or DBPCB, backout takes place to the prior sync point and the application is pseudoabended with a U3033 status code. Control does **not** return to the application.

In this example, PCB5 is the number of a DB PCB that has received a 'data unavailable' status code. This command results in the same action that would have occurred had the program not issued an ACCEPT STATUSGROUPA command. (See Chapter 8, "Data Availability Enhancements," on page 105.)

**Example 3:**

```
EXEC DLI ROLS
```

*Explanation:* In this example, for IOPCB or DBPCB, backout takes place to the prior sync point and the application is pseudoabended with a U3033, provided the previous reference to that PCB gave an unavailable status code. Control does **not** return to the application.

**Restrictions**

Restrictions for the ROLS command:

- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

**SETS Command**

The Set a Backout Point (SETS) command is used to define points in your application at which to preserve the state of the DL/I databases before initiating a set of DL/I requests to perform a function. Your application can issue a ROLS command later if it cannot complete the function.

**Format**



**Options**

**TOKEN(mytoken)**

A 4-byte token associated with the current processing point.

**AREA(data\_area)**

The name of the area to be restored to the program when a SETS command is issued. The first 2 bytes of the data-area field contain the length of the data-area, including the length itself. The second 2 bytes must be set to X'0000'.

**Usage**

You can use the SETS command to define multiple points at which to preserve the state of the DL/I databases and to return to these points later. For example, you can use the SETS command to allow your program to handle situations that can occur when PSB scheduling completed without all of the referenced DL/I databases being available.

## SETS Command

The SETS command applies only to DL/I full-function databases. If a logical unit of work (LUW) is updating types of recoverable resources other than full-function databases, for example VSAM files, the SETS command has no effect on the non-DL/I resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. It is up to you to ensure the consistency of all the resources involved.

### Example

```
EXEC DLI SETS TOKEN(mytoken) AREA(data_area)
```

**Explanation:** This example shows how to specify the SETS command.

### Restrictions

Restrictions for the SETS command:

- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- In batch, you can only use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.
- It is rejected when the PSB contains a DEDB or MSDB PCB, or when the call is made to a DB2 database.
- It is valid, but not functional, if unsupported PCBs exist in the PSB or if the program uses an external subsystem.

## SETU Command

The Set a Backout Point Unconditionally (SETU) command is identical to the SETS command except that it does not get rejected if unsupported PCBs are in the PSB or if the program uses an external subsystem.

### Format

►►—EXEC—DLI—SETU—TOKEN(mytoken)—AREA(data\_area)—►►

### Options

#### TOKEN(mytoken)

A 4-byte token associated with the current processing point.

#### AREA(data\_area)

The name of the area to be restored to the program when a SETU command is issued. The first 2 bytes of the data-area field contain the length of the data-area, including the length itself. The second 2 bytes must be set to X'0000'.

### Usage

You can use the SETU command to define multiple points at which to preserve the state of the DL/I databases and to return to these points later. For example, you can use the SETU command to allow your program to handle situations that can occur when PSB scheduling completed without all of the referenced DL/I databases being available.

The SETU command applies only to DL/I full-function data bases. If a logical unit of work (LUW) is updating types of recoverable resources other than full-function databases, such as VSAM files, the SETU command has no effect on the non-DL/I

resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. It is up to you to ensure the consistency of all the resources involved.

### Example

```
EXEC DLI SETU TOKEN(mytoken) AREA(data_area)
```

**Explanation:** This example shows how to specify the SETU command.

### Restrictions

Restrictions for the SETU command:

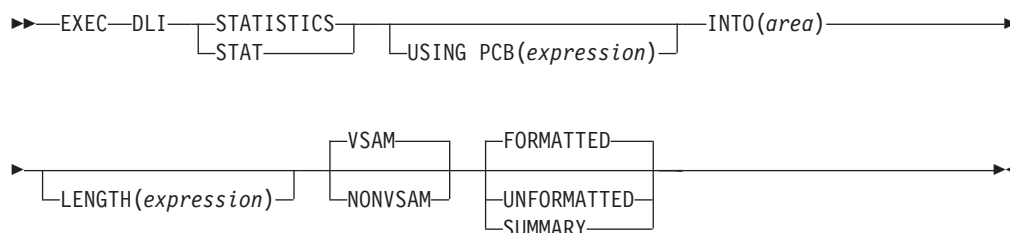
- You cannot use this command in a CICS program.
- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command when the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## STAT Command

This topic contains programming interface information.

The Statistics (STAT) command is used to obtain IMS database statistics that you can use in debugging your program.

### Format



### Options

#### USING PCB(expression)

Specifies the DB PCB you want to use. Its argument can be any expression that converts to the integer data type; you can specify either a number or a reference to a halfword in your program containing a number.

#### INTO(area)

Specifies an area into which the data is read.

#### LENGTH(expression)

Specifies the length of an area.

#### VSAM/NONVSAM

Specifies database type.

#### FORMATTED/UNFORMATTED/SUMMARY

Specifies type of output.

## STAT Command

### Usage

The STAT command is described in *IMS Version 9: Application Programming: Design Guide*.

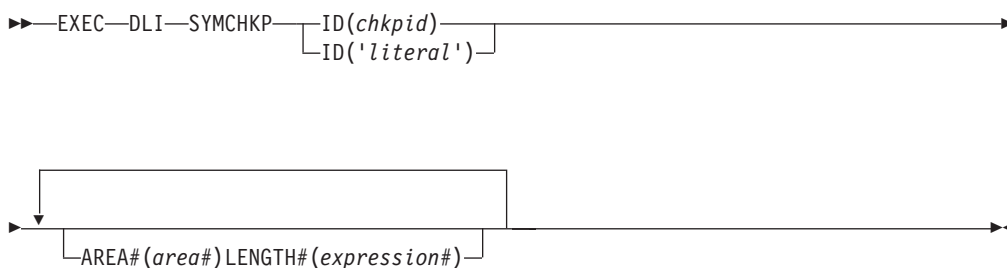
### Example

For examples of the STAT command, see *IMS Version 9: Application Programming: Database Manager*.

## SYMCHKP Command

The Symbolic Checkpoint (SYMCHKP) command is used to issue a symbolic checkpoint and to end a logical unit of work.

### Format



### Options

#### ID(*chkpid*)

Is the name of an 8-byte area in your program containing the checkpoint ID. If you are using PL/I, specify this parameter as a pointer to a major structure, an array, or a character string.

#### ID('literal')

Is the 8-byte checkpoint ID, enclosed in quotation marks.

#### AREA#(*area#*)

Specifies the areas in your program you want IMS to checkpoint. You do not need to specify any area to checkpoint; however, you cannot specify more than seven areas. If you specify more than one area, you must include all intervening areas. For example, if you specify AREA3, you must also specify AREA1 and AREA2. The areas you specify using the SYMCHKP command must be the same and in the areas specified in the XRST command.

#### LENGTH#(*expression#*)

Can be any expression in the host language that converts to the integer data type; you can specify either a number or a reference to a halfword containing a number. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or assembler language programs, LENGTH1 to LENGTH7 are optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, LENGTHx (where x is 1 to 7) is required for each AREA<sub>x</sub> (where x is 1 to 7) that you specify.

### Usage

The two kinds of commands that allow you to make checkpoints are: the CHKP, or basic Checkpoint command, and the SYMCHKP, or Symbolic Checkpoint command.

Batch programs can use either the symbolic or the basic command.



Both checkpoint commands make it possible for you to commit your program's changes to the database and to establish places from which the program can be restarted, should it terminate abnormally. You must not use the `CHKPT=EOV` parameter on any DD statement to take an IMS checkpoint.

Refer to *IMS Version 9: Application Programming: Design Guide* for an explanation of when and why you should issue checkpoints in your program. Both commands cause a loss of database position at the time the command is issued. Position must be reestablished by a GU command or other method of establishing position.

In addition to committing your program's changes to the database and establishing places from which your program can be restarted, the Symbolic Checkpoint command:

- Works with the Extended Restart (XRST) command to restart your program if it terminates abnormally.
- Can save as many as seven data areas in your program, which are restored when your program is restarted. You can save variables, counters, and status information.

### Example

```
EXEC DLI SYMCHKP
      ID(chkpid)
      AREA1(area1) LENGTH1(expression1)
      ...
      AREA7(area7) LENGTH7(expression7)
```

**Explanation:** This example shows how to issue a symbolic checkpoint and to end a logical unit of work with a SYMCHKP command.

### Restrictions

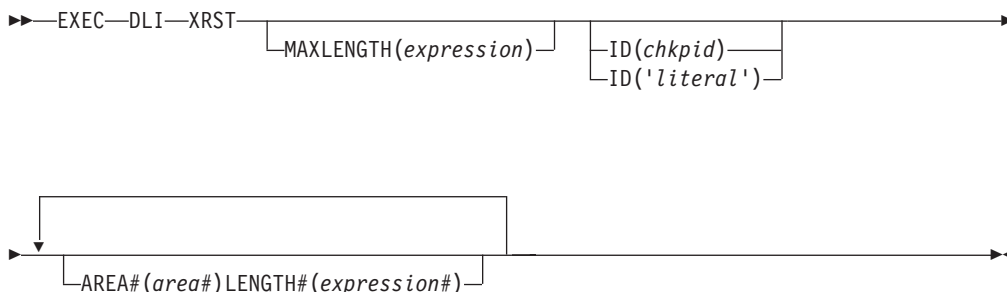
Restrictions for the SYMCHKP command:

- If you issue this command, you must also issue the XRST command.
- You cannot use this command in a CICS program.
- To use the SYMCHKP command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- The areas you specify using the SYMCHKP command must be the same, and in the same order, as the areas specified in the XRST command.
- If you specify more than one area, you must specify all intervening areas. For example, if you specify AREA3, you must also specify AREA1 and AREA2.
- When specifying expression1 with a COBOL program that is not compiled with the IBM COBOL for z/OS & VM (or the VS COBOL II) compiler, LENGTHx (where x is 1 to 7) is required for each AREAx (where x is 1 to 7) that you specify.

## XRST Command

The Extended Restart (XRST) command is used to issue an extended restart, and to perform a normal start or an extended restart from a checkpoint ID or time/date stamp. If you use Symbolic Checkpoint commands in your program, you must use the XRST command.

## Format



## Options

### MAXLENGTH(expression)

Specifies the length of an area from which a program is restarted. This parameter is the longest segment in the PSB, or of all the segments in a path, if you use path commands in your program. It can be any expression in the host language that converts to the integer data type. You can specify either a number or a reference to a halfword containing a number. MAXLENGTH is not required, and defaults to 512 bytes.

### ID(chkpid) ID('literal')

This parameter is either the name of a 30-byte area in your program or a 30-byte checkpoint ID, enclosed in quotes. This parameter is optional; you can specify a checkpoint ID or a time/date stamp in the parm field of your JCL instead. If you specify both, IMS uses the value in the parm field of the EXEC statement. If you are starting your program normally, do not specify a checkpoint ID, or ensure that the field pointed to by the *chkpid* contains blanks.

If your program is restarted and the CKPTID= value in the PARM field of the EXEC statement is not used, then the rightmost bytes beyond the checkpoint ID being used in the I/O area must be set to blanks.

You can issue a XRST command after supplying a time/date stamp of IIIIDDHMMSSST, or from a specific checkpoint in your program by supplying a checkpoint ID. IIIIDD is the region ID and day; HHMMSSST is the actual time in hours, minutes, seconds, and tenths of seconds. The system message DFS0540I supplies the checkpoint ID and time/date stamp.

If you are using PL/I, specify *chkpid* as a pointer to a major structure, an array, or a character string.

### AREA#(area#)

Area# specifies the first area in your program you want to restore. You can specify up to seven areas. You are not required to specify any areas; however, if you specify more than one area, you must include all intervening areas. For example, if you specify AREA3, you must also specify AREA1, and AREA2. The areas you specify on the XRST command must be the same—and in the same order—as the areas you specify on the SYMCHKP command. When you restart the program, only the areas you specified in the SYMCHKP command are restored.

### LENGTH#(expression#)

Specifies the length of an area from which a program is restarted. Its argument can be any expression in the host language that converts to the integer data type; you can specify either a number or a reference to a halfword containing a number. For IBM COBOL for z/OS & VM (or VS COBOL II), PL/I, or

assembler language programs LENGTH1 to LENGTH7 are optional. For COBOL programs that are not compiled with the IBM COBOL for z/OS & VM (or VS COBOL II) compiler, LENGTHx (where x is 1 to 7) is required for each AREAx (where x is 1 to 7) that you specify. Each qualification statement consists of:

- The name of a field in a segment
- The relational operator, which indicates how you want the two values compared
- The name of a data area in your program containing the value that is compared against the value of the field

### Usage

If your programs issues Symbolic Checkpoint commands it must also issue the Extended Restart (XRST) command. The XRST is issued once, at the start of your program. You can use the XRST command to start your program normally, or to extend restart it in case of an abnormal termination.

You can extend restart your program from a specific checkpoint ID, or a time/date stamp. Because the XRST attempts to reposition the database, your program also needs to check for correct position.

After issuing the XRST command, you should test the DIBSEGM field in the DIB. After a normal start, the DIBSEGM field should contain blanks. At the completion of an Extended Restart, the DIBSEGM field will contain a checkpoint ID. Normally, XRST will return the 8-byte symbolic checkpoint ID, followed by 4 blanks. If the 8-byte ID consists of all blanks, then XRST will return the 14-byte time-stamp ID. The only successful status code for an XRST command is a blank status code. If DL/I detects any error while processing the XRST command, your program abends.

### Example

```
EXEC DLI XRST MAXLENGTH(expression)
      ID(chkpid)
      AREA1(area1) LENGTH1(expression1)
      ...
      AREA7(area7) LENGTH7(expression7)
```

**Explanation:** This example shows how to specify the XRST command.

### Restrictions

Restrictions for the XRST command:

- You cannot use this command in a CICS program.
- To use this command you must first define an I/O PCB for your program.
- You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.
- You cannot use this command unless the system log is stored on direct access storage and dynamic backout has been specified. You must also specify BKO=Y in the parm field of your JCL when you execute the program.

## XRST Command

---

## Chapter 5. Recovering Databases and Maintaining Database Integrity

This chapter describes the commands you can use to help recover data accessed by your program and maintain data integrity:

- The Basic Checkpoint command, `CHKP`, which you can use to issue checkpoints from a batch or BMP program
- The Symbolic Checkpoint command, `SYMCHKP`, which you can use to issue checkpoints from a batch or BMP program and to specify data areas that can be restored when you restart your program
- The Extended Restart command, `XRST`, which you can use along with symbolic checkpoints to start or restart your batch or BMP program
- The rollback commands, `ROLL` and `ROLB`, which you can use to dynamically back out database changes from a batch or BMP program
- The managing-backout-points commands, `SETS` and `ROLS`, which you can use to set multiple backout points and then return to these points later
- The Dequeue command, `DEQ`, which releases previously reserved segments

To use any of the commands, you must have defined an I/O PCB for your program, except for the `DEDB DEQ` calls, which are issued against a `DEDB PCB`.

The following topics provide additional information:

- “Issuing Checkpoints in a Batch or BMP Program”
- “Restarting Your Program and Checking for Position” on page 84
- “Backing Out Database Updates Dynamically: The `ROLL` and `ROLB` Commands” on page 84
- “Using Intermediate Backout Points: The `SETS` and `ROLS` Commands” on page 85

---

### Issuing Checkpoints in a Batch or BMP Program

The two kinds of commands that allow you to make checkpoints are: the `CHKP`, or Basic Checkpoint command, and the `SYMCHKP`, or Symbolic Checkpoint command.

Batch programs can use either the Symbolic Checkpoint or the Basic Checkpoint command.

Both checkpoint commands make it possible for you to commit your program’s changes to the database and to establish places from which the batch or BMP program can be restarted, in cases of abnormal termination.

**Requirement:** You must not use the `CHKPT=EOV` parameter on any `DD` statement to take an `IMS` checkpoint.

Because both checkpoint commands cause a loss of database position at the time the command is issued, you must reestablish position with a `GU` command or other methods.

You cannot reestablish position in the midst of nonunique keys or nonkeyed segments.

## Issuing Checkpoints

### Issuing the CHKP Command

When you issue a CHKP command, you must provide the code for restarting your program and you must specify the ID for the checkpoint. You can supply either the name of a data area in your program that contains the ID, or you can supply the actual ID, enclosed in single quotes. For example, either of the following commands is valid:

```
EXEC DLI CHKP ID(chkpid);  
EXEC DLI CHKP ID('CHKP0007');
```

### Issuing the SYMCHKP Command

The SYMCHKP command in batch and BMP programs:

- Works with the Extended Restart (XRST) command to restart your program if it terminates abnormally.
- Can save as many as seven program data areas, which are restored when your program is restarted. You can save variables, counters, and status information.

For examples of how to specify the SYMCHKP command, see “SYMCHKP Command” on page 78.

---

## Restarting Your Program and Checking for Position

Programs that issue Symbolic Checkpoint commands must also issue the Extended Restart (XRST) command. You must issue XRST once, as the first command in the program. You can use the XRST command to start your program normally, or to restart it in case of an abnormal termination. You can restart your program from one of the following:

- A specific checkpoint ID
- A time/date stamp

Because the XRST command attempts to reposition the database, your program also needs to check for correct position.

---

## Backing Out Database Updates Dynamically: The ROLL and ROLB Commands

When a batch program determines that some of its processing is invalid, the ROLL and ROLB commands make it possible for the program to remove the effects of its inaccurate processing.

You can use both ROLL and ROLB in batch programs. You can only use the ROLB command in batch programs if the system log is stored on direct access storage and if you have specified BKO=Y in the parm field of your JCL.

Issuing either of these commands causes DL/I to back out any changes your program has made to the database since its last checkpoint, or since the beginning of the program if your program has not issued a checkpoint.

---

## Using Intermediate Backout Points: The SETS and ROLS Commands

Use the SETS and ROLS commands to define multiple points at which to preserve the state of DL/I full-function databases and to return to these points later. (For example, you can use them to allow your program to handle situations that can occur when PSB scheduling complete without all of the referenced DL/I databases being available.)

The SETS and ROLS commands apply only to DL/I full-function databases. Therefore, if a logical unit of work (LUW) is updating recoverable resources other than full-function databases (VSAM files, for example), the SETS and ROLS requests have no effect on the non-DL/I resources. The backout points are *not* CICS commit points; they are intermediate backout points that apply only to DBCTL resources. Your program must ensure the consistency of all the resources involved.

Before initiating a set of DL/I requests to perform a function, you can use a SETS command to define points in your application at which to preserve the state of DL/I databases. Your application can issue a ROLS command later if it cannot complete the function. You can use the ROLS command to back out to the state all full-function databases were in before either a specific SETS request or the most recent commit point.

## Intermediate Backout Points



---

## Chapter 6. Processing Fast Path Databases

| Using EXEC DLI commands under DBCTL, a CICS program or a batch-oriented  
| BMP program can access DEDBs. Parameters allow your program to use facilities  
| of the DEDBs such as subset pointers.

A DEDB contains a root segment and as many as 127 types of dependent segment. One of these types can be a sequential dependent; the other 126 are direct dependents. Sequential dependent segments are stored in chronological order. Direct dependent segments are stored hierarchically.

DEDBs provide high data availability. Each DEDB can be partitioned, or divided into multiple “areas.” Each area contains a different set of database records. In addition, you can make up to seven copies of each area data set. If an error exists in one copy of an area, application programs can access the data by using another copy of that area. This is transparent to the application program. When an error occurs to data in a DEDB, IMS does not stop the database. It makes the data in error unavailable, but continues to schedule and process application programs. Programs that do not need the data in error are unaffected.

DEDBs can be shared among application programs in separate IMS systems. Sharing DEDBs is virtually the same as sharing full-function databases, and most of the same rules apply. IMS systems can share DEDBs at the area level (instead of at the database level as with full-function databases), or at the block level.

The following topics provide additional information:

- “Processing DEDBs with Subset Pointers”
- “The POS Command” on page 97

---

### Processing DEDBs with Subset Pointers

Subset pointers and the options you use with them are optimization tools that significantly improve the efficiency of your program when you need to process long segment chains. Subset pointers divide a chain of segment occurrences under the same parent into two or more groups, or subsets. You can define as many as eight subset pointers for any segment type. You then define the subset pointers from within an application program (see “Subset Pointer Options” on page 90). Each subset pointer points to the start of a new subset. For example, in Figure 3 on page 88 suppose you defined one subset pointer that divided the last three segment occurrences from the first four. Your program can then refer to that subset pointer through options, and directly retrieve the last three segment occurrences.

## Processing DEDBs with Subset Pointers

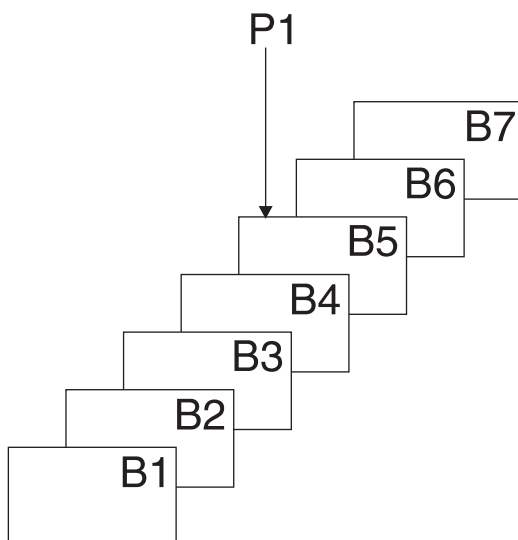


Figure 3. Processing a Long Chain of Segment Occurrences with Subset Pointers

You can use subset pointers at any level of the database hierarchy, except at the root level. Subset pointers used for the root level are ignored.

Figure 4 and Figure 5 on page 89 show some of the ways you can set subset pointers. Subset pointers are independent of one another, which means that you can set one or more pointers to any segment in the chain. For example, you can set more than one subset pointer to a segment, as shown in Figure 4.

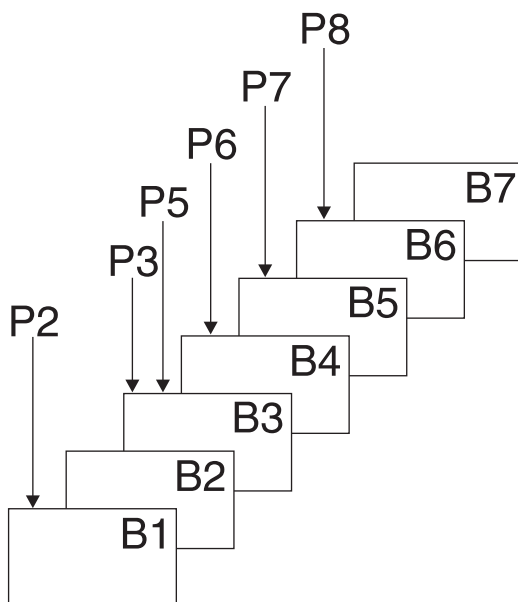


Figure 4. Examples of Setting Multiple Subset Pointers

Alternatively, you can define a one-to-one relationship between the pointers and the segments, as shown in Figure 5 where each segment occurrence has one subset pointer.

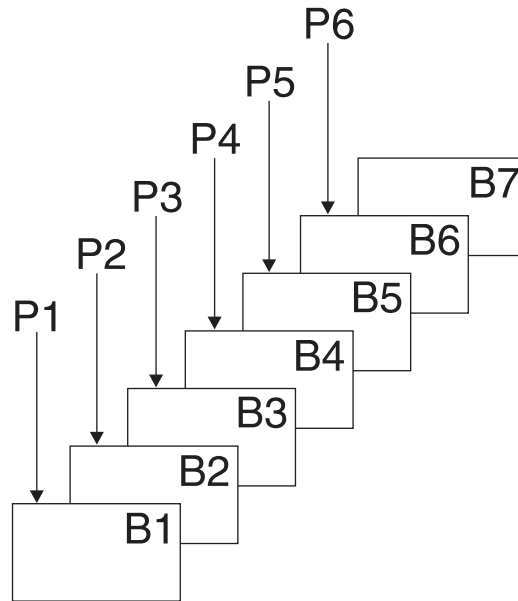


Figure 5. More Examples of Setting Subset Pointers

Figure 6 on page 89 shows how the use of subset pointers divides a chain of segment occurrences under the same parent into subsets. Each subset ends with the last segment in the entire chain. For example, the last segment in the subset defined by subset pointer 1 is B7.

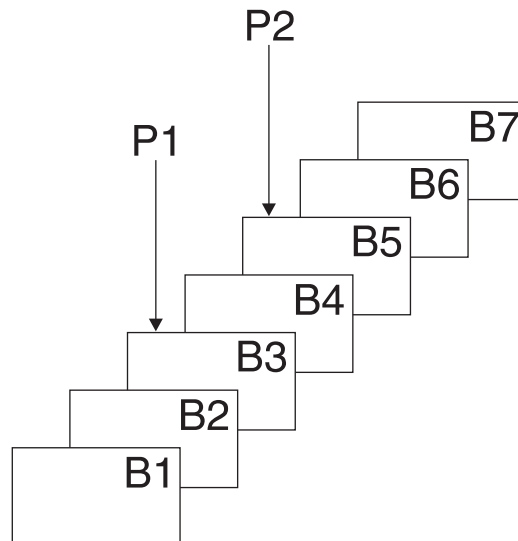


Figure 6. How Subset Pointers Divide a Chain into Subsets

## Preparing to Use Subset Pointers

For your program to use subset pointers, the pointers must be defined in the DBD for the DEDB, and in your program's PSB:

- In the DBD, you specify the number of pointers for a segment chain. You can specify as many as eight pointers for any segment chain.
- In the PSB, you specify which pointers your program uses; you define this on the SENSEG statement. (Each pointer is defined as an integer from 1 to 8.) You also specify on the SENSEG statement whether your program can set the

## Processing DEDBs with Subset Pointers

pointers it uses. If your program has read-only sensitivity, it cannot set pointers, but can only retrieve segments using subset pointers already set. If your program has update sensitivity, it can update subset pointers by using the SET, SETCOND, MOVENEXT, and SETZERO options.

After the pointers are defined in the DBD and the PSB, an application program can set the pointers to segments in a chain. When an application program finishes executing, the subset pointers used by that program remain as they were set by the program and are not reset.

### Designating Subset Pointers

To use subset pointers in your program, you must know the numbers for the pointers as they were defined in the PSB. Then, when you use the subset pointer options, you specify the number for each subset pointer you want to use immediately after the option; for example, you would use P3 to indicate that you want to retrieve the first segment occurrence in the subset defined by subset pointer 3. No default exists, so if you do not include a number between 1 and 8, IMS considers your qualification statement invalid and returns an AJ status code to your program.

### Subset Pointer Options

To take advantage of subsets, application programs use five different options.

<b>GETFIRST</b>	Allows you to retrieve the first segment in a subset.
<b>SETZERO</b>	Sets a subset pointer to zero.
<b>MOVENEXT</b>	Sets a subset pointer to the segment following the current segment. Current position is at the current segment.
<b>SET</b>	Unconditionally sets a subset pointer to the current segment. Current position is at the current segment.
<b>SETCOND</b>	Conditionally sets a subset pointer to the current segment. Current position is at the current segment.

### Banking Transaction Application Example

The examples in this chapter are based on a sample application, the recording of banking transactions for a passbook account. The transactions are written to a database as either posted or unposted, depending on whether they were posted to the customer's passbook. For example, when Bob Emery does business with the bank, but forgets to bring in his passbook, an application program writes the transactions to the database as unposted. The application program sets a subset pointer to the first unposted transaction, so it can be easily accessed later. The next time Bob remembers to bring in his passbook, a program posts the transactions. The program can directly retrieve the first unposted transaction using the subset pointer that was previously set. After the program has posted the transactions, it sets the subset pointer to zero; an application program that subsequently updates the database can determine that no unposted transactions exist. Figure 7 on page 91 summarizes the processing performed when the passbook is unavailable.

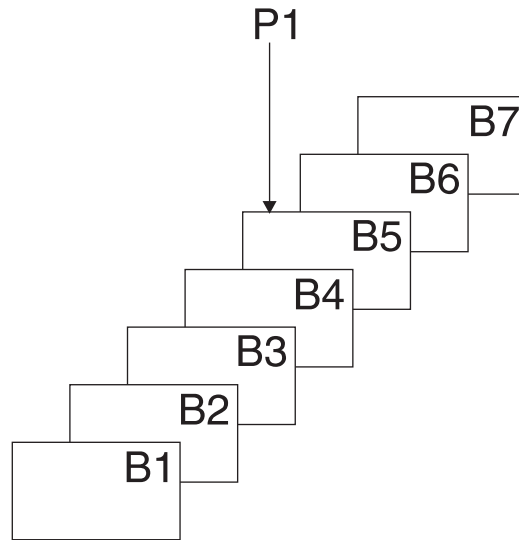


Figure 7. Processing Performed for the Sample Passbook Example when the Passbook is Unavailable

When the passbook is available, an application program adds the unposted transactions to the database, setting subset pointer 1 to the first unposted transaction. Figure 8 summarizes the processing performed when the passbook is available.

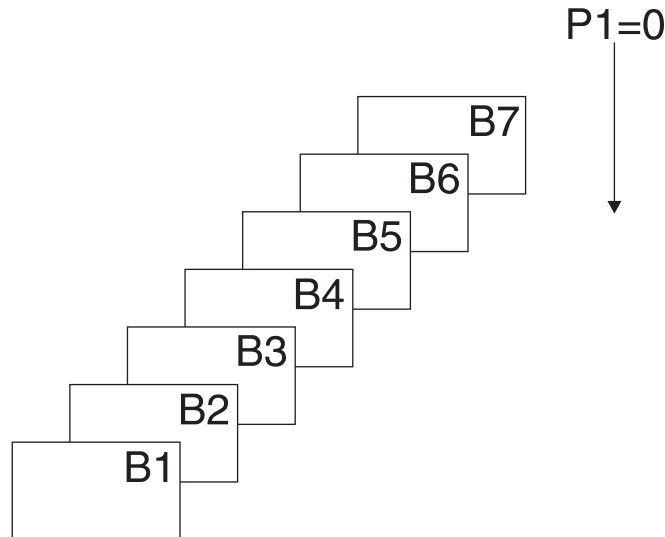


Figure 8. Processing Performed for the Sample Passbook Example when the Passbook is Available

When the passbook is available, an application program retrieves the first unposted transaction using the program, then posts all unposted transactions, setting subset pointer 1 to zero.

**GETFIRST Option: Retrieving the First Segment of a Subset**

To retrieve the first segment occurrence in the subset, your program issues a Get command with the GETFIRST option. The GETFIRST option does not set or move the pointer, but indicates to IMS that you want to establish position on the first

## Processing DEDBs with Subset Pointers

segment occurrence in the subset. The GETFIRST option is like the FIRST option, except that the GETFIRST option applies to the subset instead of to the entire segment chain.

Using the “Banking Transaction Application Example” on page 90, imagine that Bob Emery visits the bank with his passbook and you want to post all of the unposted transactions. Because subset pointer 1 was previously set to the first unposted transaction, your program can use the following command to retrieve that transaction:

```
EXEC DLI GU SEGMENT(A) WHERE(AKEY = 'A1')
      SEGMENT(B) INTO(BAREA) GETFIRST('1');
```

As shown in Figure 9, this command retrieves segment B5. To continue processing segments in the chain, you can issue Get Next commands, as you would if you were not using subset pointers.

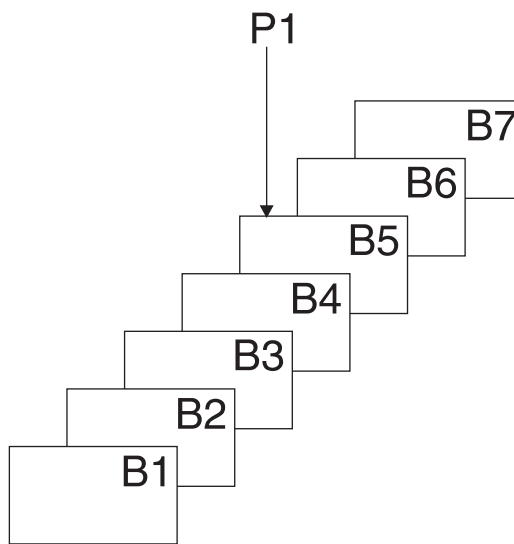


Figure 9. Retrieving the First Segment in a Chain of Segments

If the subset does not exist (subset pointer 1 has been set to zero), IMS returns a GE status code, and your position in the database immediately follows the last segment in the chain. Using the passbook example, the GE status code indicates that no unposted transactions exist.

You can specify only one GETFIRST option per qualification statement; if you use more than one GETFIRST in a qualification statement, IMS returns an AJ status code to your program. The rules for using the GETFIRST option are:

1. You can use GETFIRST with all options except:
  - FIRST
  - LOCKCLASS
  - LOCKED
2. Other options take effect after the GETFIRST option has, and position has been established on the first segment in the subset.
3. If you use GETFIRST with LAST, the last segment in the segment chain is retrieved.

4. If the subset pointer specified with GETFIRST is not set, IMS returns a GE status code, not the last segment in the segment chain. See “SETZERO, MOVENEXT, SET, and SETCOND Options: Setting the Subset Pointers.”
5. Do not use GETFIRST with FIRST. This causes you to receive an AJ status code.
6. GETFIRST overrides all insert rules, including LAST.

### SETZERO, MOVENEXT, SET, and SETCOND Options: Setting the Subset Pointers

The SETZERO, MOVENEXT, SET, and SETCOND options allow you to redefine subsets by modifying the subset pointers. Before your program can set a subset pointer, it must establish a position in the database. A command must be fully satisfied before a subset pointer is set. The segment a pointer is set to depends on your current position at the completion of the command. If a command to retrieve a segment is not completely satisfied, and a position is not established, the subset pointers remain as they were before the command was issued.

- **Setting the subset pointer to zero: SETZERO**

The SETZERO option sets the value of the subset pointer to zero. After your program issues a command with the SETZERO option, the pointer is no longer set to a segment; the subset defined by that pointer no longer exists. (IMS returns a status code of GE to your program if you try to use a subset pointer having a value of zero.)

Using the “Banking Transaction Application Example” on page 90, say that you used the GETFIRST option to retrieve the first unposted transaction. You would then process the chain of segments, posting the transactions. After posting the transactions and inserting any new ones into the chain, you would use the SETZERO option to set the subset pointer to zero as shown in the following command:

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
      SEGMENT(B) FROM(BAREA) SETZERO('1');
```

After this command, subset pointer 1 would be set to zero, indicating to a program updating the database later on that no unposted transactions exist.

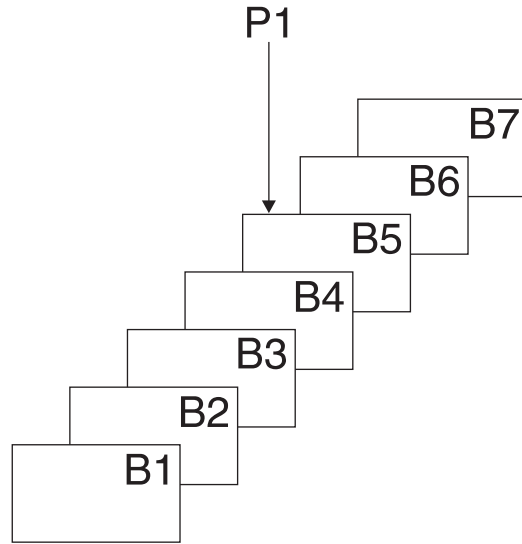
- **Moving the subset pointer forward to the next segment after your current position: MOVENEXT**

To move the subset pointer forward to the next segment after your current position, your program issues a command with the MOVENEXT option. Using the “Banking Transaction Application Example” on page 90, say that you wanted to post some of the transactions, but not all, and that you wanted the subset pointer to be set to the first unposted transaction. The following command sets subset pointer 1 to segment B6.

```
EXEC DLI GU SEGMENT(A) WHERE(AKEY = 'A1')
      SEGMENT(B) INTO(BAREA) GETFIRST('1') MOVENEXT('1');
```

The process of moving the subset pointer with this command is shown in Figure 10 on page 94. If the current segment is the last in the chain, and you use a MOVENEXT option, IMS sets the pointer to zero.

Before the command:



After the command:

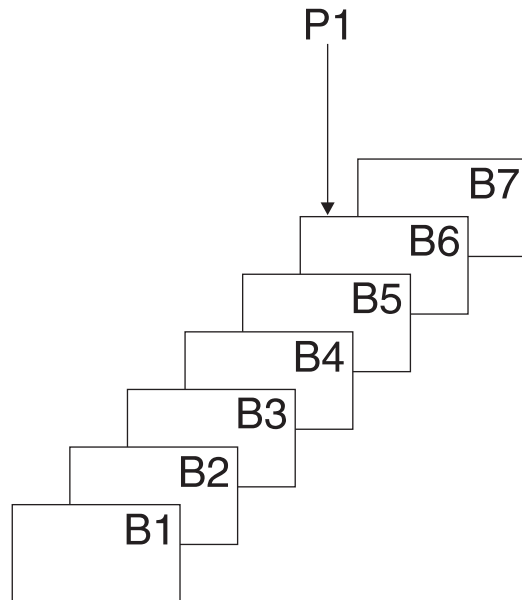


Figure 10. Moving the Subset Pointer to the Next Segment after Your Current Position

- **Setting the subset pointer unconditionally: SET**

You use the SET option to set a subset pointer. The SET option sets a subset pointer unconditionally, regardless of whether or not it is already set. When your program issues a command that includes the SET option, IMS sets the pointer to your current position.

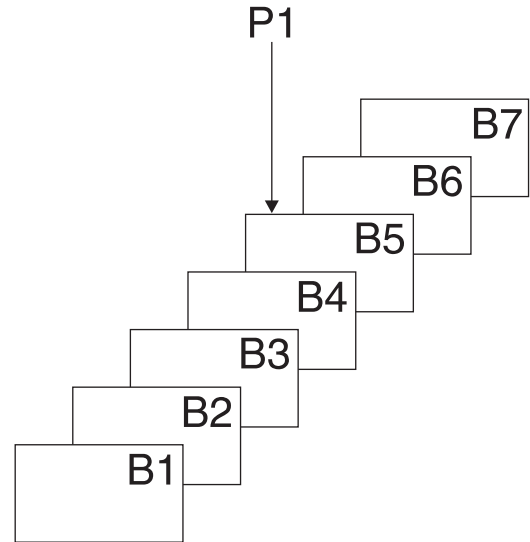
For example, to retrieve the first B segment occurrence in the subset defined by subset pointer 1, and to reset pointer 1 at the next B segment occurrence, you would issue the following commands:

```
EXEC DLI GU SEGMENT(A) WHERE(AKEY = 'A1')  
      SEGMENT(B) INTO(BAREA) GETFIRST('1');  
EXEC DLI GN SEGMENT(B) INTO(BAREA) SET('1');
```

After you have issued these commands, instead of pointing to segment B5, subset pointer 1 points to segment B6, as shown in Figure 11 on page 95.



Before the command:



After the command:

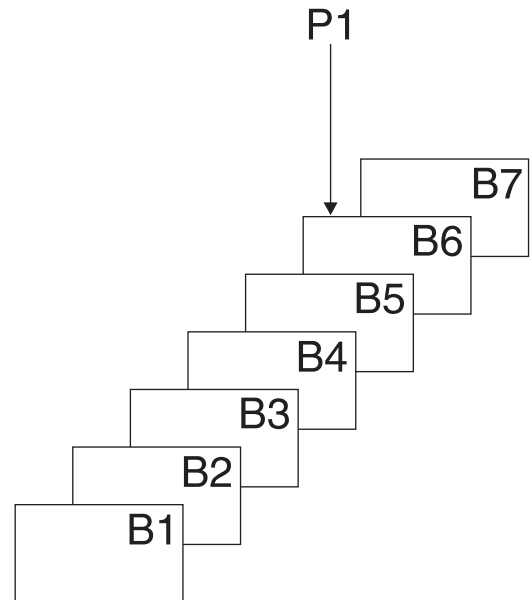


Figure 11. Unconditionally Setting the Subset Pointer to Your Current Position

- **Setting the subset pointer conditionally: SETCOND**

Your program uses the SETCOND option to conditionally set the subset pointer. The SETCOND option is similar to the SET option; the only difference is that, with the SETCOND option, IMS updates the subset pointer only if the subset pointer is *not* already set to a segment.

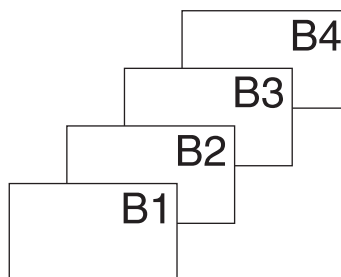
Using the passbook example, say that Bob Emery visits the bank and forgets to bring his passbook; you add the unposted transactions to the database. You want to set the pointer to the first unposted transaction so that when you post the transactions later, you can immediately access the first one. The following command sets the subset pointer to the transaction you are inserting, if it is the first unposted one:

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
      SEGMENT(B) FROM(BAREA) SETCOND('1');
```

As shown by Figure 12 on page 96, this command sets subset pointer 1 to segment B5. If unposted transactions already existed, the subset pointer is not

changed.

Before the command:



After the command:

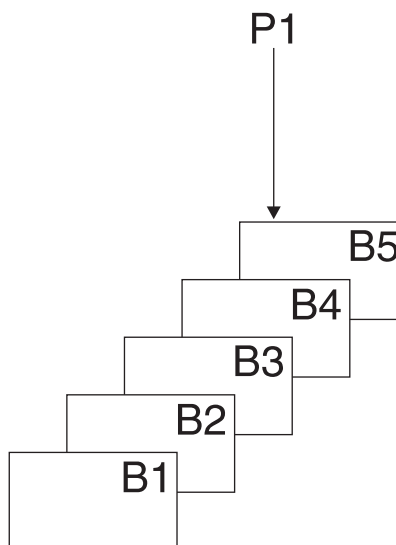


Figure 12. Conditionally Setting the Subset Pointer to Your Current Position

### Inserting Segments in a Subset

When you use the GETFIRST option to insert an unkeyed segment in a subset, the new segment is inserted before the first segment occurrence in the subset. However, the subset pointer is not automatically set to the new segment occurrence. For example, the following command inserts a new B segment occurrence in front of segment B5, but does not set subset pointer 1 to point to the new B segment occurrence:

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
      SEGMENT(B) FROM(BAREA) GETFIRST('1');
```

To set subset pointer 1 to the new segment, you use the SET option along with the GETFIRST option, as shown in the following example:

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
      SEGMENT(B) FROM(BAREA) GETFIRST('1') SET ('1');
```

If the subset does not exist (subset pointer 1 has been set to zero), the segment is added to the end of the segment chain.

### Deleting the Segment Pointed to By a Subset Pointer

If you delete the segment pointed to by a subset pointer, the subset pointer points to the next segment occurrence in the chain. If the segment you delete is the last in the chain, the subset pointer is set to zero.

### Combining Options

You can use the SET, MOVENEXT, and SETCOND options with other options, and you can combine subset pointer options with each other, provided they do not conflict. For example, you can use GETFIRST and SET together, but you cannot use SET and SETZERO together because their functions conflict. If you combine options that conflict, IMS returns an AJ status code to your program.

You can use one GETFIRST option per qualification statement, and one update option (SETZERO, MOVENEXT, SET, or SETCOND) for each subset pointer.

## Subset Pointer Status Codes

If you make an error in a qualification statement that contains subset pointer options, IMS can return these status codes to your program:

**AJ** The qualification statement used a GETFIRST, SET, SETZERO, SETCOND, or MOVENEXT option for a segment for which there are no subset pointers defined in the DBD.

The subset options included in the qualification statement are in conflict; for example, if one qualification statement contained a SET option and a SETZERO option for the same subset pointer, IMS would return an AJ status code. S means to set the pointer to current position; Z means to set the pointer to zero. You cannot use these options together in one qualification statement.

The qualification statement included more than one GETFIRST option.

The pointer number following a subset pointer option is invalid. You either did not include a number, or included an invalid character. The number following the option must be between 1 and 8, inclusive.

**AM** The subset pointer referenced in the qualification statement was not specified in the program's PSB. For example, if your program's PSB specifies that your program can use subset pointers 1 and 4, and your qualification statement referenced subset pointer 5, IMS would return an AM status code to your program.

Your program tried to use an option that updates the pointer (SET, SETCOND, or MOVENEXT) but the program's PSB did not specify pointer update sensitivity.

Your program attempted to open a GSAM database without specifying an IOAREA.

---

## The POS Command

You can use the Position (POS) command (only with DEDBs) to:

- Retrieve the location of a specific sequential dependent segment, or retrieves the location of the last inserted sequential dependent segment.
- Tell you the amount of unused space within each DEDB area. For example, you can use the position information that IMS returns for a POS command to scan or delete the sequential dependent segments for a particular time period.

For the syntax of the POS command, see "POS Command" on page 58.

## The POS Command

If the area the POS command specifies is unavailable, the I/O area is unchanged and the status code FH is returned.

### Locating a Specific Sequential Dependent Segment

When you have position on a particular root segment, you can retrieve the position information and the area name of a specific sequential dependent segment of that root. If you have a position established on a sequential dependent segment, the search starts from that position. IMS returns the position information for the first sequential dependent segment that satisfies the command.

To retrieve this information, you issue a POS command with a qualification statement containing the segment name of the sequential dependent. The current position after this kind of POS command is in the same place that it is after a GNP command.

After a successful POS command, the I/O area contains:

<b>LL</b>	A 2-byte field giving the total length of the data in the I/O area, in binary.
<b>Area Name</b>	An 8-byte field giving the ddname from the AREA statement.
<b>Position</b>	An 8-byte field containing the position information for the requested segment.  If the sequential dependent segment that is the target of the POS command is inserted in the same synchronization interval, no position information is returned. Bytes 11-18 contain X'FF'; other fields contain normal data.
<b>Unused CIs</b>	A 4-byte field containing the number of unused CIs in the sequential dependent part.
<b>Unused CIs</b>	A 4-byte field containing the number of unused CIs in the independent overflow part.

### Locating the Last Inserted Sequential Dependent Segment

You can also retrieve the position information for the most recently inserted sequential dependent segment of a given root segment. To do this, you issue a POS command with a qualification statement containing the root segment as the segment name. The current position after this type of command follows the same rules as position after a GU.

After a successful command, the I/O area contains:

<b>LL</b>	A 2-byte field containing the total length of the data in the I/O area, in binary.
<b>Area Name</b>	An 8-byte field giving the ddname from the AREA statement.
<b>Position</b>	An 8-byte field containing the position information for the most recently inserted sequential dependent segment. This field contains zeros provided no sequential dependent for this root exist.
<b>Unused CIs</b>	A 4-byte field containing the number of unused CIs in the sequential dependent part.
<b>Unused CIs</b>	A 4-byte field containing the number of unused CIs in the independent overflow part.

## Identifying Free Space with the POS Command

To retrieve the area name and the next available position within the sequential dependent part from all online areas, you can issue an unqualified POS command. This type of command also retrieves the free space in the independent overflow and sequential dependent parts.

After a successful unqualified POS command, the I/O area contains the length (LL) followed by the same number of entries as areas within the database. Each entry contains field two through five shown below:

<b>LL</b>	A 2-byte field containing the total length of the data in the I/O area, in binary. The length includes the 2 bytes for the LL field, plus 24 bytes for each entry.
<b>Area Name</b>	An 8-byte field giving the ddname from the AREA statement.
<b>Position</b>	An 8-byte field giving the next available position within the sequential dependent part.
<b>Unused CIs</b>	A 4-byte field containing the number of unused CIs in the sequential dependent part.
<b>Unused CIs</b>	A 4-byte field containing the number of unused CIs in the independent overflow part.

## The P Processing Option

If the P processing option has been specified (with the PROCOPT parameter) in the PCB for your program, a GC status code is returned to your program whenever a command to retrieve or insert a segment causes a Unit of Work (UOW) boundary to be crossed. Although crossing the UOW boundary probably has no particular significance for your program, the GC status code indicates that this is a good time to issue a CHKP command. The advantages of doing this are:

- Your position in the database is kept. Issuing a CHKP normally causes position in the database to be lost, and the application program has to reestablish position before it can resume processing.
- Commit points occur at regular intervals.

When a GC status code is returned, no data is retrieved or inserted. In your program, you can either:

- Issue a CHKP command, and resume database processing by reissuing the command that caused the GC status code.
- Ignore the GC status code and resume database processing by reissuing the command that caused the status code.

## The POS Command

---

## Chapter 7. Comparing Command-Level and Call-Level Programs

This chapter summarizes some of the differences between command- and call-level programs. If you are familiar with DL/I calls but not with EXEC DLI commands, this information will help you understand the commands and options you can use to perform the tasks that you performed using calls.

The following topics provide additional information:

- “DL/I Calls for IMS and CICS”
- “Comparing EXEC DLI Commands and DL/I Calls” on page 102
- “Comparing Command Codes and Options” on page 103

---

### DL/I Calls for IMS and CICS

Table 10 provides a quick reference for using DL/I calls in a Batch, Batch-Oriented BMP, or CICS with DBCTL environment.

Table 10. DL/I Calls Available to IMS and CICS Command-Level Application Programs

Request Type	Program Characteristics		
	Batch	Batch-Oriented BMP	CICS with DBCTL <sup>1</sup>
CHKP call (symbolic)	Yes	Yes	No
CHKP call (basic)	Yes	Yes	No
GSCD call <sup>2</sup>	Yes	No	No
INIT call	Yes	Yes	Yes
ISRT call (initial load)	Yes	No	No
ISRT call	Yes	Yes	Yes
LOG call	Yes	Yes	Yes
SCHD call	No	No	Yes
ROLB call	Yes	Yes	No
ROLL call	Yes	Yes	No
ROLS call (Roll Back to SETS) <sup>3</sup>	Yes	Yes	Yes
ROLS call (Roll Back to Commit)	Yes	Yes	Yes
SETS call <sup>3</sup>	Yes	Yes	Yes
STAT call <sup>4</sup>	Yes	Yes	Yes
TERM call	No	No	Yes
XRST call	Yes	Yes	No

**Notes:**

1. In a CICS remote DL/I environment, CALLs in the CICS-DBCTL column are supported if you are shipping a function to a remote CICS that uses DBCTL.
  2. GSCD is a Product-sensitive programming interface.
  3. SETS and ROLS calls are not valid when the PSB contains a DEDB.
  4. STAT is a Product-sensitive programming interface.
-

## Comparing Command-Level and Call-Level Programs

### Comparing EXEC DLI Commands and DL/I Calls

Table 11 compares EXEC DLI commands with DL/I calls and explains what the commands do. For example, in a command-level program, you use the LOAD command instead of the ISRT call to initially load a database.

Table 11. Comparing Call-Level and Command-Level Programs: Commands and Calls

Call-Level	Command-Level	Allows you to . . .
INIT call	ACCEPT command	Initialize for data availability status codes.
CHKP call (basic)	CHKP command	Issue a basic checkpoint.
DEQ call	DEQ command	Release segments retrieved using LOCKCLASS option or Q command code.
DLET call	DLET command	Delete segments from a database.
GU, GN, and GNP calls	GU, GN, and GNP commands <sup>1</sup>	Retrieve segments from a database.
GHU, GHN, and GHNP calls <sup>1</sup>	GU, GN, and GNP commands <sup>1</sup>	Retrieve segments from a database for updating.
GSCD call	GSCD call <sup>2</sup>	Retrieve system addresses.
ISRT call	ISRT command	Add segments to a database.
ISRT call	LOAD command	Initially load a database.
LOG call	LOG command	Write a message to the system log.
POS call	POS command	Retrieve positioning or space usage or positioning and space usage in a DEDB area.
INIT call	ACCEPT command	Initialize for data availability status.
INIT call	QUERY command	Obtain information of initial data availability.
INIT call	REFRESH command	Availability information after using a PCB.
REPL call	REPL command	Replace segments in a database.
XRST call	RETRIEVE command	Issue an extended restart.
ROLL or ROLB call	ROLL or ROLB command	Dynamically back out changes.
ROLS call	ROLS command	Back out to a previously set backout point.
PCB call	SCHD command	Schedule a PSB.
SETS call	SETS command	Set a backout point.
SETU call	SETU command	Set a backout point even if unsupported PCBs (like DEDBs or MSDBs) are present.
STAT call <sup>3</sup>	STAT command	Obtain system and buffer pool statistics.
CHKP call (extended)	SYMCHKP command	Issue a symbolic checkpoint.
TERM call	TERM command	Terminate a PSB.
XRST call	XRST command	Issue an extended restart.

#### Notes:

1. Get commands are just like Get Hold calls, and the performance of Get commands and Get calls is the same.
2. You can use the GSCD call in a batch command-level program. GSCD is a Product-sensitive programming interface.
3. STAT is a Product-sensitive programming interface.



## Comparing Command Codes and Options

Table 12 compares the options you use with EXEC DLI commands with the command codes you use with DL/I calls. For example, the LOCKED option performs the same function as a Q command code.

Table 12. Comparing Call-Level and Command-Level Programs: Command Codes and Options

Call- Level	Command-Level	Allows You to . . .
C	KEYS option	Use the concatenated key of a segment to identify the segment.
D	INTO or FROM specified on segment level to be retrieved or inserted.	Retrieve or insert a sequence of segments in a hierarchic path using only one request, instead of having to use a separate request for each segment. (Path call or command).
F	FIRST option	Back up to the first occurrence of a segment under its parent when searching for a particular segment occurrence. Disregarded for a root segment.
L	LAST option	Retrieve the last occurrence of a segment under its parent.
M	MOVENEXT option	Set a subset pointer to the segment following the current segment.
N	Leave out the SEGMENT option for segments you do not want replaced.	Designate segments you do not want replaced, when replacing segments after a get hold request. Usually used when replacing a path of segments.
P	SETPARENT	Set parentage at a higher level than what it usually is (the lowest hierarchic level of the request).
Q	LOCKCLASS, LOCKED	Reserve a segment so that other programs are not able to update it until you have finished processing it.
R	GETFIRST option	Retrieve the first segment in a subset.
S	SET option	Unconditionally set a subset pointer to the current segment.
U	No equivalent for command level programs.	Limit the search for a segment to the dependents of the segment occurrence on which position is established.
V	CURRENT option	Use the hierarchic level of and levels above the current position as qualifications for the segment.
W	SETCOND option	Conditionally set a subset pointer to the current segment.
Z	SETZERO option	Set a subset pointer to zero.
-	No command-level equivalent.	Null. Use an SSA in command code format without specifying the command code. Can be replaced during execution with the command codes you want.

## Comparing Command-Level and Call-Level Programs

---

## Chapter 8. Data Availability Enhancements

Your program might fail when it receives a status code indicating that a DL/I full-function database is unavailable. To avoid this, you can use data availability enhancements. After a PSB has been scheduled in DBCTL, your application program can issue requests to indicate to IMS that the program can handle data availability status codes and to obtain information about the availability of each database.

The following topics provide additional information:

- “Accepting Database Availability Status Codes”
- “Obtaining Information about Database Availability”

---

### Accepting Database Availability Status Codes

These status codes occur because PSB scheduling was completed without all of the referenced databases being available. Use the ACCEPT command to tell DBCTL to return a status code instead of abending the program:

```
EXEC DLI ACCEPT STATUSGROUP('A');
```

---

### Obtaining Information about Database Availability

You can put data availability status codes into each of the DB PCBs if:

- In a CICS DBCTL environment, by using the PSB scheduling request command, SCHED.
- In a Batch or BMP environment, at initialization time.

You can obtain the data availability status codes within the DL/I interface block (DIB) by using the following QUERY command:

```
EXEC DLI QUERY USING PCB(n);
```

n specifies the PCB.

The QUERY command is used after scheduling the PSB but before making the first database call. If the program has already issued a call using a DB PCB, then the QUERY command must follow the REFRESH command:

```
EXEC DLI REFRESH DBQUERY
```

The REFRESH command updates the information in the DIB. You can only issue this command one time.

For full-function databases, the DIBSTAT should contain NA, NU, TH, or blanks. For MSDBs and DEDBs, the DIBSTAT always contains blanks.

If a CICS command language translator has been used to translate the EXEC DLI commands, then, in addition to data availability status, the DBDNAME will be returned in the DIB field DIBDBDNM. Also, the name of the database organization will be returned in the DIB field DIBDBORG.

For an explanation of the data availability codes, see *IMS Version 9: Messages and Codes, Volume 1*.



---

## Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
1623-14, Shimotsuruma, Yamato-shi  
Kanagawa 242-8502 Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample

programs are provided "AS IS," without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. \_enter the year or years\_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming Interface Information

This section is intended to help the application programmer write IMS application programs. This book primarily documents General-use Programming Interface and Associated Guidance Information provided by IMS.

General-use programming interfaces allow the customer to write programs that obtain the services of IMS.

However, this book also documents Product-sensitive Programming Interface and Associated Guidance Information provided by IMS.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of IMS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Product-sensitive Programming Interface and Associated Guidance Information.

---

## Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web in the topic "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

The following terms are trademarks or registered trademarks of other companies, and have been used at least once in the IMS library:

- Adobe<sup>®</sup>, the Adobe logo, PostScript<sup>®</sup>, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
- Microsoft<sup>®</sup>, Windows<sup>®</sup>, Windows NT<sup>®</sup>, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.
- Linux<sup>®</sup> is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- UNIX<sup>®</sup> is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.



## Bibliography

This bibliography lists all of the information in the IMS Version 9 library.

*CICS Transaction Server for z/OS V2.3 CICS Application Programming Guide*, SC34-6231  
*CICS Transaction Server for z/OS V2.3 CICS Application Programming Reference*, SC34-6232  
*CICS Family: CICS TS for z/OS Communications from CICS on System/390®*, SC34-6031

---

### IMS Version 9 Library

Title	Acronym	Order number
<i>IMS Version 9: Administration Guide: Database Manager</i>	ADB	SC18-7806
<i>IMS Version 9: Administration Guide: System</i>	AS	SC18-7807
<i>IMS Version 9: Administration Guide: Transaction Manager</i>	ATM	SC18-7808
<i>IMS Version 9: Application Programming: Database Manager</i>	APDB	SC18-7809
<i>IMS Version 9: Application Programming: Design Guide</i>	APDG	SC18-7810
<i>IMS Version 9: Application Programming: EXEC DLI Commands for CICS and IMS</i>	APCICS	SC18-7811
<i>IMS Version 9: Application Programming: Transaction Manager</i>	APTM	SC18-7812
<i>IMS Version 9: Base Primitive Environment Guide and Reference</i>	BPE	SC18-7813
<i>IMS Version 9: Command Reference</i>	CR	SC18-7814
<i>IMS Version 9: Common Queue Server Guide and Reference</i>	CQS	SC18-7815
<i>IMS Version 9: Common Service Layer Guide and Reference</i>	CSL	SC18-7816
<i>IMS Version 9: Customization Guide</i>	CG	SC18-7817
<i>IMS Version 9: Database Recovery Control (DBRC) Guide and Reference</i>	DBRC	SC18-7818
<i>IMS Version 9: Diagnosis Guide and Reference</i>	DGR	LY37-3203
<i>IMS Version 9: Failure Analysis Structure Tables (FAST) for Dump Analysis</i>	FAST	LY37-3204
<i>IMS Version 9: IMS Connect Guide and Reference</i>	CT	SC18-9287

Title	Acronym	Order number
<i>IMS Version 9: IMS Java Guide and Reference</i>	JGR	SC18-7821
<i>IMS Version 9: Installation Volume 1: Installation Verification</i>	IIV	GC18-7822
<i>IMS Version 9: Installation Volume 2: System Definition and Tailoring</i>	ISDT	GC18-7823
<i>IMS Version 9: Master Index and Glossary</i>	MIG	SC18-7826
<i>IMS Version 9: Messages and Codes, Volume 1</i>	MC1	GC18-7827
<i>IMS Version 9: Messages and Codes, Volume 2</i>	MC2	GC18-7828
<i>IMS Version 9: Open Transaction Manager Access Guide and Reference</i>	OTMA	SC18-7829
<i>IMS Version 9: Operations Guide</i>	OG	SC18-7830
<i>IMS Version 9: Release Planning Guide</i>	RPG	GC17-7831
<i>IMS Version 9: Summary of Operator Commands</i>	SOC	SC18-7832
<i>IMS Version 9: Utilities Reference: Database and Transaction Manager</i>	URDBTM	SC18-7833
<i>IMS Version 9: Utilities Reference: System</i>	URS	SC18-7834

---

### Supplementary Publications

Title	Order number
<i>IMS TM Resource Adapter User's Guide and Reference</i>	SC19-1211
<i>IMS Version 9 Fact Sheet</i>	GC18-7697
<i>IMS Version 9: Licensed Program Specifications</i>	GC18-7825
<i>IRLM Messages and Codes</i>	GC19-2666

---

### Publication Collections

Title	Format	Order number
IMS Version 9 Softcopy Library	CD	LK3T-7213
IMS Favorites	CD	LK3T-7144
Licensed Bill of Forms (LBOF): IMS Version 9 Hardcopy and Softcopy Library	Hardcopy and CD	LBOF-7789

<b>Title</b>	<b>Format</b>	<b>Order number</b>
Unlicensed Bill of Forms (SBOF): IMS Version 9 Unlicensed Hardcopy Library	Hardcopy	SBOF-7790
OS/390 Collection	CD	SK2T-6700
z/OS Software Products Collection	CD	SK3T-4270
z/OS and Software Products DVD Collection	DVD	SK3T-4271

---

## **Accessibility Titles Cited in This Library**

<b>Title</b>	<b>Order number</b>
<i>z/OS V1R1.0 TSO Primer</i>	SA22-7787
<i>z/OS V1R5.0 TSO/E User's Guide</i>	SA22-7794
<i>z/OS V1R5.0 ISPF User's Guide, Volume 1</i>	SC34-4822

---

# Index

## A

abend, avoiding an 66  
abnormal termination 8  
ACCEPT command  
  description 66  
  example 67  
  format 66  
  options 67  
  system service command 66  
  usage 67  
adding  
  a segment sequentially 69  
  segments to a database 52  
adjustable character string 10  
AIB (Application Interface Block)  
  AIB mask 5  
  restrictions 5  
  supported commands 5  
AJ status code 97  
allowed commands, EXEC DLI 33  
alternate PCB 31  
AM status code 97  
AMODE(31) 29  
Application Interface Block (AIB)  
  AIB mask 5  
  restrictions 5  
  supported commands 5  
application programs, IFP 32  
application sample, subset pointers 90  
array, connected 10  
assembler language  
  DL/I command-level sample 12  
  I/O area 10  
assembler language program  
  DIB fields 6  
  variable names, mandatory 6  
automatic storage 22  
availability enhancements, data 105  
avoiding an abend 66

## B

BA status code 7  
backing out  
  changes dynamically 72  
  database changes 84, 85  
backout point  
  intermediate 85  
  setting 75  
  unconditionally setting 76  
basic checkpoint  
  description 67  
  issuing 84  
batch message processing (BMP) programs  
  issuing checkpoints 83  
  PCBs 32  
batch programs  
  command-level samples  
    assembler 12  
    C 23

batch programs (*continued*)  
  command-level samples (*continued*)  
    COBOL 16  
    PL/I 19  
  issuing checkpoints 83  
BC status code 7  
BILLING segment 4  
BKO execution parameter 72  
BMP (batch message processing) programs  
  issuing checkpoints 83  
  PCBs 32

## C

C code standard header file 28  
C program  
  DIB fields 6  
  DL/I command-level sample 23  
  variable names, mandatory 6  
call-level programs  
  comparing with command-level programs  
    command codes and options 103  
    commands and calls 102  
  DL/I calls available to IMS and CICS command-level 101  
changing the values of a segment's fields 59  
character string  
  adjustable 10  
  fixed-length 10  
checkpoint (CHKP)  
  command  
    description 67  
    description, in a Batch or BMP Program 84  
    example 68  
    format 67  
    issuing 2, 67  
    issuing in a Batch or BMP Program 84  
    options 67  
    restrictions 68  
    usage 67  
  EXEC DLI command 84  
    basic 84  
    current position 83  
    symbolic, description 84  
CHKP (Checkpoint)  
  command  
    description 67  
    description, in a Batch or BMP Program 84  
    example 68  
    format 67  
    issuing 2, 67  
    issuing in a Batch or BMP Program 84  
    options 67  
    restrictions 68  
    usage 67  
  EXEC DLI command 84  
    basic 84  
    current position 83  
    symbolic, description 84  
CHKPT=EOV parameter 67  
CICS  
  command language translator 29

- CICS (*continued*)
    - HANDLE ABEND command 8
    - Transaction Server 2
  - CMPAT option 32
  - COBOL
    - compiler 16
    - DL/I command-level sample 16
    - I/O area 9
    - II translator 29
    - OS/V5 19
    - program
      - DIB fields 6
      - variable names, mandatory 6
    - V4 19
  - command language translator, CICS 29
  - command-level programs
    - comparing with call-level programs
      - command codes and options 103
      - commands and calls 102
    - DIB (DL/I interface block) 5
    - DL/I calls available to IMS and CICS 101
    - I/O area, defining 9
    - key feedback area, defining 9
    - preparing EXEC DL/I program for execution 29
    - samples
      - assembler language 12
      - C 23
      - COBOL 16
      - PL/I 19
    - status codes, checking 6
    - syntax of EXEC DLI commands 33
  - command, path 62
  - commands
    - EXEC DLI
      - ACCEPT 66
      - CHKP (Checkpoint) 67
      - DEQ (Dequeue) 68
      - DLET (Delete) 35
      - GN (Get Next) 36
      - GNP (Get Next in Parent) 41
      - GU (Get Unique) 47
      - ISRT (Insert) 52
      - LOAD 69
      - LOG 70
      - POS (Position) 58
      - QUERY 70
      - REFRESH 71
      - REPL (Replace) 59
      - RETRIEVE 63
      - ROLB (Rollback) 72
      - ROLL 73
      - ROLS (Rollback to SETS or SETU) 73
      - SCHD (Schedule) 64
      - SETS (Set a Backout Point) 75
      - SETU (Set a Backout Point Unconditionally) 76
      - STAT (Statistics) 77
      - summary 34
      - SYMCHKP (Symbolic Checkpoint) 78
      - TERM (Terminate) 65
      - XRST (Extended Restart) 79
    - SCHD PSB 22
    - symbolic checkpoint 78
    - system service 66
  - commands allowed, EXEC DLI 33
  - commit database changes 2
  - committing your program's changes to a database 83
  - comparing EXEC DLI
    - commands with DL/I calls 102
    - options with command codes 103
  - compiler, COBOL 16
  - compiling, options with EXEC DLI 29
  - concatenated key, segment 9
  - connected array 10
  - crossing a unit of work (UOW) boundary when processing
    - DEDBs 99
  - current position in the database, determining the 63
- ## D
- data availability enhancements 105
  - data entry database
    - See* DEDB (data entry database)
  - database
    - administrator 3
    - availability
      - obtaining information 105
      - status codes, accepting 105
    - changes, committing 2
    - description name field, DIB (DL/I interface block) 8
    - determining the current position in the 63
    - establishing a starting position in a 47
    - example, medical hierarchy 2
    - integrity, maintaining 83
    - organization field, DIB (DL/I interface block) 9
    - processing, Fast Path 87
    - recovery 83
    - recovery, planning for
      - backing out database changes 84
      - checkpoints, CHKP command 84
      - checkpoints, taking 84
      - restarting your program, XRST command 84
    - types
      - GSAM 9
      - HDAM 9
      - HIDAM 9
      - HISAM 9
      - HSAM 9
      - INDEX 9
      - SHSAM 9
  - DB PCB
    - definition 31
    - specifying 74
  - DBA 3
  - DBCTL facilities
    - ACCEPT command 105
    - data availability 105
    - QUERY command 105
    - REFRESH command 105
    - ROLS (Roll Back to SETS or SETU) command 85
    - SETS (Set a Backout Point) command 85
  - DBD, defining subset pointers 90
  - DEDB (data entry database)
    - crossing a unit of work (UOW) boundary when
      - processing 99
    - processing
      - overview 87
      - POS (Position) command 97
      - subset pointers 87
  - defining application program elements to IMS
    - AIB 5
    - DIB 5
    - I/O area 9
    - key feedback area 9

- Delete (DLET) command
  - description 35
  - example 36
  - format 35
  - options 35
  - restrictions 36
  - usage 36
- dependent segments
  - retrieving
    - sequentially 41
    - the location of a 58
  - sequential
    - free space, identifying 99
    - in DEDBs 87
    - locating a specific dependent 98
    - locating the last inserted dependent 98
    - retrieving the location of the last one inserted 58
- dependents of a segment, removing 35
- DEQ (Dequeue) command
  - description 68
  - example 69
  - format 68
  - options 68
  - restrictions 69
  - usage 68
- Dequeue (DEQ) command
  - description 68
  - example 69
  - format 68
  - options 68
  - restrictions 69
  - usage 68
- design efficiency, programs 11
- determining the current position in the database 63
- DFHEIENT 15
- DFHEIRET 16
- DFHEISTG 15
- DIB (DL/I interface block)
  - accessing information 5
  - database description name field 8
  - database organization field 9
  - fields 6
  - information, obtaining the most recent 71
  - key feedback length field 8
  - label restriction 6
  - labels 6
  - segment level field 8
  - segment name field 8
  - status code field 6
  - structure 5
  - translator version 6
- differences between CICS and command-level batch or BMP programs 1
- direct dependent segments, in DEDBs 87
- DL/I
  - databases, read and update 1
  - setting a backout point 75
- DL/I interface block
  - See* DIB (DL/I interface block)
- DLET (Delete) command
  - description 35
  - example 36
  - format 35
  - options 35
  - restrictions 36
  - usage 36
- DLI option 29

- dynamic backout 84
- dynamically backing out changes 72, 73

## E

- efficient program design 11
- EIBREG parameter 15
- ending a logical unit of work 67, 78
- establishing a starting position in a database 47
- examples
  - ACCEPT command 67
  - CHKP (Checkpoint) command 68
  - DEQ (Dequeue) command 69
  - DLET (Delete) command 36
  - GN (Get Next) command 41
  - GNP (Get Next in Parent) command 46
  - GU (Get Unique) command 52
  - ISRT (Insert) command 56
  - LOAD command 70
  - LOG command 70
  - QUERY command 71
  - REFRESH command 72
  - REPL (Replace) 62
  - REPL (Replace) command 61
  - RETRIEVE command 64
  - ROLB (Rollback) command 72
  - ROLL command 73
  - ROLS (Rollback to SETS or SETU) command 74
  - SCHD (Schedule) command 65
  - SETS (Set a Backout Point) command 76
  - SETU (Set a Backout Point Unconditionally) command 77
  - STAT (Statistics) command 78
  - SYMCHKP (Symbolic Checkpoint) command 79
  - TERM (Terminate) command 65
  - XRST (Extended Restart) command 81
- EXEC DLI
  - allowable commands 33
  - commands
    - ACCEPT 66
    - CHKP (Checkpoint) 67
    - DEQ (Dequeue) 68
    - DLET (Delete) 35
    - GN (Get Next) 36
    - GNP (Get Next in Parent) 41
    - GU (Get Unique) 47
    - ISRT (Insert) 52
    - LOAD 69
    - LOG 70
    - POS (Position) 58
    - QUERY 70
    - REFRESH 71
    - REPL (Replace) 59
    - RETRIEVE 63
    - ROLB (Rollback) 72
    - ROLL 73
    - ROLS (Rollback to SETS or SETU) 73
    - SCHD (Schedule) 64
    - SETS (Set a Backout Point) 75
    - SETU (Set a Backout Point Unconditionally) 76
    - STAT (Statistics) 77
    - SYMCHKP (Symbolic Checkpoint) 78
    - TERM (Terminate) 65
    - XRST (Extended Restart) 79
  - compiler options, required 29
  - linkage editor options, required 29
  - options, using with subset pointers 90
  - preparing program for execution 29

EXEC DLI (*continued*)  
 program summary 34  
 recovery commands  
   CHKP (Checkpoint) 84  
   SYMCHKP (Symbolic Checkpoint) 84  
   XRST (Extended Restart) 84  
 syntax of commands 33  
 translator options, required 29  
 execution diagnostic facility 5  
 Extended Restart (XRST) command  
 description 79  
 example 81  
 format 80  
 options 80  
 restrictions 81  
 usage 81

## F

Fast Path  
 database, processing 87  
 P (position) processing option 99  
 subset pointers with DEDBs 87  
 FH status code 7  
 fields  
 changing the values of a segment's 59  
 in DIB 6  
 FIRST insert rule 56  
 fixed-length character string 10  
 formats  
 ACCEPT command 66  
 CHKP (Checkpoint) command 67  
 DEQ (Dequeue) command 68  
 DLET (Delete) command 35  
 GN (Get Next) command 36  
 GNP (Get Next in Parent) command 41  
 GU (Get Unique) command 47  
 ISRT (Insert) command 52  
 LOAD command 69  
 LOG command 70  
 POS (Position) command 58  
 PSB 33  
 QUERY command 70  
 REFRESH command 71  
 REPL (Replace) command 59  
 RETRIEVE command 63  
 ROLB (Rollback) command 72  
 ROLL command 73  
 ROLS (Rollback to SETS or SETU) command 74  
 SCHED (Schedule) command 64  
 SETS (Set a Backout Point) command 75  
 SETU (Set a Backout Point Unconditionally) command 76  
 STAT (Statistics) command 77  
 SYMCHKP (Symbolic Checkpoint) command 78  
 TERM (Terminate) command 65  
 XRST (Extended Restart) command 80  
 free space, identifying 99  
 FW status code 7

## G

GA status code 7  
 GB status code 7  
 GC status code 99  
 GD status code 7  
 GE status code 7, 23, 28

general programming guidelines 11  
 Get Next (GN) command  
 description 36  
 examples 41  
 format 36  
 options 38  
 restrictions 41  
 usage 40  
 Get Next in Parent (GNP) command  
 description 41  
 examples 46  
 format 41  
 options 43  
 restrictions 47  
 usage 46  
 Get Unique (GU) command  
 description 47  
 examples 52  
 format 47  
 options 48  
 restrictions 52  
 usage 51  
 GETFIRST option  
 examples 91  
 retrieving first segment in subset 91  
 getting IMS database statistics 77  
 GG status code 7  
 GK status code 7  
 GN (Get Next) command  
 description 36  
 examples 41  
 format 36  
 options 38  
 restrictions 41  
 usage 40  
 GNP (Get Next in Parent) command  
 description 41  
 examples 46  
 format 41  
 options 43  
 restrictions 47  
 usage 46  
 GSAM database 9  
 GSAM PCB 31  
 GU (Get Unique) command  
 description 47  
 examples 52  
 format 47  
 options 48  
 restrictions 52  
 usage 51  
 guidelines, general programming 11

## H

HANDLE ABEND command, CICS 8  
 HDAM database 9  
 HERE insert rule 56  
 HIDAM database 9  
 hierarchical database example, medical 2  
 HISAM database 9  
 HOUSHOLD segment 4  
 HSAM database 9

## I

- I/O area
  - assembler language 10
  - COBOL 9
  - coding 9
  - command-level program 9
  - DL/I 2
  - PL/I 10
  - restriction 9
  - symbolic CHKP 84
  - XRST 84
- I/O PCB 31
- IBM COBOL for z/OS & VM 19
- IFP application programs 32
- II status code 7
- ILLNESS segment 3
- IMS database statistics, obtaining 77
- INDEX database 9
- Insert (ISRT) command
  - description 52
  - examples 56
  - format 52
  - insert rules 56
  - options 53
  - restrictions 57
  - usage 56
- interface block, DL/I 2
- intermediate backout points 85
- ISRT (Insert) command
  - description 52
  - examples 56
  - format 52
  - insert rules 56
  - options 53
  - restrictions 57
  - usage 56
- issuing
  - a basic checkpoint 67
  - an extended restart 79
  - checkpoints in batch or BMP programs 83

## K

- key feedback area
  - command-level program 9
  - length field in DIB 8
- keyword, SYSSERVE 66

## L

- LAST insert rule 56
- last inserted sequential dependent segment, retrieving the
  - location of the 58
- LB status code 7
- level number field in DIB 8
- link editing, EXEC DLI 29
- linkage editor options with EXEC DLI 29
- LOAD command
  - description 69
  - example 70
  - format 69
  - options 69
  - usage 70
- locating
  - a specific sequential dependent 38
  - last inserted sequential dependent 98

- location of a dependent segment, retrieving the 58
- LOCKCLASS option 68
- LOG command
  - description 70
  - example 70
  - format 70
  - options 70
  - restrictions 70
  - usage 70
- logical unit of work, ending 67, 78

## M

- maintaining database integrity 83
- major structure 10
- medical database example
  - description 2
  - segments 2
- minor structure 10
- MOVENEXT option
  - examples 93
  - use when moving subset pointer forward 93
- moving subset pointer forward 93
- MPPs 32

## N

- NI status code 7

## O

- obtaining
  - IMS database statistics 77
  - recent information from the DIB 71
  - status code 70
- online programs, command-level samples
  - assembler 12
  - C 23
  - COBOL 16
  - PL/I 19
- options
  - ACCEPT command 67
  - CHKP (Checkpoint) command 67
  - CMPAT 32
  - DEQ (Dequeue) command 68
  - DLET (Delete) command 35
  - GN (Get Next) command 38
  - GNP (Get Next in Parent) command 43
  - GU (Get Unique) command 48
  - ISRT (Insert) command 53
  - LOAD command 69
  - LOCKCLASS 68
  - LOG command 70
  - MOVENEXT 93
  - P processing 99
  - POS (Position) command 58
  - QUERY command 71
  - REFRESH command 71
  - REPL (Replace) command 59
  - RETRIEVE command 63
  - ROLB (Rollback) command 72
  - ROLL command 73
  - ROLS (Rollback to SETS or SETU) command 74
  - SCHD (Schedule) command 64
  - SETS (Set a Backout Point) command 75
  - SETU (Set a Backout Point Unconditionally) command 76



- options (*continued*)
  - STAT (Statistics) command 77
  - SYMCHKP (Symbolic Checkpoint) command 78
  - TERM (Terminate) command 65
  - XRST (Extended Restart) command 80
- options for subset pointers
  - GETFIRST 91
  - MOVENEXT 93
  - SET 94
  - SETCOND 95
  - SETZERO 93
- OS/VIS COBOL 19
- overlap, storage 9
- overrides, PROCESS statement 29

## P

- P processing option 99
- parameters
  - BKO execution 72
  - CHKPT=EOV 67
  - EIBREG 15
  - RCREG 16
  - RULES 54, 56
- path command 62
- PATIENT segment 2
- PAYMENT segment 4
- PCB (program communication block)
  - alternate 31
  - in application programs, summary 32
  - types 31
- PL/I
  - DL/I command-level sample 19
  - I/O area 10
  - program variable names, mandatory 6
- pointers, subset
  - DBD, defining 90
  - description 87
  - GETFIRST option 91
  - MOVENEXT option 93
  - preparation for using 89
  - PSB, defining 90
  - sample application 90
  - SET option 94
  - SETCOND option 95
  - SETZERO option 93
  - status codes 97
- POS (Position) command
  - description 58
  - EXEC DLI command format 58
  - format 58
  - free space, identifying 99
  - locating a specific sequential dependent 98
  - locating the last inserted sequential dependent 98
  - options 58
  - restriction 59
  - usage 58
  - using with DEDBs 97
- Position (POS) command
  - description 58
  - EXEC DLI command format 58
  - format 58
  - free space, identifying 99
  - locating a specific sequential dependent 98
  - locating the last inserted sequential dependent 98
  - options 58
  - restriction 59

- Position (POS) command (*continued*)
  - usage 58
  - using with DEDBs 97
- position in the database, determining the current 63
- preparing programs
  - for EXEC DLI 11
  - for EXEC DLI execution 29
- PROCESS statement overrides 29
- processing
  - DEDBs 87
  - Fast Path, P (position) option 99
- program
  - design efficiency 11
  - entry 2
- programming guidelines, general 11
- programs, BMP 32
- PSB (program specification block)
  - defining subset pointers 90
  - format 33
  - in a CICS online program
    - scheduling a 64
    - terminating a 65
  - PCB, types of 31

## Q

- QUERY command
  - description 70
  - example 71
  - format 70
  - options 71
  - restrictions 71
  - usage 71

## R

- RCREG parameters 16
- recovering databases 83
- recovery EXEC DLI commands
  - basic CHKPT 84
  - SYMCHKP 84
  - XRST 84
- reentrance 15
- reentry 15
- REFRESH command 105
  - description 71
  - example 71
  - format 71
  - options 71
  - restrictions 72
  - usage 71
- releasing
  - a segment 68
  - resources 2
- removing a segment and its dependents 35
- REPL (Replace) command
  - description 59
  - examples 61, 62
  - format 59
  - options 59
  - restrictions 62
  - usage 60
- Replace (REPL) command
  - description 59
  - examples 61, 62
  - format 59



- Replace (REPL) command *(continued)*
    - options 59
    - restrictions 62
    - usage 60
  - replacing a segment 59
  - resetting a subset pointer 94
  - resources, releasing 2
  - restarting your program, with EXEC DLI XRST command 84
  - restrictions
    - AIB 5
    - CHKP (Checkpoint) command 68
    - DEQ (Dequeue) command 69
    - DIB label 6
    - DLET (Delete) command 36
    - GN (Get Next) command 41
    - GNP (Get Next in Parent) command 47
    - GU (Get Unique) command 52
    - I/O area 9
    - I/O area, PL/I 10
    - ISRT (Insert) command 57
    - LOG command 70
    - POS (Position) command 59
    - QUERY command 71
    - REFRESH command 72
    - REPL (Replace) command 62
    - RETRIEVE command 64
    - ROLB (Rollback) command 72
    - ROLL command 73
    - ROLS (Rollback to SETS or SETU) command 74, 75
    - SETS (Set a Backout Point) command 76
    - SETU (Set a Backout Point Unconditionally) command 77
    - SYMCHKP (Symbolic Checkpoint) command 79
    - XRST (Extended Restart) command 81
  - RETRIEVE command
    - description 63
    - example 64
    - format 63
    - options 63
    - restrictions 64
    - usage 63
  - retrieving
    - dependent segments sequentially 41
    - segments sequentially 36
    - specific segments 47
    - the location of a dependent segment 58
    - the location of the last inserted sequential dependent segment 58
  - returning a status code 66
  - ROLB (Rollback) command
    - description 72
    - example 72
    - format 72
    - options 72
    - restrictions 72
    - usage 72
  - ROLL command
    - description 73
    - example 73
    - format 73
    - options 73
    - restrictions 73
    - usage 73
  - Rollback (ROLB) command
    - description 72
    - example 72
    - format 72
    - options 72
  - Rollback (ROLB) command *(continued)*
    - restrictions 72
    - usage 72
  - Rollback to SETS or SETU (ROLS) command
    - description 73
    - examples 74
    - format 74
    - options 74
    - restrictions 75
    - usage 74
  - ROLS (Rollback to SETS or SETU) command
    - backout point, intermediate 85
    - description 73
    - examples 74
    - format 74
    - options 74
    - restrictions 75
    - usage 74
  - RULES parameter 54, 56
  - RULES= 56
- ## S
- sample application, subset pointers 90
  - sample programs, command level
    - assembler language 12
    - C 23
    - COBOL 16
    - PL/I 19
  - SCHD (Schedule) command
    - description 64
    - example 65
    - format 64
    - options 64
    - usage 65
  - SCHD PSB command 15, 22
  - Schedule (SCHD) command
    - description 64
    - example 65
    - format 64
    - options 64
    - usage 65
  - scheduling a PSB in a CICS online program 64
  - segment
    - adding one sequentially 69
    - adding to a database 52
    - and its dependents, removing 35
    - concatenated key 9
    - level number field 8
    - name field, DIB (DL/I interface block) 8
    - releasing a 68
    - replacing 59
    - retrieving sequentially 36
    - retrieving specific 47
    - sequential dependent
      - free space, identifying 99
      - in DEDBs 87
      - locating a specific dependent 98
      - locating the last inserted dependent 98
      - retrieving the location of the last one inserted 58
  - segments
    - in medical database example 2
  - sequential dependent segments
    - free space, identifying 99
    - in DEDBs 87
    - locating a specific dependent 98
    - locating the last inserted dependent 98

- sequential dependent segments (*continued*)
  - POS (Position) command 97
  - retrieving the location of the last one inserted 58
- sequentially retrieving
  - dependent segments 41
  - segments 36
- Set a Backout Point (SETS) command
  - description 75
  - example 76
  - format 75
  - options 75
  - restrictions 76
  - usage 75
- Set a Backout Point Unconditionally (SETU) command
  - description 76
  - example 77
  - formats 76
  - options 76
  - restrictions 77
  - usage 76
- SET option
  - examples 94
  - resetting a subset pointer 94
- SETCOND option
  - examples 95
  - setting a subset pointer conditionally 95
- SETS (Set a Backout Point) command
  - description 75
  - example 76
  - format 75
  - options 75
  - restrictions 76
  - usage 75
- setting a backout point
  - DL/I 75
  - unconditionally 76
- setting a subset pointer
  - conditionally 95
  - to zero 93
- SETU (Set a Backout Point Unconditionally) command
  - description 76
  - example 77
  - formats 76
  - options 76
  - restrictions 77
  - usage 76
- SETZERO option
  - examples 93
  - setting a subset pointer to zero 93
- SHSAM database 9
- specific segments, retrieving 47
- specifying the DB PCB 74
- standard header file, C code 28
- starting position in a database, establishing a 47
- STAT (Statistics) command
  - description 77
  - example 78
  - format 77
  - options 77
  - usage 78
- Statistics (STAT) command
  - description 77
  - example 78
  - format 77
  - options 77
  - usage 78
- statistics, obtaining IMS database 77
- status codes
  - AJ 97
  - AM 97
  - BA 7
  - BC 7
  - checking in a command-level program 6
  - FH 7
  - field in DIB 6
  - FW 7
  - GA 7
  - GB 7
  - GD 7
  - GE 7, 23, 28
  - GG 7
  - GK 7
  - II 7
  - LB 7
  - NI 7
  - obtaining 70
  - P processing option 99
  - returning a 66
  - subset pointers 97
  - TG 7
- storage overlap 9
- structure
  - major 10
  - minor 10
- subset pointers
  - DBD, defining 90
  - description 87
  - GETFIRST option 91
  - MOVENEXT option 93
  - moving forward 93
  - preparation for using 89
  - PSB, defining 90
  - resetting 94
  - sample application 90
  - SET option 94
  - SETCOND option 95
  - SETZERO option 93
  - status codes 97
- summary, EXEC DLI commands 34
- Symbolic Checkpoint (SYMCHKP) command
  - description 78
  - example 79
  - format 78
  - options 78
  - restart 84
  - restrictions 79
  - usage 78
  - XRST 84
- SYMCHKP (Symbolic Checkpoint) command
  - current position 79
  - description 78, 84
  - example 79
  - format 78
  - options 78
  - restart 84
  - restrictions 79
  - usage 78
  - XRST 84
- syntax diagram
  - how to read xi
- syntax of EXEC DLI commands 33
- SYSSERVE keyword 66
- system log, writing information to the 70

- system service
  - ACCEPT 66
  - CHKP (Checkpoint) 67
  - command summary 66
  - DEQ (Dequeue) 68
  - LOAD 69
  - LOG 70
  - QUERY 70
  - REFRESH 71
  - ROLB (Rollback) 72
  - ROLL 73
  - ROLS (Rollback to SETS or SETU) 73
  - SETS (Set a Backout Point) 75
  - SETU (Set a Backout Point Unconditionally) 76
  - STAT (Statistics) 77
  - SYMCHKP (Symbolic Checkpoint) 78
  - XRST (Extended Restart) 79

## T

- TERM (Terminate) command
  - description 65
  - example 65
  - format 65
  - options 65
  - usage 65
- Terminate (TERM) command
  - description 65
  - example 65
  - format 65
  - options 65
  - usage 65
- terminating a PSB in a CICS online program 65
- termination, abnormal 8
- TG status code 7
- Transaction Manager 33
- Transaction Server, CICS 5
- translating, EXEC DLI program 29
- translator
  - COBOL II 29
  - options required for EXEC DLI 29
  - version, DIB (DL/I interface block) 6
  - z/OS & VM 29
- TREATMNT segment 3

## U

- unconditionally setting a backout point 76
- unit of work (UOW)
  - crossing a boundary when processing DEDBs 99
  - ending a logical 67
- UOW (unit of work)
  - crossing a boundary when processing DEDBs 99
  - ending a logical 67
- usage
  - ACCEPT command 67
  - CHKP (Checkpoint) command 67
  - DEQ (Dequeue) command 68
  - DLET (Delete) command 36
  - GN (Get Next) command 40
  - GNP (Get Next in Parent) command 46
  - GU (Get Unique) command 51
  - ISRT (Insert) command 56
  - LOAD command 70
  - LOG command 70
  - POS (Position) command 58

- usage (*continued*)
  - QUERY command 71
  - REFRESH command 71
  - REPL (Replace) command 60
  - RETRIEVE command 63
  - ROLB (Rollback) command 72
  - ROLL command 73
  - ROLS (Rollback to SETS or SETU) command 74
  - SCHD (Schedule) command 65
  - SETS (Set a Backout Point) command 75
  - SETU (Set a Backout Point Unconditionally) command 76
  - STAT (Statistics) command 78
  - SYMCHKP (Symbolic Checkpoint) command 78
  - TERM (Terminate) command 65
  - XRST (Extended Restart) command 81

## V

- variable names, mandatory 6
- VS COBOL II 19

## W

- writing information to the system log 70

## X

- XRST (Extended Restart) command
  - description 79
  - example 81
  - format 80
  - options 80
  - restrictions 81
  - usage 81

## Z

- z/OS & VM 29
- z/OS & VM translator 29







Program Number: 5655-J38

Printed in USA

SC18-7811-03



Spine information:



IMS

**Application Programming: EXEC DLI Commands for  
CICS and IMS**

Version 9