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

Application Programming: CALL and RQDLI Interfaces

Version 1 Release 7

SH12-5411-06

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

Application Programming: CALL and RQDLI Interfaces

Version 1 Release 7

SH12-5411-06

Note !

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

Seventh Edition (December 2002)

This edition applies to Version 1, Release 7 (Version 1.7), of Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS), Program Number 5746-XX1, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

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

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

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

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

Contents

Figures

Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

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

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

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

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

Trademarks and Service Marks.

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

```
CICS
DB2
IBM
```

Preface

This manual describes how to write a DL/I DOS/VS application program using the DL/I call and request DL/I (RQDLI) interfaces. These application programs can be executed in the batch, multiple partition support (MPS), or online environments. The manual is intended for programmers, who code the application programs, and data base administrators, who design the applications.

It is assumed that you are familiar with DL/I to the extent that it is described in the *DL/I DOS/VS General Information* manual. It is also assumed that you are familiar with one of the four programming languages DL/I supports. These languages include: COBOL, PL/I, RPGII, and Assembler.

Dl/I is a data management control system that assists you in creating, accessing, and maintaining large common data bases. DL/I can be used in an online teleprocessing environment in conjunction with the Customer Information Control System (CICS/VS). Users of this manual should be familiar with the use of VSE and CICS/VS if DL/I is to be used in the online or MPS environment.

This manual contains five main sections:

1. An overall view of how a DL/I application program is written, including batch programming structure examples for each of the four languages.
2. A description of the call function codes and some general programming techniques and suggestions.
3. In the online environment, programming considerations are presented along with online coding examples for each of the four languages.
4. A description of optional DL/I programming functions, including such things as command codes, multiple positioning with DL/I calls, secondary indexing, and so on.
5. A Glossary of DL/I terms is also presented for your convience should you come across a term or phrase you do not understand.

Because of the special nature of the RPG II language, a subchapter dealing with RPG II specifics has been added to Chapter 1. Where applicable, RPG II references have been made in the text.

Related Publications:

DL/I VSE Release Guide, SC33-6211-05

DL/I DOS/VS Release Guide, SC33-6211-04

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

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

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

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

Other DL/I publications:

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

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

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

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

DL/I DOS/VS Guide for New Users, SH24-5001

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

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

Summary of Changes

Summary of Changes
for SH12-5411-6
Version 1.7

This edition has been revised to include information concerning the use of the MPS Restart Facility. Various additions, corrections, and improvements are also included.

Summary of Changes
for SH12-5411-5
Version 1.6

This edition has been revised to include information concerning the DL/I DOS/VS Boolean Qualification Statements, changes in the System and GSCD title of this manual and in the titles of other manuals produced for Version 1.6 of the DL/I DOS/VS library. Various additions, System and GSCD Calls corrections, and improvements are also included.

Boolean Qualification Statements
This support enables the application programmer to incorporate additional qualification into call statements for the selection of specific segments by specifying multiple qualification statements. The qualification statements can be logically related to each other by using the Boolean AND and OR operators between them.

Summary of Changes
for SH12-5411-4
as updated by SN24-5630
Version 1.5

This Technical Newsletter includes information concerning the DL/I DOS/VS field level sensitivity functional System and GSCD Calls enhancement, as well as various additions. corrections, and improvements.

Field Level Sensitivity
This feature makes it possible for the user to specify only those fields in the physical definition of a given segment that are to be included in the application's view of that segment, while remaining insenitive to the other fields in the segment.

Summary of Changes
for SH12-5411-4
Version 1.4

This edition has been revised to include the following system changes and DL/I DOS/VS functional enhancements. System and GSCD Calls

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

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

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

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

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

Performance Improvements
Two enhancements have been added to program isolation. They are the “O” procopt and MPS Batch Notification.

Chapter 1. DL/I Application Programming

Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS, hereafter referred to in this manual as DL/I) is a control system developed to help the user implement data base applications. It may be used in conjunction with the IBM Customer Information Control System/Virtual Storage (CICS/VS) to allow online access to data bases.

Data Base Concepts

A data base may be likened to a conventional file in that both consist of a named organized collection of data entered and maintained in one logical sequence for processing by application programs. However, when an application program processes records in a file, it must be tailored to the physical characteristics (block size, record length, access method, etc.) of the file. The application programmer must be aware of the format of all the fields in the record.

In DL/I, fields of data are grouped together in segments, and in turn segments may be grouped into data base records. DL/I application programs may reference these segments by name. A data base record may be made up of many different types of segments. A particular application program need be concerned only with those segments which contain the data needed for the application.

Data Base Characteristics

Figure 1-1 on page 1-2 shows a conventional data management physical record in a file with its elements: NAME, ADDRESS, and PAYROLL. The segments, as viewed by DL/I, are shown in the lower portion of this figure. They form a hierarchical data structure. The physical storage of the segments may differ significantly from the way the data is viewed as a data structure.

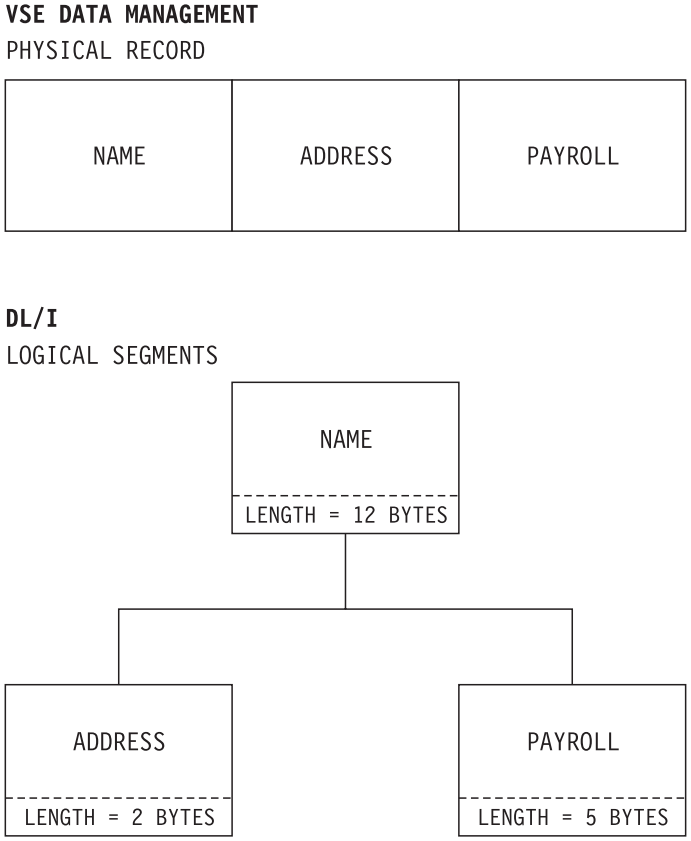

Figure 1-1. Physical Record - Segment Relationship (Example 1)

Each segment in the record usually contains several fields of data that are related and typically processed together.

Another example of a conventional data management physical record is shown in Figure 1-2 on page 1-3. Again, the lower portion of the figure shows the hierarchical data structure that DL/I makes available to the application programmer.

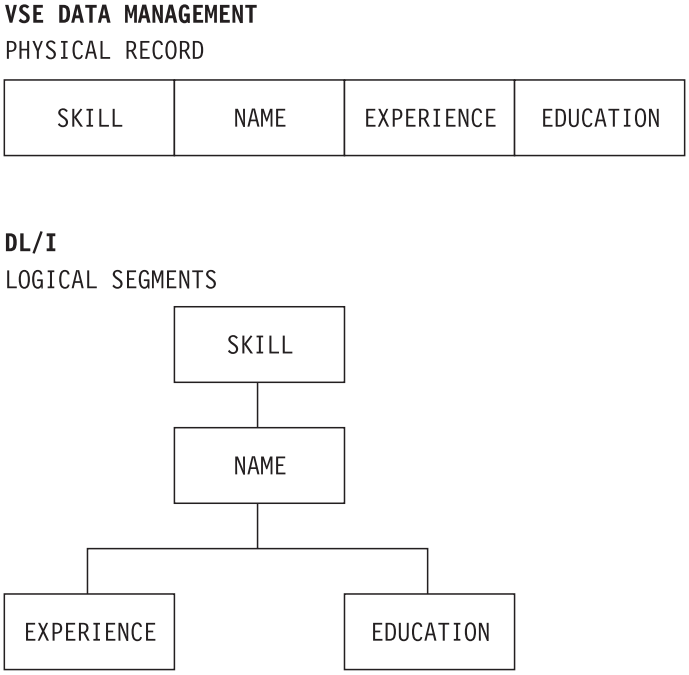

Figure 1-2. Physical Record - Segment Relationship (Example 2)

In the figure, assume that SKILL, NAME, EXPERIENCE, and EDUCATION are segments of a skills inventory data base. What Figure 1-2 does not show is that there can be multiple experience and education segments for each name, and many names for each skill. To expand upon the data structure, Figure 1-3 on page 1-4 shows a typical data base record within the skills inventory base. Notice that multiple name segments exist under the skill segment, multiple education segments exist under each name segment, and multiple experience segments exist under two of the name segments.

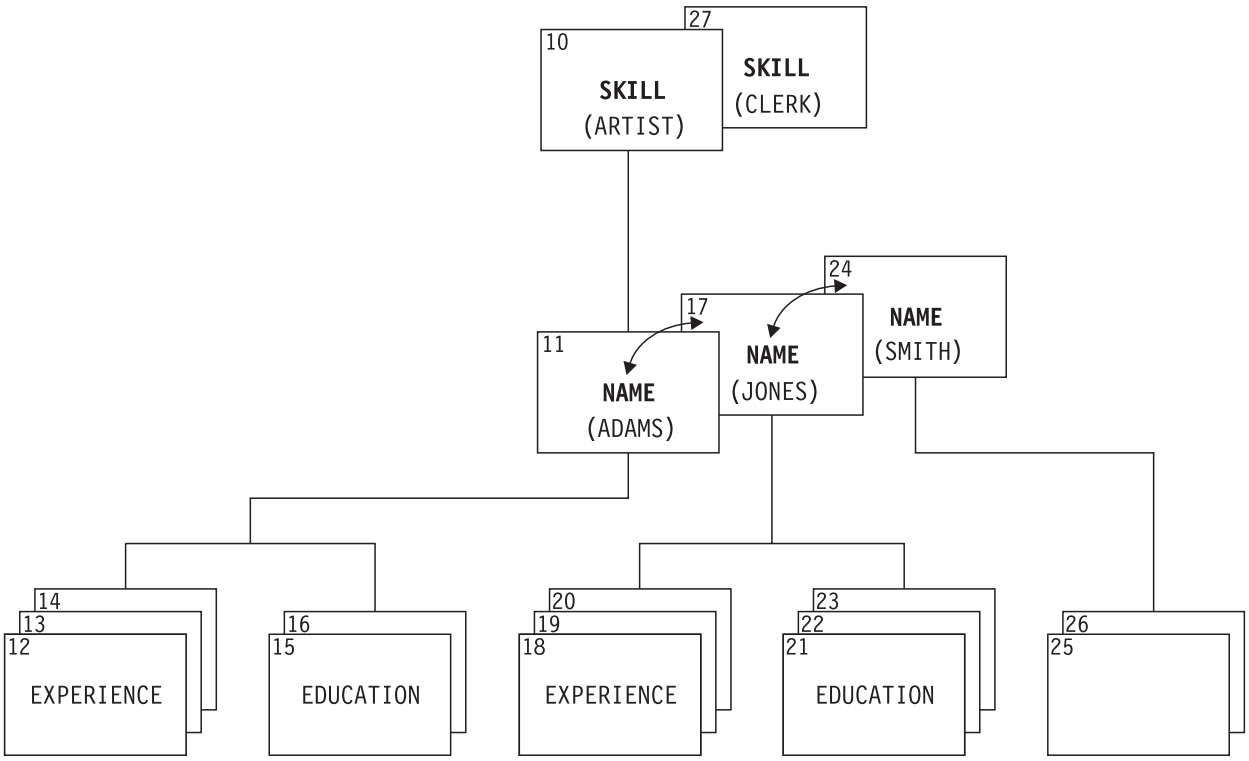

Figure 1-3. Expanded Data Base Structure

This leads to the following summary of the characteristics of a DL/I data base.

- A data base may contain a maximum of 255 segment types. (Only four are shown in Figure 1-2 on page 1-3: SKLILL, NAME, EXPERIENCE, EDUCATION.)
- A data base may contain a maximum of 15 hierarchical levels. (Figure 1-2 on page 1-3 shows three levels of hierarchy.)
- One root segment is allowed per data base record. (The root segment in Figure 1-2 on page 1-3 is SKILL.)
- The root segment always has a key field associated with it. The key field of the root controls the placement of the data base record in the data base.
- The lower-level segments in the hierarchy may also have key fields. These keys may be used to sequence multiple occurrences of the same type of segment within a data base record.
- Segments at lower levels are said to be dependent on those above. There may be 0 to n dependent child segments per parent. (In Figure 1-3, EXPERIENCE and EDUCATION are dependent segments of the parent segment NAME. NAME, EXPERIENCE, and EDUCATION segments are dependent segments of the parent segment SKILL, which is also the root segment.)
- The term parent describes the reverse relationship to that of the dependent. In Figure 1-3, SKILL is the parent of NAME, NAME is the parent of EXPERIENCE and EDUCATION.
- A data base record (such as shown in Figure 1-3) may consist of 1 to n segment occurrences.
- A data base may consist of 1 to n data base records.

- Segments within a hierarchical structure are always referenced in hierarchical sequence of top-to-bottom, left-to-right, front-to-back, as indicated by the numbers 10 through 27 in Figure 1-3.

Preparing To Use DL/I

DL/I is a program product that interfaces between the application program and the data base stored on auxiliary storage. This section explains the steps that must be taken before an application program can execute in conjunction with DL/I. Figure 1-4 shows the environment in which DL/I operates.

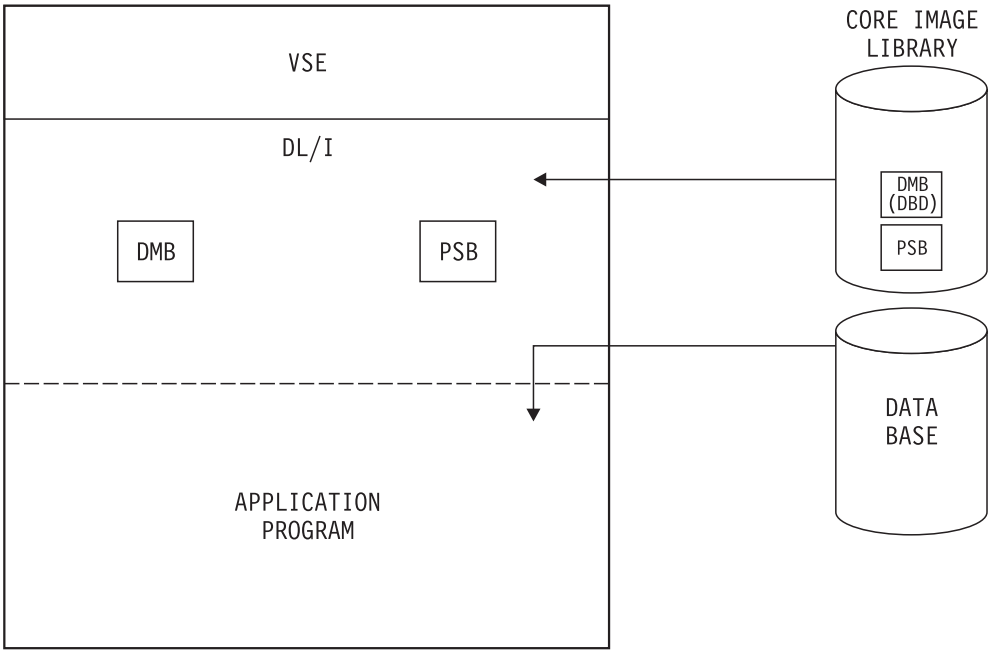

Figure 1-4. DL/I Environment

Building a Data Base Description (DBD)

The descriptive information about a data base record—the segment relationships and the physical device and data set description and access method used by DL/I—is stored apart from the data base and application program in a data base description (DBD) block. The DBD is a control block which is built using the DL/I data base description macro and stored in a core image library. A DBD for a particular data base must be completed before the data base can be created.

This operation must be done once for each data base and is the responsibility of the data base administrator. A complete description of DBD generation is given in *DL/I DOS/VS Resource Definition and Utilities*. *DL/I DOS/VS Interactive Resource Definition and Utilities* describes online DBD generation using the Interactive Macro Facility (IMF).

Building a Program Specification Block (PSB)

The PSB is a control block that is built using program specification block macros. The PSB identifies which types of segments from the data base record can be processed by a particular application program. The application program is then said to be sensitive to these segments. The PSB also identifies the type of processing to be performed (read only, update, load, etc.). This is referred to as the processing option (PROCOPT). The PROCOPT can refer to an entire data base or to a particular segment type. The PSB also contains an indicator that shows whether the program processes one or more data bases. One PSB is built for each application program. The definition of a PSB is the responsibility of the data base administrator. A complete description of PSB generation is given in *DL/I DOS/VS Resource Definition and Utilities*. *DL/I DOS/VS Interactive Resource Definition and Utilities* describes online PSB generation using the Interactive Macro Facility (IMF).

Other Preparatory Steps

The application program is written in the user-selected language (COBOL, PL/I, RPG II, or Assembler) and then compiled, link-edited, and cataloged into a core image library. For RPG II a translation step is required prior to compilation. One final DL/I-connected step must be completed by the data base administrator before the program can be executed under DL/I. Internal DL/I control blocks must be created for the previously generated PSB and related DBDs. The created control blocks are link-edited into a core image library ready for use by DL/I in conjunction with the application program.

Each VSAM file to be processed by DL/I must first be processed by Access Method Services with the specification DEFINE. This is a VSAM requirement. The application program may then be executed in a DL/I environment.

Figure 1-5 on page 1-7 depicts two environments. One is the conventional application program with its embedded file description and its direct use of VSE data management. The second is the DL/I environment. In this case, the DL/I control program loads both the application program to be executed and its associated PSB from a core image library. The PSB contains the PCB(s) to be used by the application program. The program communication block (PCB) provides specific areas used by DL/I to advise the application program of results of its calls, or RQDLI command in RPG II. The data management block (DMB) referenced by the PSB is also loaded from the core image library. The DMB describes all physical characteristics of a data base.

Restriction: Overlay structures are not supported for application programs executing under DL/I. Refer to “Restrictions” later in this chapter with regard to the use of the COBOL SORT verb, as its use produces an overlay structure.

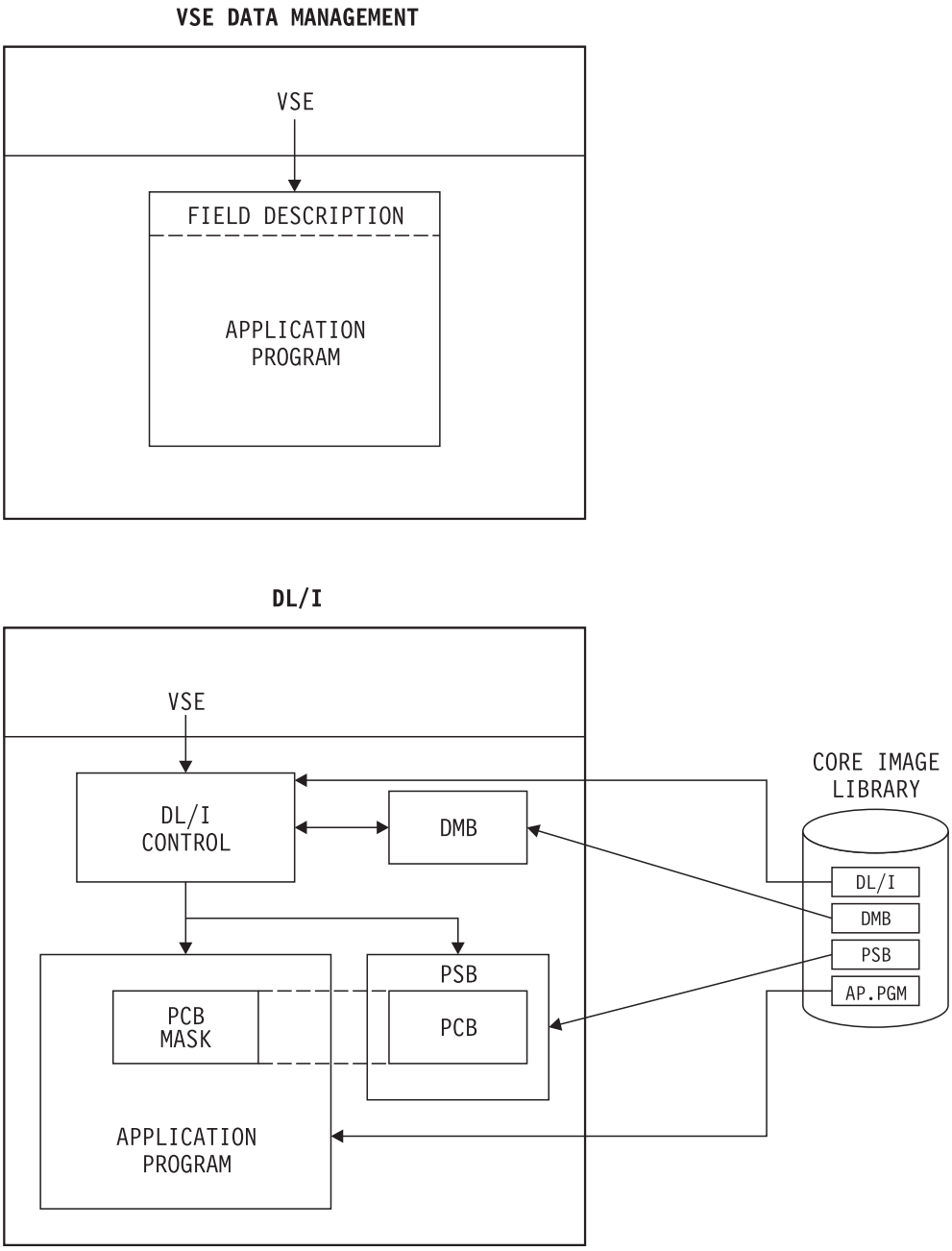

Figure 1-5. DL/I Batch Environment Compared to DOS/VSE

Coding Conventions

The following conventions are followed in illustrating the format and coding of control statements and the format of messages:

- The control statements are free form. Operation codes must begin after column one. Operands must follow an operation code or prior operand. The first operand must be separated from the operation code by at least one blank. Each operand must be separated from the previous operand by a comma with no intervening blanks. Operands may be continued on subsequent statements by placing a nonblank character in column 72, and by continuing in column 16 on the continuation statement.

- Uppercase letters, stand-alone numbers, and punctuation marks must be coded exactly as shown. The only exceptions to this convention are brackets [ ]; braces { }; ellipses ...; and subscripts. These are never coded.
- Lowercase letters and words and associated numbers represent variables for which the specific information or specific values must be substituted.
- The symbol ␣ is used to indicate one blank position.
- Items or groups of items within brackets [ ] are optional; they may be omitted if not required. Any item or group of items not within brackets must be coded.
- Stacked items, enclosed in braces { } represent alternative items. No more than one of the stacked items may be coded.
- If an alternative item is underlined, that item is implied: that is, DL/I automatically assumes that the underlined item is the choice if none of the items is coded.
- Ellipses, ..., indicate that the preceding item or group of items can be coded more than once in succession.

The following statement illustrates the coding conventions:

```
CALLDLI  {CBLTDLI},([prmcount,]function,pcb-name,
         {ASMTDLI}
                    i/o-area[,ssa...])
```

- CALLDLI must be coded, beginning after column one.
- From the two lines of parameters stacked within the braces ({CBLTDLI} and {ASMTDLI}), one must be chosen.
- The opening parenthesis must be coded.
- The parameters with the brackets ([prmcount] and [ssa...]) are optional.
- The closing parenthesis must be coded.

DL/I Application Program

The communication between the application program and DL/I takes place in three ways:

- On entry to and return from the application program.
- By defining a PCB mask in the application program for each data base accessed.
- By issuing DL/I calls; or RQDLI commands and, optionally, defining DB-files in RPG II.

Entry To An Application Program

In Figure 1-5 on page 1-7, when the system gives control to DL/I, the DL/I control program in turn passes control to the application program (through the entry point defined below). Register 1 contains an address which points to a list of the PCB names used by the application program. The PCB names in the entry statement must specified in the same sequence as specified in the PSB generation for the application program. The sequence of PCBs in the linkage section or declaration portion of the application program need not be the same as the sequence in the entry statement.

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

```
ENTRY 'DLITCBL' USING pcb-name-1, ..., pcb-name-n.
```

PL/I: The first statement of a PL/I program must be:

```
DLITPLI:  PROCEDURE (pcb-pointer-1,...,pcb-pointer-n)  OPTIONS(MAIN);
```

RPG II: In order to run an RPG II program using DL/I in batch mode, position 56 of the Header Specification must contain a “B.” If “B” is not specified, the Translator does not perform any translate functions. For the DL/I control program to establish addressability to the PCBs and pass control to the application program, an *ENTRY PLIST must be the first entry in the Calculation Specifications.

The Translator will automatically generate the *ENTRY PLIST for a main program if the programmer does not explicitly specifyit. However, the programmer must define all data bases as DB-files in the File Description Specifications with corresponding Continuation Lines (K-lines) specifying the PCBs. (For a detailed description of DB-files and PCB specification, see “DB (Data Base) File Definition” in this chapter.) The entry parameter list will contain a PARM statement for each PCB, ordered according to the integers ‘ij’ as specified by PCBij in the K-line. If the programmer chooses to specify the *ENTRY PLIST himself, the PCB names in the PARM statements must be in the same sequence as in the PSB generation for the program. The Translator will not check the contents of the list.

Assembler: The entry point to an Assembler language program that utilizes DL/I may have any desired name. However, when control is passed to the application program, register 1 contains the address of a variable-length fullword parameter list. Each word in this list contains a PCB control block address which must be saved by the application program. These addresses are in the same order as the PCB statements were specified during PSB generation. The 0 bit in the high-order byte of the last word in the parameter list should be reset to 0, unless only explicit calls using the count field are issued, in which case the 0 bit should be set to 1. The

addresses in this list are subsequently used by the application program when executing DL/I calls.

Register 15 contains the address of the application program entry point. Additionally, registers 14 through 12 must be stored on entry to the application program in an 18 fullword save area which is pointed to by register 13. Register 13 must then be set with the address of another 18 fullword save area prior to the issuance of the first DL/I call. Generally, this is performed during program initialization.

The following is an example of the intialization performed by an application program:

```
ANYNAME    CSECT
           USING    *,BASEREG
           SAVE     (14,12)
           LR       BASEREG,R15
           ST       R13,SAVEAREA+4
           LA       R13,SAVEAREA
           MVC      PCB,0(R1)
           .
           .
           .
SAVEAREA   DC       18F'0'
PARMLIST   DC       A(PARMCT)
FUNC       DC       A(DLIFUNC)
PCB        DC       A(0)
IOAREA     DC       A(DETSEGIN)
           DC       A(SSANAME)
           .
           .
           .
           END
```

Terminating the Application Program

At the completion of processing of any application program, control must be returned to DL/I as follows:

COBOL	PL/I	ASSEMBLER		RPG II
		L 13,4(13)	GET R13 FROM SAVE AREA	
GOBACK.	RETURN;	RETURN (14,12)	RETURN TO DL/I	SETON LR

The GOBACK or RETURN statement in a batch program returns control to DL/I. In RPG II control is returned to DL/I by setting on the Last Record (LR) Indicator, specified in the calculation specifications. After DL/I resources are released and the data bases are closed, DL/I subsequently returns control to VSE.

Notes:

1. The COBOL STOP RUN must not be used as control would not be returned to DL/I to allow it to release its resources and close the data bases and log.
2. The Assembler macros CANCEL, DUMP, JDUMP, and EOJ must not be used as control would not be returned to DL/I to allow it to release its resources and close the data bases and log.

Program Communication Block (PCB) Mask

The data base PCB provides specific areas used by DL/I to advise the application program of the results of its calls, or RQDLI commands in RPG II. At execution time, all PCB entries are controlled by DL/I. Accesses to PCB entries by the application program are for read-only purposes. The application program should not attempt to alter any of the values contained in the fields of the PCB.

As shown in Figure 1-6, the application program contains a mask of the PCB, but the real PCB exists outside the application program. All PCBs used by a particular application program are contained in a program specification block (PSB) associated with that application program. In a batch DL/I environment, there is a one-to-one relationship between PSBs and application programs; however, an application program may use more than one PCB for a data base. A PSB and the PCBs within it are created by the DL/I program specification block generation procedure. The PSB object module resulting from PSB generation is placed in a core image library.

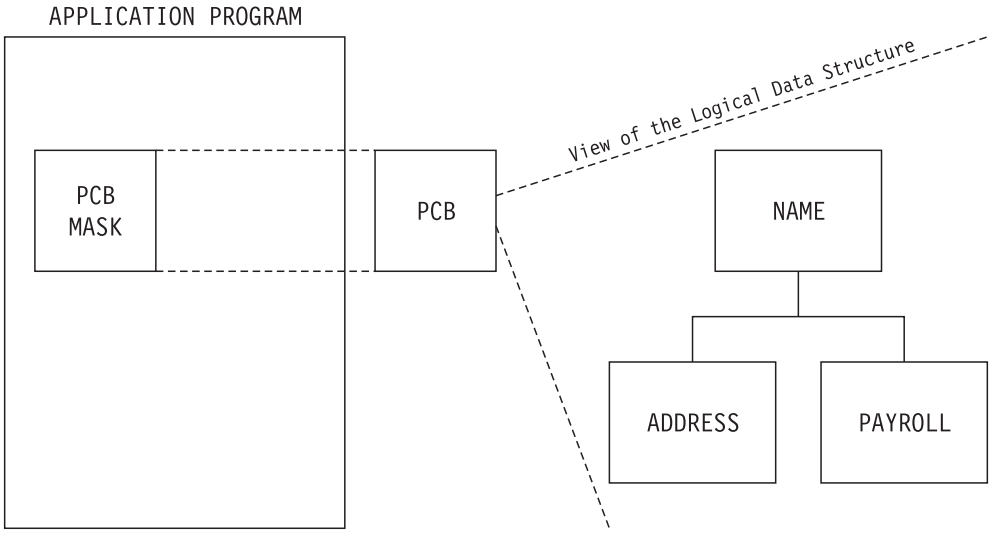

Figure 1-6. Application Program Data Base PCB Mask

The following examples, which refer to Figure 1-1 on page 1-2, illustrate the definition of the PCB mask in an application program. The numbers on the right of the examples refer to the notes that follow.

COBOL: In a COBOL program each PCB mask is set up in the Linkage Section.

```
                                                                    Notes
01 PCBNAME.                                                           1
 02 DBD-NAME        PICTURE  X(8).                                    2
 02 SEG-LEVEL       PICTURE  XX.                                      3
 02 STATUS-CODE     PICTURE  XX.                                      4
 02 PROC-OPTIONS    PICTURE  XXXX.                                    5
 02 RESERVE-DL/I    PICTURE  S9(5) COMPUTATIONAL.                     6
 02 SEG-NAME-FB     PICTURE  X(8).                                    7
 02 LENGTH-FB-KEY   PICTURE  S9(5) COMPUTATIONAL.                     8
 02 NUMB-SENS-SEGS  PICTURE  S9(5) COMPUTATIONAL.                     9
 02 KEY-FB-AREA     PICTURE  X(17).                                   10
```

PL/I: A unique pointer variable must be established for each PCB mask with the attribute POINTER. This pointer is used to establish addressability to the actual PCB data.

```
DECLARE      PCB_POINTER POINTER;
DECLARE 1    PCBNAME BASED(PCB_POINTER),                    1
             2 DBD_NAME CHAR(8),                            2
             2 SEG_LEVEL CHAR(2),                           3
             2 STATUS_CODE CHAR(2),                         4
             2 PROC_OPTIONS CHAR(4),                        5
             2 RESERVE_DLI FIXED BIN(31,0),                 6
             2 SEG_NAME_FB CHAR(8),                         7
             2 LENGTH_FB_KEY FIXED BIN(31,0),               8
             2 NUMB_SENS_SEGS FIXED BIN(31,0),              9
             2 KEY_FB_AREA CHAR(17);                        10
```

ASSEMBLER:

```
PCBNAME   DSECT                                             1
DBPCBDBD  DS     CL8    DBD NAME                            2
DBPCBLEV  DS     CL2    LEVEL FEEDBACK                      3
DBPCBSTC  DS     CL2    STATUS CODES                        4
DBPCBPRO  DS     CL4    PROC OPTIONS                        5
DBPCBRSV  DS     F      RESERVED                            6
DBPCBSFD  DS     CL8    SEGMENT NAME FEEDBACK               7
DBPCBMKL  DS     F      CURRENT LENGTH OF KEY               8
*                       FEEDBACK AREA
DBPCBNSS  DS     F      NO OF SENSITIVE SEGMTS              9
*                       IN PCB
DBPCBKFD  DS     0CL    KEY FEEDBACK AREA                   10
DBPCBNM   DS     0CL12  NAME KEY FEEDBACK
DBPCBNMA  DS     0CL14  ADDRESS KEY FEEDBACK
DBPCBNMP  DS     CL17   PAYROLL KEY FEEDBACK
```

RPG II: The PCB mask may be defined in either of two ways:

1. The following example (Figure 1-7 on page 1-13) shows a PCB structure which is automatically generated by the Translator if a K-line is specified for a DB-file. The numbers on the right refer to the notes that follow. Notes 6a, 8a, and 9a are given separately for this example.

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

   Note 6a: RESBij overlays RESRij
   Note 8a: KEYBij overlays KEYLij
   Note 9a: SSGBij overlays SSGNij

2. If no K-line is specified in the F-specs for a DB-file, the user has to define the proper PCB.

   The user defines a data structure, using names of his choice, but following precisely the layout of the automatically generated PCB.

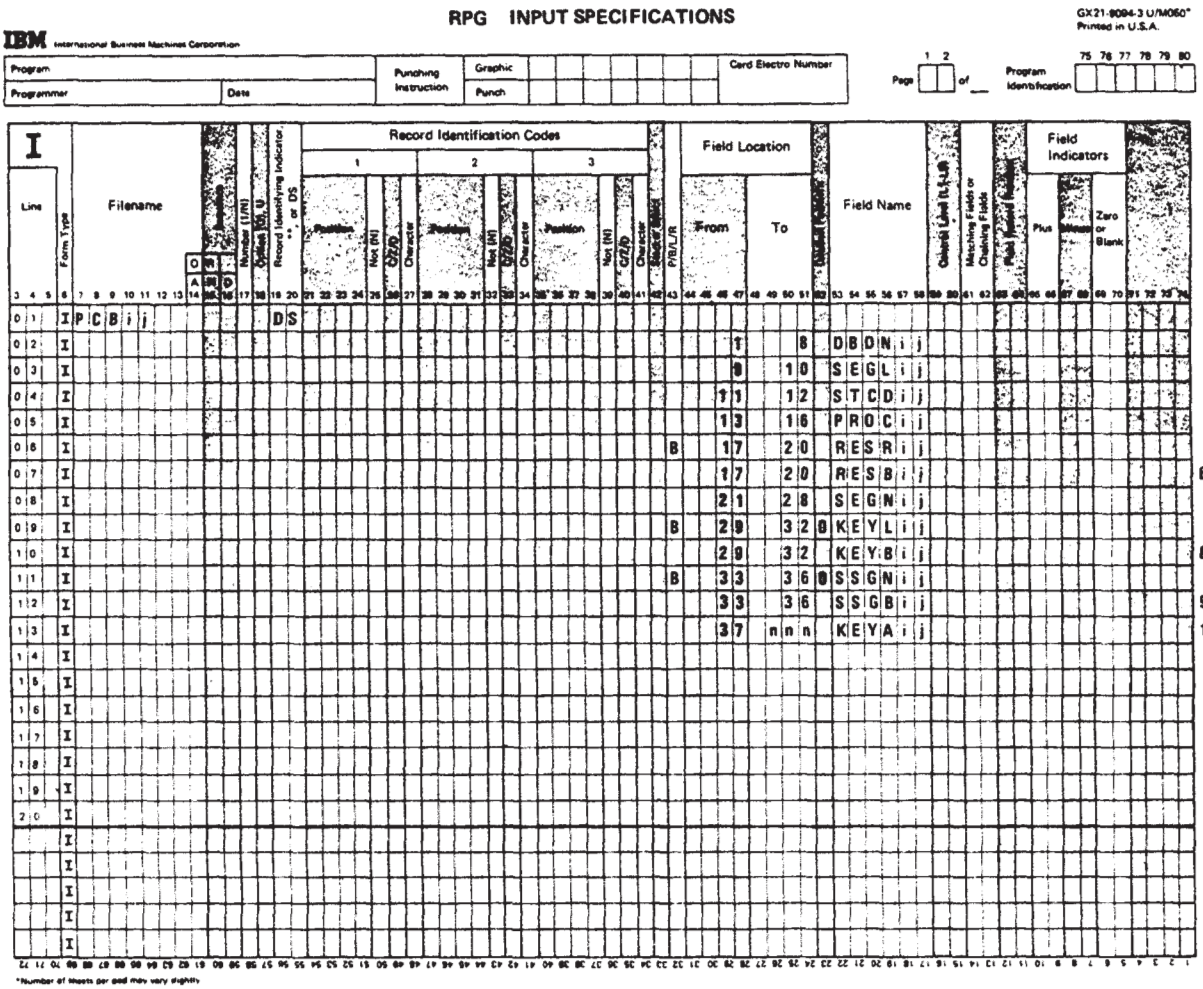

Figure 1-7. RPG II PCB Mask.

Notes:

1. The name of the PCB refers to the entire structure of PCB entries and is used in program statements. This name is not a field in the PCB. It is, for example, the 01 level name in the COBOL mask in the above examples. In RPG II, it is the data structure name of the PCB mask.
2. Name of data base description - This is the first field in the PCB and provides the DBD name associated with a particular data base. It contains character data and is 8 bytes long.
3. Segment hierarchy level indicator - DL/I loads this area with the level number of the lowest segment encountered in its attempt to satisfy a program request. When a retrieve operation is successfully completed, the level number of the retrieved segment is placed in this indicator. If retrieval is unsuccessful, the level number returned is that of the last segment, along the path to the desired segment, that satisfied the search criteria. This field contains numeric character data; it is 2 bytes long and is right-justified.
4. DL/I status code - A status code that indicates the results of a DL/I call is placed in this field and remains in the field until another DL/I call uses this PCB. (Specific status codes are discussed with their associated data base calls in Chapter 2 “DL/I Programming Reference Information.”) This field

contains 2 bytes of character data. When a successful call is executed, this field contains blanks or a warning status indication. This field should be tested immediately following each call, as discussed in the section “Status Code Summary” in Chapter 2.

5. DL/I processing options - This area contains a character code which tells DL/I the kinds of calls that may be used by the program for data base processing. This field is 4 bytes long. It is left-justified, with the remaining bytes filled with blanks.

   Possible values for the processing options are:

   G GET call.

   I INSERT call.

   R REPLACE call.

   D DELETE call.

   A All, includes the above four functions.

   P Required if command D is to be used on GET calls or INSERT calls. It must also be specified for those segments retrieved by path call even if their lowest level SSA has no D command code. It determines the maximum length of the I/O area and is used in conjunction with G, I, and A.

   E Exclusive use of the data base or segment. It is used in conjunction with G, I, R, D, and A.

   L Load function for data base loading (except HIDAM).

   LS Segments loaded in ascending sequence only. This load option is required for HIDAM, and for HSAM and simple HSAM if key fields are present in root segments.

   O Inhibits all program isolation queuing made under the PCB in which this PROCOPT appears. This option must be used with caution as the data retrieved may have just been updated by another user and is subject to being backed out if that user should abend. The O option must be used in conjunction with options G or GP. For further detail, see *DL/I DOS/VS Data Base Administration*.

   The L and LS values are mutually exclusive with G, I, R, D, and A, for a single PCB. A PROCOPT of G is implied when R or D is specified.

   The only options available for HSAM and simple HSAM data bases are G, L, and LS.

6. Reserved area for DL/I - DL/I uses ths 4-byte area for its own internal linkage related to an application program. The first byte is used as a flag byte. See “MPS (Multiple Partition Support) Considerations” in Chapter 3.

7. Segment name feedback area - DL/I fills this area with the name of the lowest segment encountered in its attempt to satisfy a call. When a retrieve operation is successful, the name of the retrieved segment is placed in this area. If retrieval is unsuccessful, the name returned is that of the last segment, along the path to the desired segment, that satisfied the search criteria. This field contains 8 bytes of character data. This field may be useful in GN and GNP calls. If the status code is AI (data management open error), the data set filename is returned into this area.

8. Length of key feedback area - This 4-byte binary entry specifies the current active length of the key feedback area described below. For restrictions on the contents of binary fields in RPG II, see *DOS/VS RPG II Language*.
9. Number of sensitive segments - This 4-byte binary entry specifies the number of segment types in the data base to which the application program is sensitive.
10. Key feedback area - DL/I places in this area the concatenated key of the lowest segment encountered in its attempt to satisfy a call. The area must be long enough to contain the maximum possible length of key fields along a hierarchical path. When a key operation is successful, the key field of the requested segment and the key field of each segment along the path to the requested segment are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key field and following the hierarchical path. When retrieval is unsuccessful, the key fields of all segments along the path to the requested segment for which search criteria were successful are placed in this area. Since data in the key feedback area is not cleared prior to a DL/I call, the length of the key feedback area should be used to obtain the valid key feedback data.

DL/I Batch Program Call

COBOL, PL/I and Assembler application programs communicate with DL/I using a program call. In RPG II, communication with DL/I is established by using an RQDLI (Request DL/I) command which is translated into a call statement by the Translator. Therefore, “call” in this manual will imply “RQDLI command” for RPG II applications, unless RQDLI is specifically mentioned.

Note: Because the syntax of RPG II is significantly different, RPG II is discussed separately. See “DL/I Application Program for RPG II” later in this chapter. An input/output call request is composed of a call statement with an argument list. The argument list provides the information, which is assembled by the application program, to describe a particular call function and the segment of data to be operated upon. One segment may be operated upon with a single input/output request (that is, one call statement).

One of the data fields in a segment is normally considered to be the key field. Each segment type has a fixed or variable length and a format definition.

Examples of DL/I program calls for PL/I, COBOL, RPG II, and Assembler appear in the “Program Examples” section of this chapter.

The general format of a DL/I call is as follows:

For COBOL

```
CALL 'CBLTDLI' USING[parm-count,] call-function, db-pcb-name,i/o-area
                          [,ssa...].
```

For PL/I

```
CALL PLITDLI (parm-count,call-function,db-pcb-name,i/o-area[,ssa...]);
```

For RPG II: See “DL/I Application Program for RPG II” later in this chapter.

For Assembler:

```
      {ASMTDLI}
CALL  {CBLTDLI}
```

In addition, register 13 must contain the address of an 18-fullword register save area provided by the application program. Register 1 must contain the address of the parameter list to be used with this call.

parm-count

The first parameter is the address of a 4-byte binary field containing the number of other parameters that are in the list. Note that the value in the count field does not include the count field itself. This parameter is required for PL/I and is optional for other languages.

call-function

The second parameter contains the address of a 4-character field that contains the DL/I code for the function to be performed. The application program can request DL/I to perform the following functions:

Meaning	Function Codes
GET UNIQUE	'GUbb'
GET NEXT	'GNbb'
GET NEXT WITHIN PARENT	'GNPb'
GET HOLD UNIQUE	'GHUb'
GET HOLD NEXT	'GHNb'
GET HOLD NEXT WITHIN PARENT	'GHNP'
INSERT	'ISRT'
DELETE	'DLET'
REPLACE	'REPL'
CHECKPOINT	'CHKP'

The use and meaning of the call functions are explained in Chapter 2.

db-pcb-name

The third parameter is the address of the program communication block (PCB) that is used for communication between DL/I and the application program. There is one PCB, which is contained within the PSB, for each data base being processed. More details of the use of the PCB are given in the section "PCB Mask" earlier in this chapter.

i/o-area

The I/O area address is the fourth parameter in the call statement. The I/O area is an area in the application program into which DL/I puts a requested segment or from which DL/I takes a designated segment, or, for the checkpoint call, from which it obtains the 8-character checkpoint identification. The area must be as long as the longest segment to be processed. If path calls are used, the area must be long enough to hold the largest concatenated segment; that is, all segments involved in each path call. The I/O area name points to the leftmost byte of the area. Segment data is always left-justified within a common area. Because of the structure of PL/I, i/o-area must be the name of a fixed-length character string, an area, a level 1 in a structure, or an array. If the user wishes to deal with substructures or elements of an array, he should use the DEFINED or BASED attribute.

COBOL Example:

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

PL/I Example:

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

Assembler Example:

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

The length of these work areas is 90 bytes.

ssa

The addresses of one or more segment search arguments (SSAs) are the final parameters in the call statement. When an application programmer requests DL/I to perform data base functions, it is frequently necessary for him to identify a particular segment by its name and the names of all parent segments along the hierarchical path leading to the desired segment. These values do not appear directly in the call statement argument provided to DL/I. Instead, an SSA name is given that points to an area in the user's program that contains the actual SSA values.

SSAs may be used with DL/I GET calls and, for path calls (see “Command Codes” in Chapter 4), with DL/I REPLACE or DELETE calls. They are required for DL/I INSERT calls.

The SSA may consist of three elements: the segment name, the command code, and a segment qualification statement. The segment name provides DL/I with enough information to define the type of segment. The command code is optional and provides specification of functional variations applicable to the call function. They are fully explained in Chapter 4. The segment qualification statement is optional and contains information that DL/I uses to test the value of the segment's key or data fields within the data base to determine whether the segment meets the user's specifications. Using this approach, DL/I performs the data base segment searching and the program need process only those segments in which it is interested.

A segment qualification statement is composed of three parts: a segment field name, a relational operator, and a comparative value. Boolean qualification may be performed by connecting qualification statements together with the AND and OR Boolean operators. Except where they are used to fill out a field, there must be no blanks in this statement. The complete qualification for each segment is contained between the left and right parentheses.

An *unqualified SSA* is built using a 9-byte area with the segment name occupying the leftmost 8 bytes and blank in position 9, as shown below.

Segment Name	b
8	1

When an unqualified SSA is used in a call statement, or an RQDLI command in RPG II, it indicatesto DL/I the type of segment to be processed. It does not identify a particular occurrence.

The second type of SSA is a *qualified SSA*. In this case, not only is the segment type identified, but a particular occurrence or group of occurrences is indicated. This is done by quoting the name of the key field for the segment and a search value which must be satisfied.

The structure of an SSA, with the length in bytes of each section, is shown below.

	(optional)			(optional)												
elements	Segment Name	Com- mand Codes		BOOLEAN STATEMENT →												
				Begin Qualif.	Qualification Statement # 1			Boolean Operator	Qualification Statement # 2			Boolean Operator	Qualification Statement # n			End Qualif.
contents	Name of Segment Type	•	Code Char- acters	'('	Field Name	RO	Compar. Value	'&' '•' or '+' '|'	Field Name	RO	Compar. Value	'&' '•' or '+' '|'	Field Name	RO	Compar. Value	')'
number of bytes	8	1	Var.	1	8	2	1 to 255	1	8	2	1 to 255	1	8	2	1 to 255	1

Segment Name: must be 8 bytes long, with rightmost (trailing) blanks to fill out the field as required. It is the segment name that pertains to a specific segment type in the hierarchical structure of a data base record and which is defined during data base generation.

Command Codes: The command codes are optional. They provide functional variations to be applied to the call for that segment type. An asterisk (*) following the segment name indicates the presence of one or more command codes. A blank or a left parenthesis is the ending delimiter for command codes.

Left Parenthesis, '(': indicates the beginning of a segment qualification statement. If the SSA is unqualified, the eight-byte segment name, or optional command codes must be followed by a blank.

Qualification Statement: The presence of a qualification statement is indicated by a left parenthesis following either the segment name or, if present, command codes. Each qualification statement consists of at least one field name, relational operator, and a comparative value. In case of a Boolean expression, multiple field qualifications would exist.

Segment Field Name: is the name of a segment field that appears in the description of that segment type in the DBD. The name is 8 characters long, with rightmost blanks as required to fill 8 bytes. The named field may be either the key field or a data field within a segment.

RO (Relational Operator): is a set of two characters that expresses the manner in which the contents of the field referred to by the segment field name are to be tested against the comparative value. The choice of the relational operator does not affect the starting point of the search or the order of the search.

Operator	Meaning
¢=	Equal to
=>	Equal to or greater than
=<	Equal to or less than
¢>	Greater than
¢<	Less than
¬=	Not equal to.

Note: The operator characters above may be used in any order (for example, => or >=).

Comparative Value: is the value against which the contents of the field referred to by the segment field name is to be tested. The length of this field must be equal to the length of the named field as defined in the DBD, that is, it includes leading or trailing blanks (for alphameric) or zeros (usually needed for numeric fields) as required. When using HDAM, do not use special characters because a program check may occur in the randomizing module. Further information on the randomizing module may be found in *DL/I DOS/VS Data Base Administration*.

Boolean Operator or Right Parenthesis, ')': Following the comparative value is either a Boolean operator, relating this qualification statement to the next qualification statement, or a right parenthesis as the ending delimiter indicating the last qualification statement for this segment.

Boolean logic qualifications can be performed on each segment by specifying multiple qualification statements. The qualification statements can be logically related to each other by using the Boolean *AND* and *OR* operators between them.

The logical *AND* is expressed by the EBCDIC character '&' or '*'. The logical *OR* is expressed by the EBCDID '+' or '|'.

All Boolean statements connected by *AND* operators are considered a set of qualification statements. An *OR* operator between two qualification statements begins a new set of qualification statements. A set can consist of one or more statements. To satisfy an SSA, a segment can satisfy any set of qualification statements. To satisfy any set, the segment must satisfy all statements within the set.

The qualification statement test is terminated as soon as a segment type that satisfies the qualification test is found in the data base. This procedure continues for all SSAs in a DL/I data base call, or RQDLI command in RPG II, until the desired segment is found.

Assume that the generic name of the skill segment in Figure 1-8 on page 1-20 is SKILLINV and its key field name is SKILCODE. Thus the SSA for a GET UNIQUE call for the skill segment with skill code equal to artist appears as:

```
SKILLINV(SKILCODE¢=ARTIST)
```

For retrieval or addition of a root segment, only one SSA must be provided. Normally the retrieval or insertion of a dependent segment requires that multiple SSAs be provided in the call request. Each SSA in the list describes a segment to which the segment to be operated upon is dependent. The SSAs for a given DL/I call must be in proper hierarchical relationship. Assume that the generic name of a name segment-type is NAME, its key field name is NAME, and there are employees with key field values of ADAMS, JONES, and SMITH whose parent is a skill segment having a key field value of ARTIST. Retrieval is accomplished by two segment search arguments with Boolean qualification on the second level included within the parameter list of the DL/I call, or RQDLI command in RPG II:

```
SKILLINV(SKILCODEb=ARTIST)
NAMEbbbb(NAME=bbbbb=ADAMS|NAMEbbbbbJONES)
```

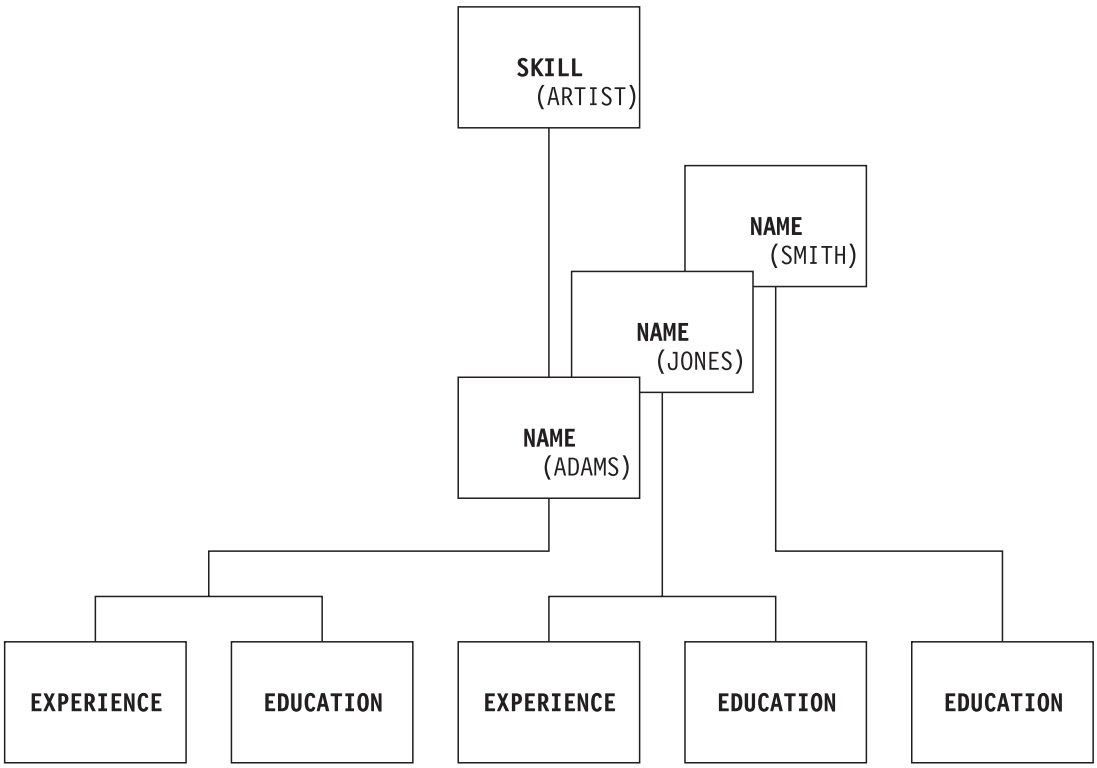

Figure 1-8. Logical Data Base Record Structure

DL/I Application Program for RPG II

Access to DL/I is provided in RPG II by means of RQDLI commands (Request DL/I) and, optionally,DB-files. The Translator translates the RQDLI commands into RPG II call statements and parameter lists and the DB-file specifications into File Description Specifications for SPECIAL files.

Note: The following syntax notation is used in the RPG II statement formats.

- | is used to separate alternatives, one of which has to be coded.
- (optional) is used to indicate that the construct is optional.
- uppercaseletters are used to indicate system-defined information.
- lowercase letters are used to indicate user-defined information.

RQDLI Commands For DB Access

The application program accesses a data base, which may be defined previously in the File Description Specifications, with the help of RQDLI commands, which have to be specified in the Calculation Specifications. An RQDLI command consists of an RQDLI statement followed by optional ELEM, USSA, and QSSA statements.

The format of the RQDLI statement is as follows:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | Ln | SR
9-17	see the publication, *DOS/VS RPG II Language*
18-27	func-name
28-32	RQDLI
33-42	file-name (optional)
43-55	blank
56-57	indicator
58-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

Note: No AN or OR lines are allowed with RQDLI commands.

func-name: The following function names may be used in an RQDLI statement:

•	GU	Get unique
	GHU	Get hold unique
•	GN	Get next
	GHN	Get hold next
•	GNP	Get next within parent
	GHNP	Get hold next within parent
•	DLET	Delete
•	REPL	Replace
•	ISRT	Insert: - load a new data base - add to an existing data base
•	PCB	Schedule a PSB
•	TERM	Release a PSB
•	CHKP	Establish a checkpoint

The use and meaning is the same as explained in “call function” in this chapter.

file-name: The file-name specifies the data base to be accessed. If no FROM|INTO option is explicitly specified in the RQDLI command, standard RPG data transfer will be used.

standard RPG data transfer: Extracting input fields from records, or building output records from fields. It is used if an RQDLI command requires a FROM or INTO option, which is not explicitly specified. In this case the I/O operation is executed in an RPG-like manner, namely using the record specification in the Input Specifications for input operations (that is, using the extract fields routine via READ statement instead of an explicit INTO option) or building the output record with the help of Output Specifications (that is, using the build lines routines via EXCPT instead of an explicit FROM option).

With an RQDLI command, only the first record is put out to the specified file; if more records are conditioned they will be ignored. In addition, the RQDLI command causes all E-records with indicators on to be put out to the corresponding non-DB files. The user must ensure that files are conditioned in accordance with the RPG II rules for update files (read before write). A user-written EXCPT causes output to only non-DB files, but DB files also must be conditioned so that no output is attempted before a read. For standard data transfer, an EXCPT is automatically generated.

Note: Using the RPG II standard data transfer for an input operation on a DL/I data base, a READ will be issued even if the "record not found" condition is encountered. That means that in any case the contents of the fields within the record will be initiated with the information at which xREC is pointing.

indicator: An indicator must be reserved for use by the Translator. The user may specify in the RQDLI command which indicator is to be used. If no indicator is specified, the Translator will use indicator 13. The indicator should not be tested since, on return from DL/I, the status is undefined.

An RQDLI statement may be followed by one or more ELEM, USSA, or QSSA statements. The ELEM statements specify the FROM—INTO option, the PCB option, and the SSA option. The SSAs can also be specified by USSA and QSSA statements, which allow the definition of an SSA in RPG-like format. The statements specifying the SSA list must be in the proper hierarchical sequence.

The CHKP RQDLI statement may be followed by ELEM statements specifying the CHKPID option and the PCB option. No other ELEM statements are allowed.

An ELEM statement for the CHKPID option has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	CHKPID
28-32	ELEM
33-42	literal (see note)
43-48	var-name (see note)
49-52	optional entries (see note below and the publication, *DOS/VS RPG II Language)*
53-59	blank
60-80	see the publication, *DOS/VS RPG II Language)*

Note: Entries in positions 33-42 and 43-52 are mutually exclusive. The checkpoint identification can be specified either in positions 33-42 as an alphameric literal (maximum length eight bytes) or in positions 43-48 as a variable referring to an eight byte field. If no checkpoint identification is specified, the file-name, if any, specified in the CHKP RQDLI statement is used as a default checkpoint identification and for the PCB option if it is not explicitly specified and a K-line for a PCB has been defined for the DB-file.

var-name: denotes the name of a variable that describes an RPG II field, array, array-element, or data structure.

An ELEM statement for the FROM|INTO option has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	FROM|INTO
28-32	ELEM
33-42	blank
43-48	var-name
53-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

If a FROM|INTO option is explicitly specified in an ELEM statement, the input/output request is executed using the specified area, ignoring any record definitions for the named DB-file in the Input or Output Specifications. If no FROM|INTO option is used with an RQDLI command, the record area optionally defined with the DB-file is loaded with the segment handled by the operation. The record area (corresponding to a data base segment) may be described in the Input or Output Specifications, depending on the requested function. The INTO option is used with input operations, and the FROM option is used with output operations.

An ELEM statement for the PCB option has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	PCB
28-32	ELEM
33-42	blank
43-48	var-name(see note)
49-52	optional entries (see the publication, *DOS/VS RPG II Language)*
53-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

The PCB option may be used to specify the PCB-address to which the RQDLI request is directed. If not specified, the PCB-address is derived from the filename specified with the RQDLI statement.

Statements for SSA Specification

There are two kinds of statements used to describe an SSA, which may be used intermixed; either the SSA-option or the SSA specification in RPG-like format. In addition, an SSALIST option together with an ELIST-command are provided for ease of use. (The physical makeup of the SSA is fully described in “DL/I Batch Program Call” in this chapter.)

SSA-option: The SSA is a var-name. It is the user's responsibility to define the proper format and to put the correct values into it together with delimiters.

Note: The format of the area has to correspond exactly to the requirements as specified for the SSA in “DL/I Batch Program Call.”

ELEM statements of this kind are characterized by the keyword SSA in factor 1 of an ELEM statement and have the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	SSA
28-32	ELEM
33-42	blank
43-48	var-name (see note)
49-52	optional entries (see the publication, *DOS/VS RPG II Language*)
53-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

The area referred to by var-name must describe the SSA with all required entries as defined under SSA in “DL/I Batch Program Call.”

Note: For USSA and QSSA statements, var-name must not be an array name.

SSA Specification in RPG-Like Format: (USSA and QSSA statement)

The statement contains all the relevant fields of an SSA in RPG-like format. The Translator maps these fields into the proper DL/I format. For details see the definition below.

USSA Statement

For an *unqualified SSA* it is only necessary to specify either the segment-name in quotes or a field containing the segment name in factor1 of the Calculation Specifications in a USSA statement.

The proper area is provided by the Translator, and the segment will be moved into it with the required blanks.

USSA statements for an unqualified SSA have the following format in the Calculation Specifications:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	segment-name
28-32	USSA
33-55	blank
56-57	command code (optional)
58-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

segment-name: Either var-name containing the name of a segment (up to 8 characters) or the name of a segment in apostrophes.

command code: One or two command codes may be specified. For a more detailed definition of command codes, see Chapter 4 “Command Codes.”

QSSA Statement

A QSSA statement for a *qualified SSA* has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	segment-name
28-32	QSSA
33-42	segment-field-name
43-48	comparative-value
49-51	length of segment-field
52	blank
53	blank
54-55	relational-operator
56-57	command-code (optional)
58-59	blank
60-80	see the publication, *OS/VS RPG II Language*

segment-name: As above with unqualified SSA.

segment-field-name: Name of the segmanet-field in apostrophes or var-name containing name of the segment-field (up to 8 characters). The length of the field as defined in the DBD is specified by positions 49-51.

comparative-value: Var-name containing the value against which the contents of the field referred to by the segment-field-name are to be tested. The length of the contents of var-name should correspond to that defined in positions 49-51. This information is used to generate the proper area. The length as specified must correspond to the actual length of the field defined by the segment field name in the DBD.

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

length: Length of the segment-field (in bytes) in the DBD.

position 52: A blank entry indicates that the field is alphameric. MOVEL is used to put the comparative value into the generated SSA (possibly padded with blanks to the right).

relational operator: The following relational operators may be used:

relational operator	meaning
EQ	equal to
GE	greater than or equal to
LE	less than or equal to
GT	greater than
LT	less than
NE	not equal to

command-code: One or two command codes may be specified for each SSA. For a more detailed definition, see Chapter 4 “Command Codes.”

SSALIST-Option

It is possible to specify in an ELEM statement the name of an SSA-list. This ELEM statement has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	SSALIST
28-32	ELEM
33-42	name-of-SSA-list
43-52	blank
53-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

The keyword SSALIST indicates that this statement stands for a list of statements defined elsewhere in an ELIST. The Translator will expand the SSALIST-option by the list of SSAs defined in the ELIST. The indicator in position 7-8 of the SSALIST option is appended to each SSA. As default, the indicator in position 7-8 of the RQDLI statement is used.

name-of-SSA-list: This name refers to the name of the ELIST defined in an ELIST statement.

ELIST-Command

The ELIST command defines the SSA list. The ELIST command consists of an ELIST statement immediately followed by one or more statements specifying SSAs. The ELIST statement has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	C
7-8	blank | SR | Ln
9-17	blank
18-27	name-of-SSA-list
28-32	ELIST
33-59	blank
60-80	see the publication, *DOS/VS RPG II Language*

The statements specifying SSAs must be specified in the proper hierarchical sequence. The format of the statements is the same as that used to describe the SSA directly in the RQDLI commands.

Restriction: The SSALIST-option must not be used in an ELIST command. Optionally, a DB-file may be specified to access DL/I.

DB (Data Base) File Definition

Each data base an application program wants to access may be defined in the File Description Specifications. The File Description Specifications for such a DB-file are only required if standard data transfer is intended for that DB-file and/or if use is made of the possibility of defining the PCB for a DB-file via a K-line in the File Description Specifications.

The File Description Specification for a DB-file has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language* F
7-14	file-name
15	I | U | O
16	D | blank
17-18	blank
19	F | blank
20-23	blank
24-27	maximum-segment-length
28-39	blank
40-46	DB
47-74	blank
75-80	see the publication, *DOS/VS RPG II Language*

file-name: The file-name can be freely chosen; it is the name by which the application refers to the data base.

maximum-segment-length: This length specifies the maximum length (in bytes) of the segments of the data base which the application is going to access. This length is used if no explicit FROM| INTO option is specified in an RQDLI command referencing the specific DB-file. In this case the segment has to be defined as a record in the Input or Output Specifications. If this length is omitted, a length of 80 is assumed.

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

Notes:

1. If position 19 is blank, it will default to F.
2. Output Specifications for DB-files must be of type E (position 15=E), exception records.

Additionally, for each DB File Description Specification, a continuation line may be specified which defines the corresponding PCB. The continuation line has the following format:

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	F
7-23	blank
24-27	pcb-key-length (optional)
28-50	blank
51-52	blank
53	K
54-59	PCB
60-65	PCBij
66-74	blank
75-80	see the publication, *DOS/VS RPG II Language*

PCBij: This defines the program communication block (PCB) connected with the DB file. ij ... establishes the relationship to the ordering of the PCBs in the PSB. ij defines this data base PCB as the element ij of the ordered list of PCBs. This ordering is used when the addressability of PCBs is established; ij may range between 01 and 99.

pcb-key-length: This integer specifies the length (less than or equal to 256) of the field in the data structure defining the PCB. If a K-line is specified, the Translator automatically generates the definition of the data structure for the PCB and puts it into the Input Specifications, with the names of the fields qualified by ij. The general format and the naming conventions can be seen in Figure 1-7 on page 1-13 in “Program Communication Block Mask,” in this chapter. If the K-lines for several DB Files define the same PCBij name, only the first causes the PCB data structure to be generated. The others are ignored and a warning message is issued. However, when these file names are specified in RQDLI statements, this PCBij name is used as the default value for the PCB option.

If no K-line is specified, it is the user's responsibility to define the proper PCB. For more detailed information, see “Program Communication Block Mask,” in this chapter.

Note: With the automatic generation of the PCB data structure, name clashes with user-defined field names may occur.

The user should never write into PCB fields.

Data Base Processing

The contents of this section assume that the programming ground rules have been established. This section deals with the processing of the segments of the data base(s) used by an application.

Data Base Loading

A data base is loaded by a user-written application program issuing DL/I calls, or RQDLI commands in RPG II, to insert data base records presorted by the key field of the root segment. This is a requirement of simple HSAM, HSAM, simple HISAM, HISAM, HIDAM, and HD primary index data bases. In an HSAM, HISAM, HIDAM, or HD primary index data base, when a data base record is composed of more than the root segment, all segments within the data base record must be presorted by their hierarchical relationship and key field value and must be inserted in correct hierarchical order. An HDAM or HD primary randomized data base can be created from sorted or unsorted data base records.

The PROCOPT entry in the PSB for this program must be L or LS. The only DL/I call which can be issued in such a program is ISRT.

Consider the process of inserting the segments of a skill inventory data base record shown in Figure 1-8 on page 1-20 earlier in this chapter. First, the skill (root) segment ARTIST is inserted. The name segment for ADAMS is inserted next. Then the experience segment of ADAMS is inserted, followed by the education segment of ADAMS. This continues with the name segment JONES, its experience segment and education segment, then name segment SMITH and its education segment. If this data base record represented the segments of data associated with skill X, the segments to be inserted into the data base next would be those associated with skill X + 1.

The INSERT function is used to create or load (recreate or reorganize) a data base. Prior to the execution of a DL/I call to insert a segment, the segment to be inserted must be moved into an I/O area and the proper list of SSAs must be provided. For RPG II, an output DB-file and standard RPG data transfer may be used.

Assume that, for creating the skill inventory data base, the segments of data associated with ARTIST are to be loaded. The first four segments to be loaded would be SKILL, NAME (ADAMS), EXPERIENCE (ADAMS), and EDUCATION (ADAMS). For example, the associatedSSAs and I/O area contents for the third DL/I INSERT call is as follows:

Experience Segment Insertion

```
[SSA1         -   SKILLINV(SKILCODE♭=ARTIST)]
[SSA2         -   NAME♭♭♭♭(NAME♭♭♭♭♭=ADAMS) ]
 SSA3         -   EXPERIEN♭

                  ┌──────────────────────────────────────────┐
I/O Area      -   │ Experience Code │ Data Field │ Data Field │
                  └──────────────────────────────────────────┘
```

The SSAs of a DL/I call, or RQDLI command in RPG II, for inserting a segment into a data base may describe the complete hierarchical path to the segment. It is not necessary , however, to describe the complete path. When no SSA is specified,

DL/I loads the next segment into the position indicated by the position pointer (see "Position Pointer" in Chapter 2). When creating a data base, therefore, it is only necessary to supply the segment name of the segment being inserted. Notice that the last segment search argument within each INSERT call does not and must not include the qualification statement portion. The qualification information is taken from the image of the segment in the I/O area.

Data Base Retrievals

The retrieval of segments within a data base is accomplished by the three GET call functions: GET UNIQUE, GET NEXT, and GET NEXT WITHIN PARENT. GET UNIQUE provides for the retrieval of a specific segment by direct reference into the data base. GET NEXT provides for sequential segment retrieval. Usually the GET NEXT function is used after a GET UNIQUE call or a GET NEXT call that has provided positioning to a unique segment within the data base. However, a GET NEXT may be used without positioning by a previous GET UNIQUE or GET NEXT. If no position has been established within a data base when a GET NEXT call is issued, the request is satisfied by proceeding from the beginning of the data base.

GET NEXT WITHIN PARENT allows sequential retrieval of all segments subordinate to a parent segment. Using Figure 1-8 on page 1-20, an example would be to retrieve all experience and education segments within the skill inventory data base for a given skill code and employee name. The parent segment is a unique name segment, and parentage must have been previously established with a GET UNIQUE or GET NEXT request. Once all the experience and education segments for a given skill code and employee name have been retrieved by a succession of GET NEXT WITHIN PARENT requests, a further attempt will result in a status code of GE being returned to the application program. GE indicates that the end of subordinate segments for the particular skill code and employee number has been reached.

In addition to direct retrieval of a unique segment and sequential retrieval of segments, an ability to sequentially skip from one segment to another of a common type is provided. Assume that it becomes necessary to retrieve all name segments within a particular skill segment. However, it is not necessary to retrieve the segments subordinate to each name segment (that is, experience and education data segments). The first name segment is retrieved with a GET UNIQUE request. The SSAs are:

```
SSA1    -   SKILLINV(SKILCODEb=ARTIST)
SSA2    -   NAMEbbbbbb
```

By changing the function to GET NEXT and repeating the above SSAs, all NAME segments whose skill is ARTIST are retrieved. A status code of GE is returned if a further attempt is made to retrieve when there are no more NAME segments for ARTIST.

Data Base Updates

The updating of data within a segment of a data base is performed through the REPLACE function. Before a DL/I call, or RQDLI command in RPG II, to replace a segment may be executed, the segment to be updated must be retrieved through a call with a GET function. The GET functions which may be specified are those previously discussed, but must include the addition of a HOLD definition (GET HOLD UNIQUE, GET HOLD NEXT, and GET HOLD NEXT WITHIN PARENT). The REPLACE function must then be executed in the *next* call by this program to

the data base. Any intervening calls to the same data base, using the same PCB, by this program cause the rejection of the subsequent REPLACE call. SSAs should not appear with the REPLACE function unless command codes are being used. The key field of the segment to be updated through the REPLACE function *must not* be modified.

The following PL/I example shows the calls necessary to change the segments of ARTIST from COMMERCIAL to COMMERCIAL CARTOON:

The first PL/I call statement is:

```
CALL PLITDLI (COUNT4,FUNCTION_GHU,DB_PCB,WORK_AREA,SSA1);
```

where:

```
SSA1 is SKILLINV(SKILCODEb=ARTIST)
```

```
                    ┌----------------------┐
I/O Area is then    | ARTIST | COMMERCIAL |
                    └----------------------┘
```

The program modifies the I/O area as follows:

```
                   ┌-----------------------------┐
I/O Area is now    | ARTIST | COMMERCIAL CARTOON |
                   └-----------------------------┘
```

The second PL/I call statement is:

```
CALL PLITDLI (COUNT3,FUNCTION_REPL,DB_PCB,WORK_AREA);
```

With this call statement the SKILL segment is taken from the I/O area and placed back in the data base.

For HSAM, updating can only be done by copying the data base from the input file (DD1 in DBD generation) to the output file (DD2), omitting, modifying, or inserting segments as required. The PSB for such a program requires two PCBs, one with PROCOPT=G for reading and one with PROCOPT=L or LS for inserting.

Data Base Deletions

The deletion of an entire segment within a data base is performed through the DELETE function. Before a DL/I call, or RQDLI command in RPG II, to delete a segment may be executed, the segment to be deleted must be retrieved using one of the GET HOLD calls. The DELETE function must be executed as the *next* call to the data base and the same PCB; otherwise, the DELETE function is rejected. The deletionof a parent segment causes deletion of all segments physically subordinate to the deleted segment.

The following is an example of how to delete the skill segment (both key and data fields) of ARTIST:

The first PL/I call statement is:

```
CALL PLITDLI (COUNT4,FUNCTION_GHU,DB_PCB,WORK_AREA,SSA1);
```

where:

```
SSA1 is SKILLINV(SKILCODE=ARTIST)

I/O Area then contains  ┌-----------------------------┐
                        | ARTIST | COMMERCIAL CARTOON |
                        └-----------------------------┘
```

The second PL/I call statement is:

```
CALL PLITDLI (COUNT3,FUNCTION_DLET,DB_PCB,WORK_AREA);

I/O Area still contains  ┌-----------------------------┐
                         | ARTIST | COMMERCIAL CARTOON |
                         └-----------------------------┘
```

As a result, the ARTIST segment and its dependent segments are deleted. That is, name segment (ADAMS), experience segment (ADAMS), and education segment (ADAMS) are deleted as well as all other name, experience, and education segments under this root segment.

If another GET UNIQUE call is made to this particular skill segment immediately after its deletion, a status code of GE (not found) is returned, but the I/O area, if not blanked out, still contains the above data.

Data Base Insertions

The addition or insertion of a new segment (all fields) into an existing data base is performed through the INSERT function. The techniques used for performing an INSERT function to add a segment to an existing data base are identical to those used with the INSERT function when creating a new data base. Remember that the addition of a dependent level segment is not permitted unless all parent segments in the complete hierarchical path already exist in the data base. An example referring to figref refid=ldbr. would be the addition of an experience segment subordinate to a particular name segment. The name segment must already exist in the data base or be added before any experience segment subordinate to that name segment may be added, otherwise a GE (not found) status code is returned. (See Chapter 2 for rules governing the INSERT call.)

Data Base Checkpoint

The CHECKPOINT function is used to establish a point of reference, during the execution of your program, which indicates all data base operations prior to this point were completed satisfactorily. A checkpoint can be issued at any appropriate points in your program as determined by you. It can be issued in batch, MPS batch, or online tasks.

In case of a failure in the batch environment, the backout utility may be run to back out data base changes occurring since the last checkpoint. For MPS batch and online tasks, CICS/VS dynamic transaction backout will automatically back out data base changes since the last checkpoint, when the CICS/VS journal is being used.

In order to provide a restart capability for MPS batch users, DL/I supports the use of VSE checkpoint/restart in conjunction with the DL/I checkpoint facility. This feature is only available to MPS batch programs running with MPS Restart active. For more information about the MPS Restart Facility, see the *DL/I DOS/VS Recovery/Restart Guide*.

Program Examples

COBOL Batch Program Structure

Figure 1-9 illustrates in outline form the fundamental parts in the structure of a COBOL batch program which, in this example, is to retrieve data from a detail file to update a master data base. The following explanation relates to the reference numbers along the left side of the figure.

Figure 1-9 (Page 1 of 2). General COBOL Batch Program Structure

```
REF.
NO.
                ENVIRONMENT DIVISION.
                      .
                      .
                DATA DIVISION.
                WORKING-STORAGE SECTION.
                    77 FUNC-GU           PICTURE XXXX VALUE 'GU  '.
                    77 FUNC-GHU          PICTURE XXXX VALUE 'GHU '.
                    77 FUNC-GN           PICTURE XXXX VALUE 'GN  '.
 1                   77 FUNC-GHN          PICTURE XXXX VALUE 'GHN '.
                     77 FUNC-GNP          PICTURE XXXX VALUE 'GNP '.
                     77 FUNC-GHNP         PICTURE XXXX VALUE 'GHNP'.
                     77 FUNC-REPL         PICTURE XXXX VALUE 'REPL'.
                     77 FUNC-ISRT         PICTURE XXXX VALUE 'ISRT'.
                     77 FUNC-DLET         PICTURE XXXX VALUE 'DLET'.
                     77 FUNC-CHKP         PICTURE XXXX VALUE 'CHKP'.
                     77 COUNT             PICTURE S9(5)VALUE +4 COMPUTATIONAL.
 2                   01 UNQUAL-SSA.
                        02 SEG-NAME       PICTURE X(08) VALUE  '        '.
                        02 FILLER         PICTURE X     VALUE  ' '.
 3                   01 QUAL-SSA-MAST.
                        02 SEG-NAME-M     PICTURE X(08) VALUE 'ROOT    '.
                        02 BEGIN-PAREN-M  PICTURE X     VALUE '('.
                        02 KEY-NAME-M     PICTURE X(08) VALUE 'KEY     '.
                        02 REL-OPER-M     PICTURE X(02) VALUE ' ='.
                        02 KEY-VALUE-M    PICTURE X(06) VALUE 'vvvvvv'.
                        02 END-PAREN-M    PICTURE X     VALUE ')'.
                     01 QUAL-SSA-DET.
                        02 SEG-NAME-D     PICTURE X(08) VALUE 'ROOT    '.
                        02 BEGIN-PAREN-D  PICTURE X     VALUE '('.
                        02 KEY-NAME-D     PICTURE X(08) VALUE 'KEY     '.
                        02 REL-OPER-D     PICTURE X(02) VALUE ' ='.
                        02 KEY-VALUE-D    PICTURE X(06) VALUE 'vvvvvv'.
                        02 END-PAREN-D    PICTURE X     VALUE ')'.
 4                      01 DET-SEG-IN.
                        02 --
                        02 --
                     01 MAST-SEG-IN.
                        02 --
                        02 --
```

Figure 1-9 (Page 2 of 2). General COBOL Batch Program Structure

```
 5          LINKAGE SECTION.
               01 DB-PCB-MAST.
                  02 MAST-DBD-NAME  PICTURE X(8).
                  02 MAST-SEG-LEVEL PICTURE XX.
                  02 MAST-STAT-CODE PICTURE XX.
                  02 MAST-PROC-OPT  PICTURE XXXX.
                  02 FILLER         PICTURE S9(5) COMPUTATIONAL.
                  02 MAST-SEG-NAME  PICTURE X(8).
                  02 MAST-LEN-KFB   PICTURE S9(5) COMPUTATIONAL.
                  02 MAST-NU-SENSEG PICTURE S9(5) COMPUTATIONAL.
                  02 MAST-KEY-FB    PICTURE XXXXX.
               01 DB-PCB-DETAIL.
                  02 DET-DBD-NAME   PICTURE X(8).
                  02 DET-SEG-LEVEL  PICTURE XX.
                  02 DET-STAT-CODE  PICTURE XX.
                  02 DET-PROC-OPT   PICTURE XXXX.
                  02 FILLER         PICTURE S9(5) COMPUTATIONAL.
                  02 DET-SEG-NAME   PICTURE X(8).
                  02 DET-LEN-KFB    PICTURE S9(5) COMPUTATIONAL.
                  02 DET-NU-SENSEG  PICTURE S9(5) COMPUTATIONAL.
                  02 DET-KEY-FB     PICTURE XXXXX.
            PROCEDURE DIVISION.
 6             ENTRY 'DLITCBL' USING DB-PCB-MAST, DB-PCB-DETAIL.
                .
                .
 7             CALL 'CBLTDLI' USING FUNC-GU, DB-PCB-DETAIL,
                     DET-SEG-IN, QUAL-SSA-DET.
                .
                .
 8             CALL 'CBLTDLI' USING COUNT, FUNC-GHU, DB-PCB-MAST,
                     MAST-SEG-IN, QUAL-SSA-MAST.
                .
                .
 9             CALL 'CBLTDLI' USING FUNC-GHN, DB-PCB-MAST,
                     MAST-SEG-IN.
                .
                .
10             CALL 'CBLTDLI' USING FUNC-REPL, DB-PCB-MAST,
                     MAST-SEG-IN.
                .
                .
11             GOBACK.
```

12 COBOL LANGUAGE INTERFACE

1. A 77 level or 01 level working storage entry defines each of the call functions used by the batch program. Each picture clause is defined as 4 alphameric characters and has a value assigned for each function (for example, 'GUbb'). If the optional count fieldwere to be included in the call statement, count values could be initialized for each type of call. The COBOL copy function could be used to include these standard descriptions into the program.
2. A 9-byte area is set up to be used in the calls that require an unqualified SSA. Before the call is issued, a segment name is moved into this field. If a call requires 2 or more unqualified SSAs, additional areas may be required.
3. An 01 level working storage entry defines each SSA used by an application program.

A separate SSA structure is required for each segmewnt type accessed by the program because the key-value fields should be different. Once the fields other than key-value are initialized they need not be altered.

4. A 01 level working storage entry defines the program segment I/O area. This area can be further defined with 02 entries. Separate I/O areas may be allocated for each segment type prescribed, or a single area can be used.
5. A 01 level Linkage Section entry describes the data base PCB entry for every input or output data base. It is through this linkage that a COBOL program may access the status codes after a DL/I call. The individual fields in the PCB are defined in the linkage section so that they may be referenced in the program.
6. This is the standard entry point in the procedure division of a batch program. After DL/I control has loaded the PSB for the program in the batch partition, it gives control to the application program. The PSB contains all the PCBs used by the program. The USING statement at the entry point to the batch program must contain the same number of names in the same sequence as there are PCBs in the PSB.
7.
8. These are typical calls to retrieve data from a data base using a qualified search argument. Before issuing the call, the key value of the SSA must be initialized to specify the particular segment to be retrieved. Immediately following the call a test should be made of the status-code field of the PCB to determine if the call was successful. See “Problem Determination” in Chapter 2 of this manual.
9.
10. This is a typical call to retrieve data from a data base using no SSA. This call is also a hold call for a subsequent delete or replace operation.
11. This statement replaces data in the data base with data from a COBOL batch program.
12. The GOBACK statement causes the batch program to return control to DL/I.
13. A language interface module (DLZLI000), which must be link-edited to the batch program after compilation, provides a common interface to DL/I. The call statement causes a V-type address constant (CBLTDLI) to be generated for the language interface module. When the application program is link-edited, the VSE automatic library look-up (AUTOLINK) feature retrieves the language interface module from a relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module. You must also include the following additional statements which link to ILBDSET0 (a COBOL requirement), in the input to the linkage editor:

```
INCLUDE DLZBPJRA
ENTRY   CBLCALLA
```

COBOL MPS Restart Example

A VSE checkpoint is issued automatically in COBOL programs each time a file is processed a predetermined number of times. The number of times may be specified by using the RERUN statement. To invoke a VSE checkpoint as part of a combined checkpoint in COBOL, the following format is suggested:

```
 CBL XOPTS(CICS,DLI)...
 ID DIVISION
 PROGRAM-ID. DLZCBLMP
         .
         .
         .
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT CHKPT-MSGS ASSIGN TO SYS101-UR-3203-S.
 I-O-CONTROL.
* TAKE A VSE CHKPT ON EVERY WRITE TO CHKPT-MSGS
*
     RERUN ON SYS100-UT-2400-S
         EVERY 1 RECORDS OF CHKPT-MSGS.
 DATA DIVISION.
 FILE SECTION.
 FD  CHKPT-MSGS
     LABEL RECORD OMITTED
     RECORDING IS F
     RECORD CONTAINS 72 CHARACTERS
     DATA RECORD IS MSG-LINE.
 01  MSG-LINE.
     02  FILLER    PIC X
     02  ALL-71    PIC X(71).
         .
         .
         .
 WORKING-STORAGE SECTION.
 01  INITMSG       PIC X(72) VALUE ' READY FOR VSE CHECKPOINTS '.
 01  CHKPTMSG      PIC X(72) VALUE ' TAKING A VSE CHECKPOINT '.
 01  CHKPTID       PIC X(8)  VALUE 'DL/ICHKP'.
         .
         .
         .
 PROCEDURE DIVISION.
      OPEN OUTPUT CHKPT-MSGS.
* INITIALIZE CHECK MESSAGE FILE
*
      WRITE MSG-LINE FROM INITMSG AFTER POSITIONING 2.
         .
         .
         .
* TAKE AN IMPLICIT VSE CHECKPOINT
*
      WRITE MSG-LINE FROM INITMSG AFTER POSITIONING 2.
* TAKE A DL/I CHECKPOINT
*
     CALL 'CBLTDLI' USING
        FUNC-CHKP,DB-PCB-DETAIL,CHKPTID
         .
         .
         .
```

In this example, a print file is defined so that a VSE checkpoint is invoked each time a message is written to it. The initial message is necessary because COBOL does not start issuing VSE checkpoints until the write statement after the first write

occurs. COBOL also issues a VSE checkpoint when the print file is closed at the end of the program. This additional checkpoint does not affect MPS Restart.

Since the invocation of a VSE checkpoint in COBOL is automatic and the VSE checkpoint ID is not available to the application program, it cannot be used as the checkpoint ID on the DL/I checkpoint command.

When using MPS Restart, a VSE checkpoint must be coded before each DL/I checkpoint in the application program. The two checkpoints together are referred to as a combined checkpoint.

PL/I Batch Program Structure

Figure 1-10 illustrates in outline form the fundamental parts in the structure of a PL/I batch program which, in this example, is to retrieve data from a detail file to update a master data base. The following explanation relates to the reference numbers along the left side of the figure.

Figure 1-10 (Page 1 of 2). General PL/I Batch Program Structure

```
REF.    /*                                                          */
NO.     /*                    ENTRY POINT                           */
        /*                                                          */
 1      DLITPLI: PROCEDURE (DB_PTR_MAST,DB_PTR_DETAIL)
                  OPTIONS (MAIN);
        /*                                                          */
        /*                      DESCRIPTIVE STATEMENTS              */
        /*                                                          */
        DCL  DB_PTR_MAST POINTER;
        DCL  DB_PTR_DETAIL POINTER;
        DCL  FUNC_GU      CHAR(4)     INIT('GU  ');
        DCL  FUNC_GN      CHAR(4)     INIT('GN  ');
 2      DCL  FUNC_GHU     CHAR(4)     INIT('GHU ');
        DCL  FUNC_GHN     CHAR(4)     INIT('GHN ');
        DCL  FUNC_GNP     CHAR(4)     INIT('GNP ');
        DCL  FUNC_GHNP    CHAR(4)     INIT('GHNP');
        DCL  FUNC_ISRT    CHAR(4)     INIT('ISRT');
        DCL  FUNC_REPL    CHAR(4)     INIT('REPL');
        DCL  FUNC_DLET    CHAR(4)     INIT('DLET');
        DLC  FUNC_CHKP    CHAR(4)     INIT('CHKP');
        DCL 1    QUAL_SSA             STATIC UNALIGNED,
 3                2   SEG_NAME         CHAR(8) INIT('ROOT    '),
                  2   SEG_QUAL         CHAR(1) INIT('('),
                  2   SEG_KEY_NAME     CHAR(8) INIT('KEY     '),
                  2   SEG_OPR          CHAR(2) INIT(' ='),
                  2   SEG_KEY_VALUE    CHAR(6) INIT('vvvvvv'),
                  2   SEG_END_CHAR     CHAR(1) INIT(')');
        DCL 1    UNQUAL_SSA           STATIC UNALIGNED,
                  2   SEG_NAME_U       CHAR(8) INIT('NAME    '),
                  2   BLANK            CHAR(1) INIT(' ');

        DCL 1    MAST_SEG_IO_AREA,
```

Figure 1-10 (Page 2 of 2). General PL/I Batch Program Structure

```
4           2 ---
            2 ---
            2 ---
          DCL 1    DET_SEG_IO_AREA,
            2 ---
            2 ---
            2 ---
          DCL 1    DB_PCB_MAST          BASED (DB_PTR_MAST),
            2   MAST_DB_NAME     CHAR(8),
            2   MAST_SEG_LEVEL   CHAR(2),
            2   MAST_STAT_CODE   CHAR(2),
5           2   MAST_PROC_OPT    CHAR(4),
            2   FILLER           FIXED BINARY (31,0),
            2   MAST_SEG_NAME    CHAR(8),
            2   MAST_LEN_KFB     FIXED BINARY (31,0),
            2   MAST_NO_SENSEG   FIXED BINARY (31,0),
            2   MAST_KEY_FB      CHAR(*);
          DCL 1    DB_PCB_DETAIL        BASE (DB_PTR_DETAIL),
            2    DET_DB_NAME     CHAR(8),
            2    DET_SEG_LEVEL   CHAR(2),
            2    DET_STAT_CODE   CHAR(2),
            2    DET_PROC_OPT    CHAR(4),
            2    FILLER              FIXED BINARY (31,0),
            2    DET_SEG_NAME    CHAR(8),
            2    DET_LEN_KFB     FIXED BINARY (31,0),
            2    DET_NO_SENSEG   FIXED BINARY (31,0),
            2    DET_KEY_FB      CHAR(*);

          DCL      THREE     FIXED BINARY    (31,0)   INITIAL(3);
          DCL      FOUR      FIXED BINARY    (31,0)   INITIAL(4);
          DCL      FIVE      FIXED BINARY    (31,0)   INITIAL(5);
6         DCL      SIX       FIXED BINARY    (31,0)   INITIAL(6);
          /*                                                          */
          /*    MAIN PART OF PL/I BATCH PROGRAM                       */
           .
7               CALL PLITDLI(FOUR,FUNC_GU,DB_PCB_DETAIL,
                DET_SEG_IO_AREA,QUAL_SSA);
           .
8               CALL PLITDLI(FOUR,FUNC_GHU,DB_PCB_MAST,
                MAST_SEG_IO_AREA,QUAL_SSA);
           .
9               CALL PLITDLI(THREE,FUNC_GHN,DB_PCB_MAST,
                MAST_SEG_IO_AREA;
            .
10              CALL PLITDLI(THREE,FUNC_REPL,DB_PCB_MAST,
                MAST_SEG_IO_AREA);
            .
11              RETURN;
                END DLITPLI;
```

```
12          PL/I LANGUAGE INTERFACE
```

1. This is the main entry point to a PL/I batch program. After the DL/I control program has loaded and relocated the PSB for the program, it gives control to this entry point. The PSB contains all the PCBs used by the program. The entry point statement of the batch program must contain the same number of names in the same sequence as there are PCBs in the PSB.

2. Each area defines one of the call functions used by the PL/I batch program. Each character string is defined as four alphameric characters, with a value assigned for each function (for example, 'GUbb'). Other constants may be defined in the same manner. Standard definitions could be stored in a source library and included using a %INCLUDE statement.
3. A structure declaration defines each SSA used by the problem program. The unaligned attribute is required for SSA data interchange with DL/I. The SSA character string must reside contiguously in storage. Assignment of variables to key values, for example, could result in the construction of an invalid SSA if the key value has the aligned attribute.

   A separate SSA structure is required for each segment type accessed by the program because the key-value fields should be different. Once the fields other than key-value are initialized, they should not have to be altered.

   A 9-byte area should be reserved for use as an unqualified SSA. Before issuing an unqualified call, a segment name is moved into this field.
4. The segment I/O areas are defined as structures.
5. One level 1 declarative (similar to COBOL's linkage section) describes as a structure the data base PCB entry for each input or output data base. It is through this description that a PL/I program may access the status codes after a DL/I call.
6.
7. This statement is used to identify a binary number (fullword) that represents the parameter count of a call to DL/I. The parameter count value equals the remaining number of arguments following the parameter count set off by commas.
8.
9. These are typical calls to retrieve data from a data base using a qualified SSA.

   Prior to execution of the call the SEG_KEY_VALUE field of the SSA must be initialized if a fully qualified SSA is required. For a call using an unqualified SSA, the segment name field must be moved to one of the 9-byte UNQUAL—SSA areas.

   Immediately following the call the status code field of the PCB must be checked to determine the results of the call. See "Problem Determination" in Chapter 2 of this manual.
10. This is a typical call to retrieve data from a data base using no SSA. This call is also a HOLD call for subsequent delete or replace operation.
11. This call is used to replace data in the data base with data from a PL/I batch program.
12. This RETURN statement causes the batch program to return control to DL/I.

13. A language interface module (DLZLI000), which must be link-edited to the batch program, provides a common interface to DL/I. The call statement causes a V-type address constant (PLITDLI) to be generated for the language interface module. When the application program is link-edited, the VSE automatic library look-up (AUTOLINK) feature retrieves the language interface module from a VSE relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module. The user must also include the following additional statements (a PL/I requirement) in the input to the linkage editor:

```
INCLUDE IBMBPJRA
ENTRY   PLICALLB
```

PL/I MPS Restart Example

The PL/I application program interface to a VSE checkpoint consists of a call to the built-in function PLICKPT. An example is:

```
 DECLARE CHKPTID CHAR(8);
 DECLARE RETCODE FIXED BIN(31);
      .
      .
      .
CALL PLICKPT('',CHKPTID,'SYS100,2400',RETCODE); /*ISSUE A VSE CHKPT*/
IF RETCODE > 4 THEN DO            /*VSE CHKPT ERROR*/
  PUT EDIT('ERROR DURING CHECKPOINT. RETCODE=',RETCODE)(A,F(2));
  STOP;                           /*ABNORMAL END*/
  END;
IF RETCODE = 4 THEN               /*VSE RESTART OCCURRED*/
  PUT EDIT('RESTARTED AT CHECKPOINT #',CHKPTID)(A);
EXEC DLI CHECKPOINT ID(CHKPTID); /*ISSUE A DL/I CHKP  WITH VSE CHKPT ID*/
EXEC DLI GET UNIQUE SEGMENT(ROOT) /*REPOSITION DATA BASE*/
            INTO(IOAREA)
            WHERE(KEYFIELD = LASTKEY);
```

When using MPS Restart, a VSE checkpoint must be coded before each DL/I checkpoint in the application program. The two checkpoints together are referred to as a combined checkpoint.

RPG II Batch Program Structure

Figure 1-11 illustrates in outline form the fundamental parts in the structure of an RPG II batch program which, in this example, is to retrieve data from a detail file to update a master data base. The following explanation relates to the reference numbers along the right side of the figure.

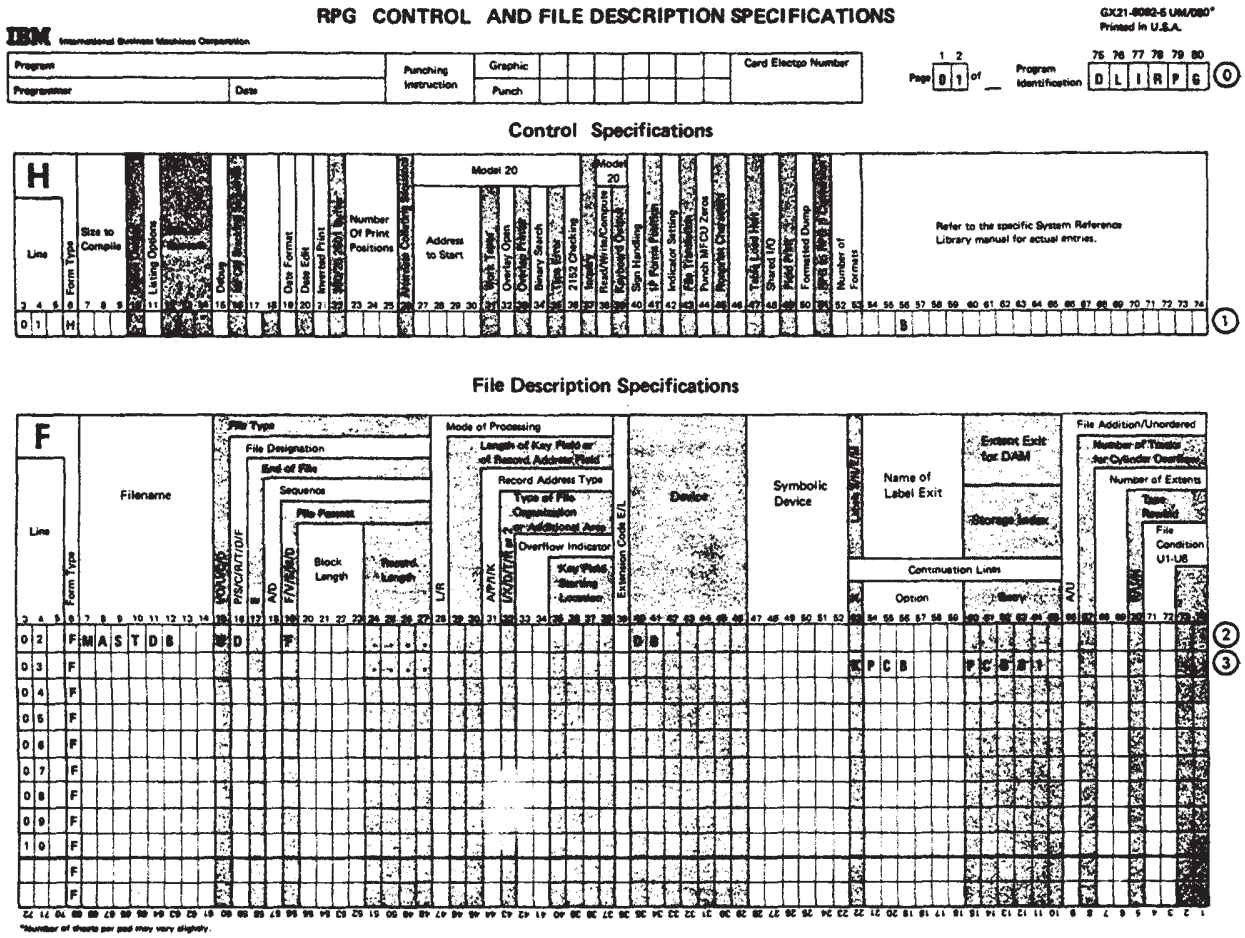

Figure 1-11 (Part 1 of 6). General RPG II Batch Program Structure

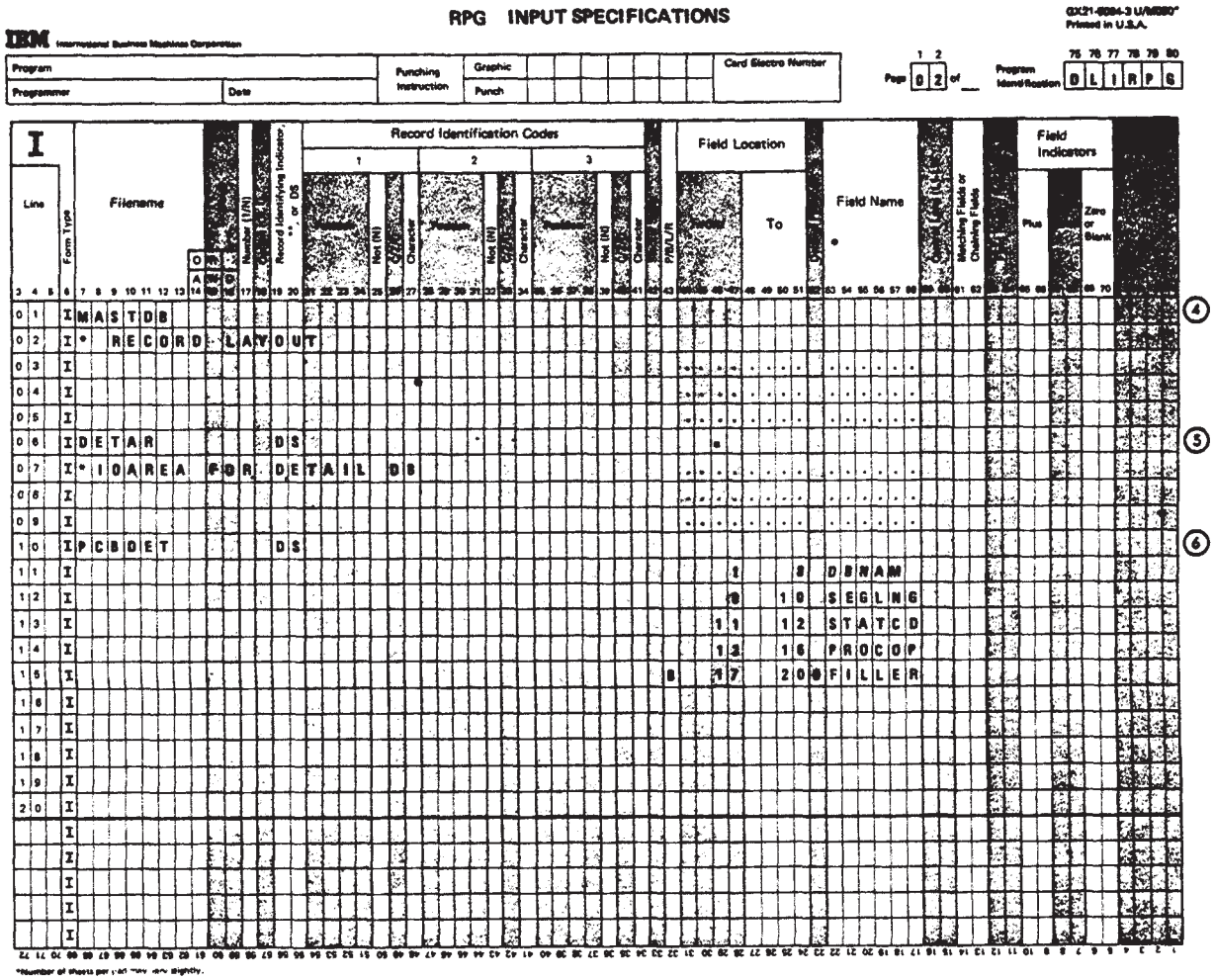

Figure 1-11 (Part 2 of 6). General RPG II Batch Program Structure

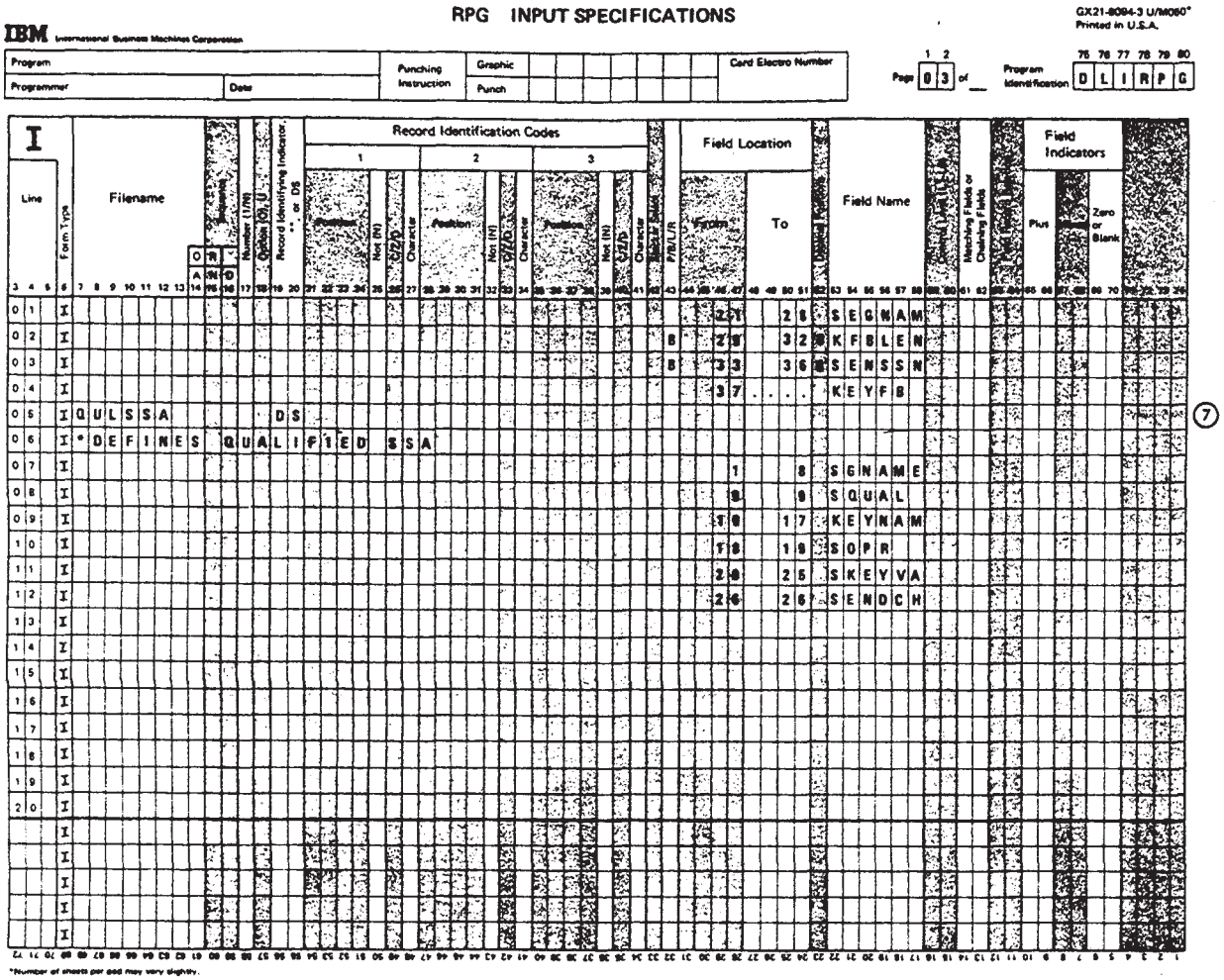

Figure 1-11 (Part 3 of 6). General RPG II Batch Program Structure

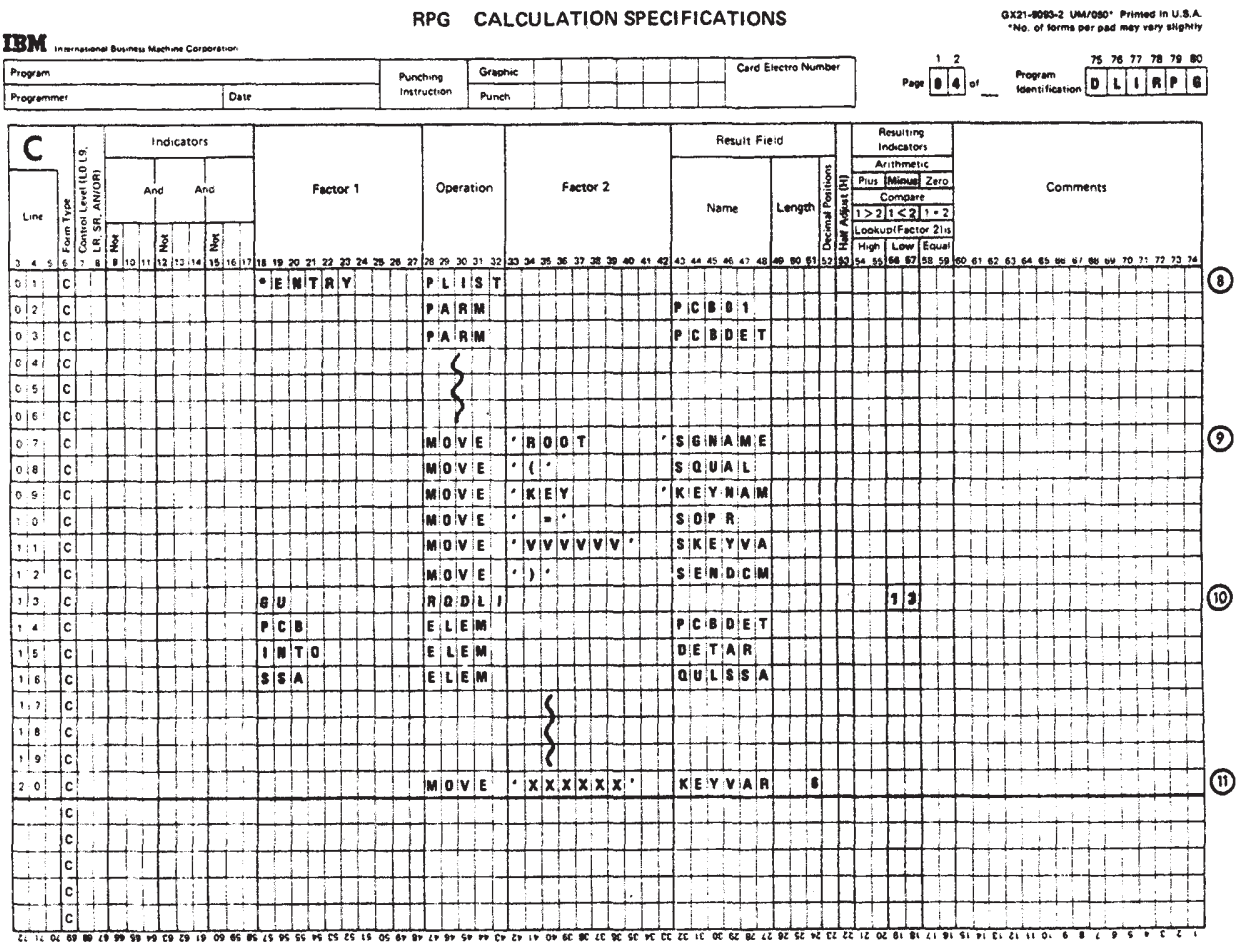

Figure 1-11 (Part 4 of 6). General RPG II Batch Program Structure

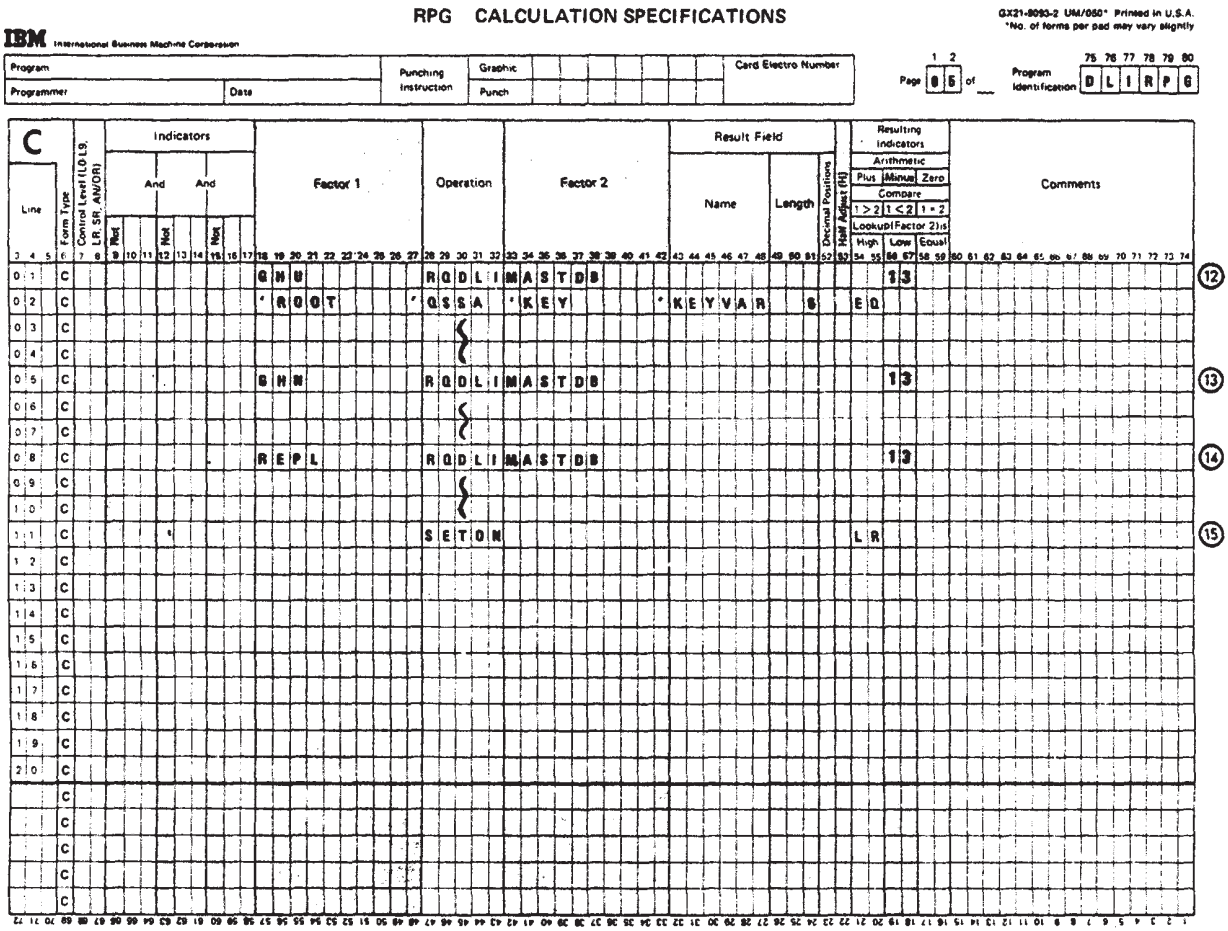

Figure 1-11 (Part 5 of 6). General RPG II Batch Program Structure

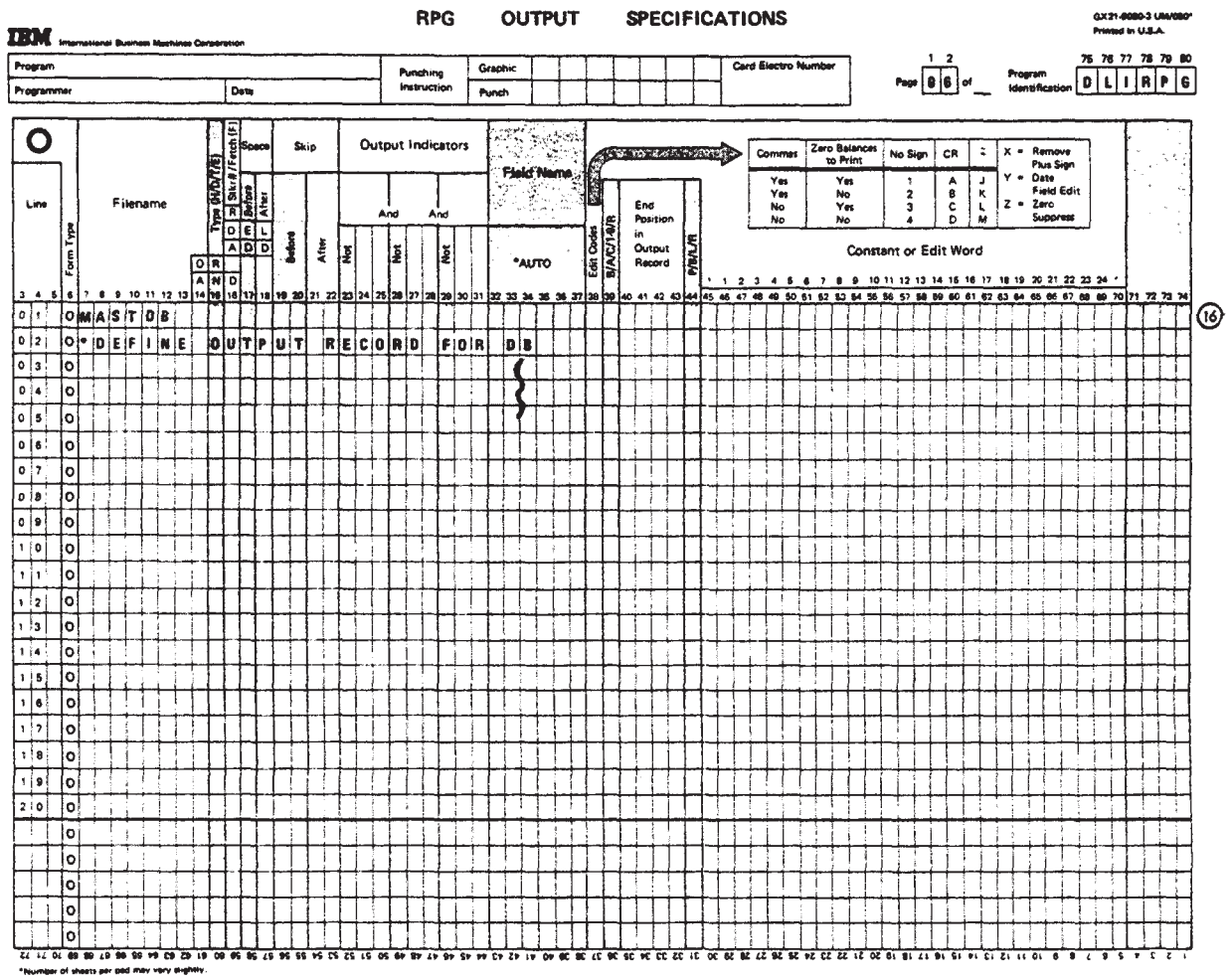

Figure 1-11 (Part 6 of 6). General RPG II Batch Program Structure

1. The user may specify any program name.
2. Specify B in position 56 to tell the Translator that this is an RPG II program which will run under DL/I.
3. A data base may be defined as a DB-file. In this case the PCB can be specified in a continuation line (K in position 53) to the DB-file specification.
4. The continuation line defines the PCB associated with the DB-file. In this case the PCB will be automatically generated. The Translator will put the definition into the Input Specifications as a data structure.
5. This is the segment definition of the DB-file MASTDB.
6. This data structure provides an I/O area for use by RQDLI commands without referring to a File Description Specification.
7. This is an example of a user-defined PCB. It must have the same layout as the automatically generated PCB; however, you may use names of your own choice.
8. This data structure can be used to build a qualified SSA.
9. Since not all data bases are defined as DB-files, you must explicitly specify the *ENTRY PLIST. After the DL/I control program has loaded the PSB, it gives control to the RPG II program. The PSB contains all the PCBs used by the

program. Therefore, the PARM statements of the *ENTRY PLIST must specify the PCBs in the same sequence they are specified in the PSB.

10. The following MOVE statements build a qualified SSA to be used in an SSA option of the RQDLI command. Reference numbers 10-12 show typical commands to retrieve data from a data base.
11. This RQDLI command explicitly uses the I/O area DETAR.

    Immediately following the RQDLI command, you must check the status code field of the PCB (by using compare operations, as illustrated in Figure 2-4) to determine the results of the RQDLI command.
12. This MOVE statement specifies that a new value for KEYVAR will be used in the following RQDLI command (reference number 12).
13. This RQDLI command does not use our I/O area. In this case the Translator takes the information and adds it to the records defined in the Input Specifications. This command is used to show how it is possible to specify a qualified SSA in a QSSA statement.
14. This is a typical RQDLI command to retrieve data from a data base using no SSA. This RQDLI command is also a HOLD RQDLI command for subsequent delete or replace operations.
15. This RQDLI command is used to replace data in the data base with data from an RPG II batch program. It uses MASTDB as defined in the Output Specifications to define the I/O area.
16. SETON LR must be coded to return control to DL/I.
17. This is the segment definition for output of the MASTDB file.

Note: A language interface module (DLZLI000) must be link-edited to the application program to provide a common interface to DL/I. When the application program is link-edited, the VSE automatic library look-up (AUTOLINK) feature retrieves the language interface module from a relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module.

Assembler Language Batch Program Structure

Figure 1-12 illustrates, in outline form, the fundamental parts in the structure of an Assembler language batch program which, in this example, is to retrieve data from a detail file to update a master data base. The following explanation relates to the references along the left side of the figure. Throughout this figure, CBLTDLI could appear instead of ASMTDLI.

Figure 1-12 (Page 1 of 2). General Assembler Language Batch Program Structure

```
         PGMSTART     CSECT
                      USING     *,R12
                      SAVE      (R14,R12)
1                     LR        R12,R15
                      ST        R13,SAVEAREA+4
                      LA        R13,SAVEAREA
                      •
2                     MVC       DBPCBMST(8),0(R1)
                      •
                      MVC       DLIFUNC,GU
                      MVC       PCB,DBPCBDET
                      LA        R1,DETSEGIO
                      ST        R1,IOAREA
                      LA        R1,SSANAME
                      ST        R1,SSA
                      LA        R1,PARMLIST
3                     CALL      ASMTDLI
                      •
                      MVC       DLIFUNC,GHU
                      MVC       PCB,DBPCBMST
                      LA        R1,MSTSEGIO
                      ST        R1,IOAREA
                      MVI       SSANAME+8,C' '
                      LA        R1,PARMLIST
4                     CALL      ASMTDLI
                      •
                      MVC       DLIFUNC,GHN
                      MVI       PARMCT+3,3
                      LA        R1,PARMLIST
5                     CALL      ASMTDLI
                      •
                      MVC      DLIFUNC,REPL
                      LA       R1,PARMLIST
6                     CALL     ASMTDLI
                      •
                      L        R13,4(R13)
7                     RETURN   (R14,R12)
         *    CONSTANT AREA
                      •
8        PARMLIST     DC       A(PARMCT)
         FUNC         DC       A(DLIFUNC)
         PCB          DC       A(0)
         IOAREA       DC       A(0)
         SSA          DC       A(0)
                      •
9        DBPCBMST     DC        F'0'
         DBPCBDET     DC        F'0'
                      •
         PARMCT       DC        F'4'
         DLIFUNC      DC        CL4'    '
         GU           DC        CL4'GU  '
```

Figure 1-12 (Page 2 of 2). General Assembler Language Batch Program Structure

```
10          GHU         DC        CL4'GHU '
            GHN         DC        CL4'GHN '
            REPL        DC        CL4'REPL'
            CHKP        DC        CL4'CHKP'
                        •
            SSANAME     DS        0CL26
            ROOT        DC        CL8'ROOT    '
11                      DC        CL1'('
                        DC        CL8'KEY     '
                        DC        CL2' ='
            NAME        DC        CL6'vvvvvv'
                        DC        CL1')'
                        •
12          MSTSEGIO    DS        CL100
            DETSEGIO    DS        CL100
            SAVEAREA    DC        18F'0'
                        •
            PCBNAME     DSECT
            DBPCBDBD    DS        CL8       DBD NAME
            DBPCBLEV    DS        CL2       LEVEL FEEDBACK
            DBPCBSTC    DS        CL2       STATUS CODES
            DBPCBPRO    DS        CL4       PROC OPTIONS
            DBPCBRSV    DS        F         RESERVED
            DBPCBSFD    DS        CL8       SEGMENT NAME FEEDBACK
            DBPCBMKL    DS        F         CURRENT LENGTH OF KEY FEEDBACK
            *                               AREA
            DBPCBNSS    DS        F         NO OF SENSITIVE SEGMTS IN PCB
            DBPCBKFD    DS        CL6       KEY FEEDBACK AREA
                        END

13                      ASSEMBLER LANGUAGE INTERFACE
```

1. The entry point to the Assembler language program. See the discussion under “Assembler” in the section “Entry to an Application Program” earlier in this chapter. The base register 12 is used in this example.
2. When control is passed to the application program, register 1 contains the address of a variable-length fullword parameter list. See the discussion under “Assembler” in the section “Entry to an Application Program” earlier in this chapter. As only explicit calls are issued in this example, there is no need to reset the 0 bit from 1 to 0.
3. This is a typical call to retrieve data from the master data base using a qualified search argument. All DL/I calls should be executed with the CALL macro instruction. Prior to execution of the call statement register 1 must point to the variable-length fullword parameter list. This may be done through operands of the CALL macro instruction. If the user constructs his own parameter list, the leftmost bit of the last entry in the list must be set to one unless a parameter count is specified as shown in this example. Immediately following the call, the status code of the PCB should be tested to determine the result of the call. See “Problem Determination” in Chapter 2 of this manual.
4. This call (and calls 5 and 6) are to the detail data base and therefore the PCB address and I/O area must be reloaded. This call has an unqualified SSA by setting the left parenthesis in position 9 to blank.
5. This is a typical call to retrieve data from a data base, which, by setting the parm-count to three, uses no SSA. This call is also a HOLD call for a subsequent delete or replace.

6. This call is used to replace data in the detail data base.
7. The RETURN statement loads DL/I registers and causes the batch program to return control to DL/I.
8. In this illustration, one variable-length parameter list is defined to process all data base calls. The list contains pointers to the parameter count, DL/I call function, PCB, I/O area, and SSA. The value in PARMCT determines the length of the parameter list. In this case, the count of four indicates that the following four addresses constitute the parameter list.
9. A fullword must be defined for every data base PCB. The Assembler language program may access the status codes after a DL/I call using the PCB base addresses.
10. The call functions are defined as 4-character constants.
11. The SSAs must be defined by the problem program.
12. An I/O area large enough to contain the largest segment of a data base must be provided. In Figure 1-12 on page 1-49, it is assumed that the longest segment does not exceed 100 bytes. As previously mentioned, an 18-fullword register save area must be provided in the application program.
13. A language interface module (DLZLI000) must be link-edited to the batch program after assembly to provide a common interface to DL/I. The call statement causes a V-type address constant (ASMTDLI or CBLTDLI) to be generated for the language interface module. When the application program is link-edited, the VSE automatic library look-up (AUTOLINK) feature retrieves the language interface module from a VSE relocatable library (system or private) and link-edits it with the application program. If AUTOLINK is suppressed, an INCLUDE statement must be present for the language interface module.

Assembler MPS Batch Example

Assembler language programs use the VSE checkpoint macro directly as shown in the following example.

```
CHKPT1   DS    0H                  COMBINED CHECKPOINT #1
         LA    R5,RESUME1          GET RESUME ADDRESS
         CHKPT SYS100,RSTRT        ISSUE A VSE CHKPT - SYS100
*                                  IS TAPE, RSTRT IS ADDRESS
*                                  OF RESTART ROUTINE
RESUME1  DS    0H                  RESUME PROCESSING AFTER RESTART
         LTR   R0,R0               CHECK RETURN CODE
         BZ    ERROR               BRANCH IF ERROR
         ST    R0,CHKPTID          USE VSE CHKPT ID ON DL/I CHKP
         OI    CHKPTID+3,X'F0'     MAKE LAST DIGIT PRINTABLE
         LA    R1,CHKPPARM         GET ADDRESS OF CHKP PARM LIST
         CALL  ASMTDLI             ISSUE A DL/I CHKP
         BAL   R5,VERSTAT          VERIFY DL/I STATUS CODE
         LA    R1,GUPARM           GET ADDRESS OF GU PARM LIST
         CALL  ASMTDLI             REPOSITION DATA BASE
         .                         CONTINUE PROCESSING
         .
RSTRTR   DS    0H                  RESTART ROUTINE
         STXIT AB...               REISSUE STXIT AND OTHER
         .                         RESTART PROCESSING
         .
         BR    R5                  RESUME PROCESSING
VERSTAT  DS    0H                  VERIFY DL/I STATUS CODE
         CLC   DBPCBSTC,BLANK      IS STATUS CODE BLANKS?
         BER   R5                  YES - RETURN TO CALLER
         .                         NO - CHECK FOR POSSIBLE ERROR
ERROR    DS    0H                  VSE CHECKPOINT ERROR ROUTINE
         .
CHKPPARM DS    0F
         DC    A(CHKP)             FUNCTION
CHKPPCB  DS    A                   PCB ADDRESS
         DC    XL1'80'             LAST PARM INDICATOR
         DC    AL3(CHKPTID)        I/O AREA
GUPARM   DS    0F
         DC    A(GU)               FUNCTION
GUPCB    DS    A                   PCB ADDRESS
         DC    A(IOAREA)           I/O AREA
         DC    XL1'80'             LAST PARM INDICATOR
         DC    AL3(SSA)            SSA
BLANKS   DC    CL2'  '             BLANKS FOR STATUS CODE CHECK
CHKP     DC    CL4'CHKP'           CHECKPOINT FUNCTION
GU       DC    CL4'GU  '           GET UNIQUE FUNCTION
CKPALIGN DS    0F                  CHKPTID FULLWORD ALIGNED FOR
*                                  STORE INSTRUCTION
CHKPTID  DC    CL8' '              DL/I CHECKPOINT ID
IOAREA   DS    CL80                I/O BUFFER
SSA      DS    0F                  SSA
         .
         .
```

As shown in the example, you may want to use the VSE checkpoint ID as the checkpoint ID for the DL/I checkpoint. This would provide a cross reference between the normal checkpoint messages issued to SYSLOG as the result of taking a VSE checkpoint and a DL/I checkpoint. However, this is not necessary for MPS Restart.

When using MPS Restart, a VSE checkpoint must be coded before each DL/I checkpoint in the application program. The two checkpoints together are referred to as a combined checkpoint.

Restrictions

On COMREG Use

Bytes 16 through 19 of the communication region are used by DL/I and therefore must not be used by the application program.

On Overlay Programs

Overlay structures are not supported for application programs executed under DL/I. Although the COBOL SORT verb automatically produces an overlay structure, the restriction does not apply if the job control statements used to compile and link-edit the program as shown below are used. (For COBOL programs that do not contain the SORT verb, the INCLUDE DLZBPJRA statement may, alternatively, be placed immediately before the ENTRY statement). The use of “PL/I-SORT” programs, using the sort program product, is not affected by this restriction provided that an overlay structure is not explicitly specified.

Set Exit Abnormal (STXIT AB) Linkage

The DL/I user has the option, through the use of the user program switch indicator (UPSI), of permitting STXIT AB linkage to pass control to DL/I prior to abnormal termination so that a controlled shutdown may occur. The DL/I system log and DL/I data base are closed and a storage dump is provided. However, non-DL/I files are not closed; this is a user responsibility.

If a COBOL application program is executing under DL/I control, any attempt by the application program to execute the COBOL debug function may cause unpredictable results. Therefore, no COBOL debug function (any COBOL option which makes use of a STXIT routine) should be used if DL/I STXIT is used. Refer to your COBOL publications for options which use STXIT linkages.

The High Level Language (HLL) Debugging facility makes the job of an application programmer writing in PL/I easier by allowing diagnostic information to be supplied by both PL/I and DL/I. When a program check is detected during application program execution, a STXIT PC routine will be given control if STXIT support has been requested of DL/I (UPSI bit 7 = 0 for batch, and always for MPS batch). This facility applies only to batch and MPS batch execution of DL/I.

Application Language Use in Batch or MPS Batch Programs

If the program specified on the DL/I parameter statement was written in PL/I, then

- the PSB specified on the DL/I parameter statement must have PL/I as the language. (See LANG parameter of the PSBGEN statement or DLZACT TYPE=RPSB statement.)
- all of the program subroutines which use DL/I must be written in PL/I.

If the program specified on the DL/I parameter statement was not written in PL/I then

- the PSB specified on the DL/I parameter statement must not have PL/I as the language.
- all of the program subroutines which use DL/I may be any programming language except PL/I.

Mixing Batch PL/I and Other Languages Using DL/I

Batch PL/I application programs must not call, or be called by, another non-PL/I application program if both of them perform DL/I calls.

A PL/I program issuing DL/I calls cannot call a program written in Assembler, COBOL, or RPG II if the called program also issues calls. Conversely, an Assembler, COBOL, or RPG II program that issues DL/I calls cannot call a PL/I program that also issues DL/I calls.

Mixing these two types of batch application programs can cause a program check or other unpredictable results.

Boolean Operators and SSA Length

The length of a segment search argument (SSA) that can be transmitted to a remote DL/I system is limited to 304 bytes. Boolean SSAs with a length of 304 bytes or less will be processed; those with a length greater than 304 bytes will be rejected in the remote system, causing, the DL/I call to fail.

Job Control Statements for Batch and MPS Batch DL/I Application Programs

DL/I application programs cannot be processed in a compile-link-go environment. Programs must first be compiled and link-edited to a core image library and then executed as a separate job, as they run as a subprogram of the DL/I initialization program. RPG II programs must be translated prior to compilation.

Compile and Link-Edit

COBOL

```
// JOB COBSAMPL
// OPTION CATAL
   PHASE COBSAMPL,*
   INCLUDE DLZBPJRA
// EXEC FCOBOL
        •
        •
   SOURCE DECK
        •
        •
/*
   ENTRY CBLCALLA
// EXEC LNKEDT
/&
```

PL/I

```
// JOB PLISAMPL
// OPTION CATAL
   PHASE PLISAMPL,*
// EXEC PLIOPT
        •
        •
   SOURCE DECK
        •
        •
/*
   INCLUDE IBMBPJRA
   ENTRY PLICALLB
// EXEC LNKEDT
/&
```

Assembler

```
// JOB ASSSAMPL
// OPTION CATAL
   PHASE ASSSAMPL,*
// EXEC ASSEMBLY
        •
        •
   SOURCE DECK
        •
        •
/*
// EXEC LNKEDT
/&
```

RPG II

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

RPG II program——> (with B specified in Col. 56 of Header Specifications) **Translator**——> **RPG II Compiler**——> (Auto report optional) **Appl.prog.**

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

The function of the Translator is to accept as input a source program, written in RPG II, in which DL/I requests have been coded via RQDLI commands. The Translator produces as output an equivalent source program in which the DL/I requests have been translated into call statements together with READ and EXCPT statements. At execution time the call statements invoke DL/I, passing appropriate arguments.

The Translator is executed in a separate job step. The job step sequence for compiling an application program is thus translate-compile-link edit. The Translator requires a minimum of 96k bytes of virtual storage. Its phase name is RPGIXLTR.

The Translator reads its input from SYSIPT, produces its output (the translated source program) on SYSPCH, or optionally on SYS002 or SYS003, and writes the source listing, error messages, etc. on SYSLIST.

The RPG II Translator provides a number of options. They may be specified in the // OPTION Job Control Statement.

The Translator options for RPG II in a DL/I batch environment are:

```
LIST | NOLIST
DUMP | NODUMP
```

Defaults are according to SYSGEN.

LIST: A listing of the source program is printed on SYSLST
DUMP: When unrecoverable errors occur, a dump is produced.
NODUMP: When unrecoverable errors occur, no dump is produced.

Translator Output

The DB-file descriptions are translated into File Description Specifications for SPECIAL files.

An RQDLI command not specifying a file-name or with an explicit FROM or INTO option is translated into a call statement followed by PARM statements.

For standard data transfer (existing Input and/or Output Specifications and no FROM or INTO option explicitly specified) the RQDLI command is translated into a call statement followed by a READ statement for input, or an EXCPT statement for output.

Additionally, the Translator generates data structures and fields together with MOVE or Z-ADD statements to build the proper SSAs from the USSA and QSSA statements, which are then used in the PARM statements of the generated call statement.

Note: When link-editing an RPG II batch program using standard data transfer, an unresolved external reference message for DFHEI1 will appear in the linkage editor output listing unless the entry exists in the user's core image library. The reason for this is that, in the case of standard data transfer, the RPG module contains entry points DFHEI1 and RPGDLI for both CICS/VS and the batch environment respectively.

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

Example of job control for output on SYSPCH:

```
// JOB T
// EXEC RPGIXLTR (Translator)
    •
    •  Source to be translated
    •
/*
/&
```

If the leftmost two bits of the UPSI byte are 01, the Translator directs its output to SYS002. This method should be used if an RPG II compilation is to immediately follow the Translator run. The RPG II compiler will then read its input from SYS002 instead of SYSIPT.

Example of job control output on SYS002:

```
// JOB TRPG
// UPSI 01
// EXEC RPGIXLTR
    •
    •  Source to be translated
    •
/*
// EXEC RPGII
/&
```

If the leftmost two bits of the UPSI byte are 10, the Translator directs its output to SYS003. This method should be used if an Auto Report compilation is to

immediately follow the Translator run. Auto Report will then read its input from SYS003 instead of SYSIPT.

Example of job control for output on SYS003:

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

Batch and MPS Batch Application Program Execution

When a DL/I application program is executed, it actually runs under control of DL/I. The EXEC statement in the job stream names the DL/I initialization module rather than the application program name.

DL/I Parameter Statement

The application program to be executed and the PSB it uses are identified in a parameter statement that follows the EXEC statement.

The format of the parameter statement, beginning in column 1, is as follows:

```
DLI,progname,psbname[,{buff}]
                      {1   }
          [,HDBFR=({bufno}ffl,dbdname1,dbdname2,...])]],....“
                  {32   }
          [,HSBFR=({indno},{ksdsbuf},[{esdsbuf}],dbdname3)]
                  {3    } {2      }  {2      }
          [,TRACE=modname][,ASLOG=YES]
          [,LOG=({TAPE},{PAUSE})]
                 {DISK1}{NOPAUSE}
                 {DISK2}
```

or, for MPS Restart:

```
DLR,progname,psbname
```

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

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

DLI
: The DL/I function code is required for batch DL/I programs or MPS batch DL/I programs that do not use the MPS Restart facility.

DLR
: The DLR function code is required for MPS batch programs using the MPS Restart facility. If DLR is specified as the function code for a batch DL/I program, it will be treated exactly as if DLI had been specified. The DLR

function code must be used when the job is first started and not just when it is restarted.

progname
: specifies a one to eight alphameric character name of the application program or utility to be executed.

psbname
: specifies a one to seven alphameric character name of the PSB as indicated in the PSB generation and referenced by the application program.

buff
: specifies the number of data base subpools required for this execution which can be a numeric value 1 to 255; if omitted, 1 is assumed. If no buffer pool control options are specified, a subpool consists of 32 fixed-length buffers. The buffer size is generally consistent with the VSAM data base control interval size and may be 512 or any multiple of 512 bytes. The buffer size value is determined at DL/I system initialization and is based on the value specified in BFRPOOL, the number of data bases, and size of the VSAM control intervals. A data base is assigned a subpool which contains buffers that are equal to or greater in size than the size of the data base control interval. Refer to *DL/I DOS/VS Data Base Administration* for buffer pool information.

HDBFR
: describes one DL/I subpool. Refer to *DL/I DOS/VS Data Base Administration* for buffer pool information.

  - bufno specifies the number of buffers to be allocated for this subpool and is a numeric value from 2 to 32. If omitted for a specific subpool, 32 is assumed. A specification exceeding 2 digits will cause an abnormal termination.
  - dbdname1,dbdname2,... specify the names of DBDs that are to be allocated to this subpool. If no dbdnames are specified, this subpool is used for DMBs not explicitly assigned; the parentheses around the number of buffers are still required. The DBD name used should be the physical DBD even though a logical DBD is being used.

HSBFR
: defines VSAM buffer allocation for HISAM, SHISAM, and INDEX data bases. Refer to *DL/I DOS/VS Data Base Administration* for buffer pool information.

  - indno specifies the number of index buffers for a KSDS; if omitted, 3 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.
  - ksdsbuf specifies the number of data buffers for a KSDS; if omitted, 2 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.
  - esdsbuf specifies the number of data buffers for the ESDS (applies to HISAM only); if omitted, 2 is assumed. A specification of 1 or 2 digits is permitted. A specification exceeding 2 digits will cause an abnormal termination.
  - dbdname3 is the name of the HISAM, SHISAM, or INDEX DBD referenced by the application program.

TRACE
: indicates that tracing is to be active during this execution. See the *DL/I DOS/VS Diagnostic Guide*, for details on tracing.

ASLOG=YES
: specifies that asynchronous logging is to be used. See *DL/I DOS/VS Data Base Administration*.

LOG
: specifies the type of logging to be used.

  TAPE indicates the log records are to be written to a tape device. It is the default if the LOG parameter is omitted.

  DISK1 indicates the log records are to be written on one disk extent with the filename DSKLOG1.

  DISK2 indicates that the log records are to be written on two disk extents. If one disk extent becomes full, the extent is closed and the other extent is used. DSKLOG1 is used first; then DSKLOG2. If DSKLOG2 becomes full, logging will switch back to DSKLOG1 and continue to repeat the sequence.

  PAUSE indicates that before reusing the only disk extent (DISK1) or before switching to the next extent (DISK2), the operator is notified and the partition waits for the operator's reply. PAUSE is the default if the second option in the LOG parameter is omitted.

  NOPAUSE indicates that reusing a log extent or switching log extents is done without notifying the operator.

Note: The UPSI byte (bit 6=0) must be set to indicate that DL/I logging is required. If anything other than the above parameters are specified an error message is issued and the job is canceled.

UPSI Byte Settings for Batch DL/I

Several execution-time functions can be controlled by the UPSI setup. The format of the UPSI statement is as follows:

```
// UPSI    x0000xxx
```

The meanings of the bit settings are as follows:

Bit 0 = 0 Read parameter information via SYSIPT.

Bit 0 = 1 Read parameter information via SYSLOG.

Bits 1-4 Available for use by the application programmer.

Bit 5 = 0 Storage dump on set exit (STXIT) abnormal task termination if STXIT active (that is, bit 7 = 0).

Bit 5 = 1 No storage dump on set exit (STXIT) abnormal task termination.

Bit 6 = 0 All data base modifications written on to the DL/I system log.

Bit 6 = 1 DL/I system log function inactive.

Bit 7 = 0 Set exit (STXIT) linkage to DL/I for abnormal task termination.

Bit 7 = 1 STXIT inactive.

Note that UPSI byte settings in the online environment have different meanings than those for batch. Refer to *DL/I DOS/VS Resource Definition and Utilities* for online UPSI byte setting information.

If you are unsure of the significance of these functions, consult your system programmer or data base administrator.

UPSI Byte Settings for MPS

Bit 0 = 0 Read parameter information via SYSIPT.

Bit 0 = 1 Read parameter information via SYSLOG.

Bits 1-4 Available for use by the application programmer.

Bit 5 = 0 Storage dump on set exit (STXIT) abnormal termination.

Bit 5 = 1 No storage dump on (STXIT) abnormal termination.

Bits 6-7 Not used for MPS. Data base logging, normally controlled by UPSI bit 6 is controlled in the CICS/VS partition under MPS operation. STXIT linkage to DL/I for abnormal task termination, normally controlled by UPSI bit 7, as always active under MPS operation.

Job Control Statements

The following information tells you how to set up the job control statments for the execution of your program.

Batch If data base changes are to be logged, either disk (batch only) or tape logging may be specified on the DL/I parameter statement with the LOG parameter. If the LOG parameter is omitted and UPSI byte bit 6 is 0, the default is tape logging.

If tape logging is used, ASSGN and TLBL statements as shown below are required. The log tape must have a standard label.

```
// ASSGN  SYS011,cuu
// TLBL   LOGOUT
```

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

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

The execution job stream must contain DLBL or TLBL statements that define the data base(s) that are to be processed. ASSGN and EXTENT statements are also required for SHSAM and HSAM data bases. When initally loading a data base, additional DLBL and EXTENT statements may also be required for system work files. Consult your data base administrator for the details.

The EXEC statement specifies the DL/I initialization and the SIZE parameter. Typically you will require a 512K virtual partition for execution with a SIZE parameter of 256K. See *VSE/Advanced Functions System Control Statements,* for details.

```
// EXEC   DLZRRC00,SIZE=xxxK
```

Shown below are illustrates the execution job control statements for a program INVUPDT with a PSB of INVMSTR. It is assumed that the updates to the data base will be logged and that HISAM is the access method for the data base.

The number of DLBL and EXTENT statements varies depending on the number of data bases accessed and the DL/I access method used. See *DL/I DOS/VS Resource Definition and Utilities* for more details.

The UPSI statement is optional and when set to all zeros, as shown, can be omitted.

If the application program does retrievals only, or if the UPSI byte is used to turn off data base logging, no log tape or disk is required.

```
//  JOB     UPDATE
//  UPSI    00000000
//  ASSGN   SYS011,182
//  TLBL    LOGOUT
//  DLBL    INVPRT1,'INVENTORY',99/365,VSAM
//  DLBL    INVPRTZ,'INVENTORY-OFLOW',99/365,VSAM
//  EXEC    DLZRRC00,SIZE=256K
DLI,INVUPDT,INVMSTR
            .
            .
            DATA CARDS IF REQUIRED
            .
            .
/*
/&
```

MPS Batch

The EXEC statement specifies the DL/I MPS initialization program and the SIZE parameter. Typically, you will require a 256K virtual partition for execution with a SIZE parameter of 256K. See *VSE/Advanced Functions System Control Statements,* for details.

```
//  EXECDLZMPI00,SIZE=xxxK
```

Shown below are the execution job control statements for a program INVUPDT with a PSB pf INVMSTR. For the MPS environment, data base logging is controlled in the CICS/VS partition.

The UPSI statement is optional and when set to all zeros, as shown, it may be omitted.

```
//  JOB     UPDATE
//  UPSI    00000000
//  EXEC    DLZMPI00,SIZE=256K
DLI,INVUPDT,INVMSTR
            .
            .
    DATA CARDS IF REQUIRED
            .
            .
/*
/&
```

MPS Batch Using MPS Restart

Shown below are the execution job control statements for the same program using MPS Restart. Included in this example are statements which assign a tape to contain checkpoint records written by VSE checkpoints.

```
//  JOB     UPDATE
//  MTC     REW,280
//  ASSGN   SYS100,280
//  EXEC    DLZMPI00,SIZE=256K
DLR,INVUPDT,INVMSTR
            .
            .
    DATA CARDS IF REQUIRED
            .
            .
/*
/&
```

The MPS Restart facility is invoked for an MPS batch job by using the DLR function code, in the parameter input to DL/L. This function code (see above example) replaces the DL/I function code used for normal batch and MPS batch jobs. The DLR function code must be used when the job is first started and not just when it is restarted.

Restarting an MPS Batch Program Using MPS Restart

The following steps are required to restart an MPS batch program after a failure:

1. Get the VSE checkpoint ID from the SYSLOG message.
   a. If the individual MPS batch job failed, a message containing the correct checkpoint ID for restart is issued by DL/I at the time of failure.
   b. If there was a system failure, the message is issued when MPS is started again in the online partition.
2. Use the VSE checkpoint ID on the VSE RSTRT job control statement. The RSTRT statement is used instead of the EXEC statement when the job is resubmitted for execution.

The job control statements in the following example will restart the program in the previous job control example from checkpoint 0010. Note that the jobname must be the same on the restart job as it was on the job that failed.

```
//  JOB     UPDATE
//  MTC     REW,280
//  ASSGN   SYS100,280
//  RSTRT SYS100,0010
DLR,INVUPDT,INVMSTR
            .
            .
    DATA CARDS IF REQUIRED
            .
            .
/*
/&
```

For additional information on the function, use, and restrictions of the VSE checkpoint/restart facility, see the*VSE/Advanced Functions Macro User's Guide*.

Restart Considerations

- If an MPS batch program using MPS restart does not issue a combined checkpoint before a failure, it must be started over from the beginning, using the EXEC job control statement rather than the RSTRT statement. For an individual job failure, this is indicated in the message issued at the time of failure. For a system failure, no message is issued for such jobs when MPS is started again in the online partition.
- The VSE restart facility requires the jobname to be the same on a restart job as it was on the job that failed. It also requires that the VSE partition start and end at the same addresses as when the job failed.
- During a VSE restart, a data check (tape checkpoint files) or end-of-file (disk checkpoint files) may occur if the checkpoint ID specified is greater than the actual number of checkpoints taken before the failure.
- On a restart, parameter input is ignored by DL/I, since the parameters were already read and saved when the job first started. However, if the parameter input statement was included on SYSIPT (instead of having been entered from SYSLOG) when the job first started, it is important that one also be included when the job is restarted. This is because DL/I will attempt to position SYSIPT past the parameter input statement when the job is restarted.

Chapter 2. DL/I Programming Reference Information

Definitions

A number of terms and phrases used in the detailed explanation and description of DL/I are either new to most readers, or have new meanings. To improve the readability and understandability of this and other DL/I DOS/VS publications, the significant and important terms and phrases are defined in the Glossary.

Call Functions

The following ten call functions may be used in a DL/I call:

'GUb' Get unique
'GHUb' Get hold unique

'GNb' Get next
'GHNb' Get hold next

'GNPb' Get next within parent
'GHNP' Get hold next within parent

'DLET' Delete

'REPL' Replace, update, or rewrite

'ISRT' Insert:
- Load a new data base
- Add to an existing data base.

'CHKP' Checkpoint

Because of the functional similarity between the GET calls with HOLD and the same calls without HOLD, the two types are grouped together in the following discussion. The only difference between them, in each case, is that the hold function permits the following call to be a REPLACE or DELETE call. This function is fully explained in the following sections on the REPLACE and DELETE calls. If a HOLD call is not immediately followed by a DELETE or REPLACE call, (that is, it is followed by a GET call without hold or by an INSERT call) the hold status is lost and must be reestablished by another HOLD call before a REPLACE or DELETE call can be processed.

To simplify the explanation, the two distinct uses of the INSERT call are treated separately in the following discussion. The processing option specified by the DBA for the PSB generation determines the use of the INSERT call in any particular program.

The detailed discussion of the call functions below is, in each case, given under the following five headings:

1. *Function* gives a brief description of what the call does.
2. *SSAs or SSAs and Keys* is a detailed explanation of the effect of inclusion or exclusion of the various types of qualified and unqualified SSAs on the function.

3. *Status Codes Returned* is a discussion of which status codes are returned in the PCB by DL/I as a result of each DL/I call, and what their significance is to the programmer.
4. *Position Pointer* tells of the effect, if any, on the position pointer that normally points to the next sequential segment as a result of a successful or an unsuccessful call.
5. *Parentage* describes the relationship of one segment to another in the next higher level of the hierarchy. It is the beginning point for the use of the get next within parent (GNP) or get hold next within parent (GHNP) functions.

For details of logical relationships and the RULES parameter, which are mentioned later in this chapter, refer to *DL/I DOS/VS Data Base Administration*.

GU (Get Unique)/GHU (Get Hold Unique)

1. *Function:* Randomly retrieves any sensitive segment in the data base. The GU and GHU calls are the only calls that can establish position backward in the data base.
2. *SSA*s:
   - May have no SSA or any number of SSAs up to the number of hierarchical levels defined in the DBD.
   - If no SSA is specified, the first segment (which is a root segment) in the data base is retrieved.
   - If a call of one unqualified SSA is specified, the first root segment in the data base is retrieved, or if multiple SSAs are specified, the root segment identified by the first SSA serves as the parent root segment for whichever child is subsequently retrieved.
   - If multiple SSAs are specified, the segment retrieved has a path of parents as defined by the SSAs.
   - If an unqualified SSA is specified, any occurrence of that segment type under its parent will satisfy the call.
   - If one or more SSAs are omitted from a multiple SSA path definition, implied SSAs are generated:
     – If the previous valid call established the same higher level path to a segment represented by the missing SSA that the call is using (either through missing parent SSAs or SSAs with the same qualification as was incorporated in the prior call), then the implied SSA is developed to position to the same segment as in the prior call.
     – If the paths are different, then the implied SSA is developed as an unqualified SSA.
3. *Status Codes Returned:*

   GE No segment was found that satisfies the SSAs specified.
   b The correct segment has been retrieved.
   Ax For all other status codes the program should probably be terminated. (See “Status Code Summary” in this chapter for a complete list of all status codes.)
4. *Position Pointer:* After a successful GU or GHU call, the position pointer points to the next logical segment in the data base. If the call was unsuccessful, the

position pointed to is unpredictable and remains so until the next successful call.

5. *Parentage:* Parentage is set to the segment retrieved for any subsequent GNP call. If the call was unsuccessful, the previous parentage (if any) is destroyed.

GN (Get Next)/GHN (Get Hold Next)

1. *Function:* Sequentially retrieves the next sensitive segment in a forward direction in the data base, with or without the specification of one or more qualified or unqualified SSAs. Sequential retrieval within a hierarchy always proceeds from top to bottom, and from left to right.

2. *SSAs:* With no SSAs specified, the next sequential sensitive segment is retrieved. The next sequential segment is the one that the position pointer was pointing to at the completion of the last successful call.

   Note: In HDAM, root segments are returned in physical sequence of data base records rather than in root key sequence unless the randomizing module was designed to maintain key sequence. If not, input or output sorting or secondary indexing can be used if key sequencing is required.

   - If a single, unqualified SSA is specified, the first occurrence of that segment type found by searching in a forward direction from the current position is retrieved.
   - If one or more SSAs are specified, the path leading to the segment retrieved is as defined by the SSAs. The last specified SSA always defines the segment type that is retrieved.
   - When unqualified SSAs are specified or when they are omitted completely in a multiple SSA GN or GHN call, their presence or absence has no effect on the operation. Only qualified SSAs plus the last SSA specified are used by DL/I to determine the path and to retrieve the segment. In other words, unspecified or unqualified SSAs for higher level segments in the hierarchy indicate that *any* high level segment that is the parent of the correct lower level specified or qualified segment will satisfy the call. It is suggested that SSAs be specified however, because of the documentation, control, and future change implications; these topics are discussed under “General Programming Techniques and Suggestions” later in this chapter.
   - When the last specified SSA is qualified, it defines, in the sense of the relational operator, the unique segment that is to be retrieved. Higher level qualified SSAs, in the same way, define the unique segments that are to become a part of the path to the segment being retrieved.

3. *Status Codes Returned:*

   GA A segment has been retrieved that is at a higher level in the hierarchy than the previous segment retrieved for a call without SSAs. GA serves as a warning that the position in the data base has changed with respect to the path that existed previously.

   GB The end of data base has been reached. No segment is retrieved. If, however, the GN call specified a non-Boolean qualified root level SSA with the segment field name referring to a key field (not a data field) and the relational operator <, <=, or =, and the end of the data base was reached without locating the segment, the status code is GE and not GB. A Boolean qualified GN will result in a GB status code when the minimum key value is greater than the maximum key value. Once the end of the

data base is reached, processing can continue by issuing a GU call, or by issuing any form of the GN call. However, if the status code is GE at the end of the data base, the following GN call should be unqualified to allow DL/I to search from the beginning of the data base. The GN call starts searching again at the beginning of the data base for the next segment that satisfies the conditions of the call. In this situation much time can be wasted owing to the need to search through the whole data base.

GE The segment specified by one or more qualified SSAs was not found. A Boolean qualified GN will result in a GE status code when the minimum key value is less than the maximum key value. This could be because the segment does not exist, that the segment specified cannot be found by searching forward from the previous position established, or that the current position pointer is pointing to a segment that is forward in the data base of any possible existence of the specified segment. This status code is also returned for a programming error, namely that of specifying higher level qualified or unqualified SSAs for segments that differ from one or more of the segments in the currently established parentage path.

GK A segment has been retrieved that has the same higher level path as the previous segment processed but it is a different type of segment at the same hierarchical level. This status code is returned only as a warning for calls with no SSAs specified and merely indicates that the program is working with a different segment type.

bb Correct segment has been retrieved.

Ax For all other status codes the program should probably be terminated. Status code AC, however, could indicate problems with keys and prior positioning when specifying multiple SSAs of different types. (See “Status Code Summary” in this chapter for a complete list of all status codes.)

4. *Position Pointer:* Following a GN or GHN call in which a segment was retrieved, the position pointer points to the next segment in the data base. If the GB status code (end of data base) was returned, it points to the first segment in the data base. Following an unsuccessful call, the position pointed to is unpredictable and remains so until the next successful call.
5. *Parentage:* Parentage is set to the segment retrieved for any subsequent GNP call. If the GN or GHN call was unsuccessful, the previous parentage (if any) is destroyed.

GNP (Get Next Within Parent)/GHNP (Get Hold Next Within Parent)

1. *Function:* Sequentially retrieves the next sensitive segment in a forward direction within established parentage, with or without the specification of one or more qualified or unqualified SSAs. Parentage must have been established by a successful GU, GHU, GN, or GHN call either immediately before this call, or at some prior time, provided no other call that changes parentage has intervened.
2. *SSAs:*
   - With no SSAs specified, the next sequential sensitive segment within the established parentage path is retrieved. The next segment is the one that the position pointer was pointing to at the completion of the last successful call.

- If a single, unqualified SSA is specified, the first occurrence of that segment type found by searching in a forward direction within the established parentage path is retrieved.
- If one or more SSAs are specified, the path leading to the retrieved segment is as defined by the multiple SSAs.

  The SSAs specified must be for segments at levels lower than the previously established parentage. The last-specified SSA always defines the segment type that is searched for.
- When unqualified SSAs are specified in a call with multiple SSAs, they establish the first occurrence of the associated segment type as a part of the path.
- Any missing SSAs between the previously established parentage and the segment type to be retrieved (as specified in the last specified, or only, SSA) are generated as unqualified SSAs. This has the effect of establishing the first occurrence of the segment representing the missing SSAs as a part of the path.
- When an SSA used in a call is qualified, it specifies, in the sense of the relational operator, the unique segment that is to be retrieved, or the unique segment that is to become a part of the path to the actual segment being retrieved.

3. *Status Codes Returned:*

   GA A segment has been retrieved, for a call without SSAs, that is at a higher level in the hierarchy than the previous segment retrieved, but still below the parentage previously established. This code serves as a warning that the position in the data base has changed with respect to the path that existed previously from the established parentage to the last retrieved segment.

   GE The segment specified by one or more qualified or unqualified SSAs within the previously established parentage was not found by searching forward from the established position. For calls with or without SSAs, the segments beneath the established parentage have been exhausted without retrieving a segment.

   GK A segment has been retrieved that is within the previously established parentage and has the same higher level path as the previously processed segment, but it is a different type of segment at the same hierarchical level. This status code is returned only for calls with no SSAs specified. This status code merely indicates that the program is working with a different segment type.

   GP There has been no parentage established, or the segment specified in the SSA is at a level in the hierarchy that is equal to or higher than the lowest level that exists in the currently established parentage. The following could be reasons for no parentage being established:

   - No GU or GN calls have been issued to the data base.
   - A GU call has been unsuccessful.
   - The last established parent has just been deleted.
   - These are probably application program problems.

bb The correct segment has been retrieved.

Ax For all other status codes the program should probably be terminated. (See “Status Code Summary” in this chapter for a complete list of all status codes.)

4. *Position Pointer:* If a segment was retrieved, the position pointer points to the next segment in the data base. If the GE (no segment found) status code was returned, it points to the segment that caused the no-segment-found condition to be recognized. This is either a segment with a higher key than was specified in one of the qualified SSAs, or is the first segment located outside of the established parentage in a forward direction.
5. *Parentage:* Parentage is not established or changed.

DLET (Delete)

1. *Function:* Deletes the last segment successfully retrieved from the data base by a GHU, GHN, or GHNP call and also deletes all of that segment's physically dependent segments or children at all levels beneath it, *whether they are sensitive segments or not*. The previous GET call must immediately precede the DLET call and must be a HOLD call, or the DLET call is rejected. The previously retrieved segment and its dependent segments are removed from the data base and cannot be retrieved or used as parents again.
2. *SSAs:* No segment search arguments (SSAs) are required. An unqualified SSA is allowed to select the segment to be deleted from the path of segments just retrieved using a path command code call (see Chapter 4). If only a single segment was retrieved, an unqualified SSA is permitted to delete that segment. In all other cases, an unqualified SSA gives a status code of DJ. A qualified SSA gives a status code of AJ. When a Delete call with multiple SSAs is issued, DL/I deletes the segment in the first SSA only. Only validity checking is performed on the remaining SSAs.
3. *Status Codes Returned:*

   bb The segment and all of its dependent segments (if any) have been successfully deleted.

   Other For all other status codes the program should probably be terminated. However, two status codes are of special interest because they represent special editing and checking procedures that are performed for the DLET call:

   DJ No successful GET HOLD call immediately preceded the DLET call.

   DX The physical delete rule has been violated. If the segment is a logical parent, it still has active logical children. If the segment is a logical child, it has not been deleted through its logical path.

   (See “Status Code Summary” in this chapter for a complete list of all status codes.)
4. *Position Pointer:* The current position pointer is not changed by an unsuccessful DLET call. After a successful deletion, the pointer may continue to point to the next segment in the data base that follows the deleted segment or, if dependent segments were also deleted, to the segment that follows the last dependent segment deleted.

5. *Parentage:* Parentage is not changed by a DLET call unless the last established parent is deleted; if so, parentage must be reestablished.

REPL (Replace, Update, or Rewrite)

1. *Function:* Replaces, updates, or rewrites the last segment successfully retrieved from the data base by a GHU, GHN, or GHNP call. The previous retrieval call must immediately precede the REPL call, and must be a HOLD call, or the REPL call is rejected.
2. *SSAs:* No segment search arguments (SSAs) are required. Unqualified SSAs are permitted when replacing a segment or segments retrieved using a path command code call. If only a single segment was retrieved, an unqualified SSA is permitted to replace that segment. In all other cases, an unqualified SSA gives a status code of DJ. A qualified SSA gives a status code of AJ.
3. *Status Codes Returned:*

   bb Segment has been replaced successfully.

   NI Segment being replaced contains a secondary index source field which is a duplicate of one already reflected in the secondary index. The segment was not replaced in the data base.

   Other For all other status codes the program should probably be terminated. However four status codes are of special interest because they represent special editing and checking procedures that are performed for the REPL call:

   DA Segment key field has been changed.

   DJ No successful GET HOLD call immediately preceded the REPL call.

   RX The physical replace rule has been violated. The physical rule was specified for the destination parent and an attempt was made to change its data. When a destination parent has the physical rule it must be replaced through the physical path.

   V1 Invalid length for a variable length segment. (See “Status Code Summary" in this chapter for a complete list of all status codes.)
4. *Position Pointer:* The current position pointer is not changed by either a successful or an unsuccessful REPL call. It continues to point to the next segment in the data base following the replaced segment.
5. *Parentage:* Parentage is not changed by a REPL call.

ISRT (Load A New Data Base)

1. *Function:* A new data base is created with ISRT calls to load segments from the I/O area after they have been built there. The PSB generated for each individual program by the DBA determines if the ISRT is for new data base loading or for adding to an existing data base.
2. *SSAs and Keys:*

   - The ISRT call for each segment being loaded must have specified at least one unqualified SSA that defines the type of segment.
   - Before the segment load call is given, the segment must be built in the I/O area and, if it has a key, its correct key must be placed in the proper location in the I/O area.

- When loading a logical child segment, the I/O area must contain the logical parent's concatenated key, followed by the logical child segment.
- All segments with keys must be loaded in sequence by key.
- Segment types, as specified by their SSAs, may not be loaded out of hierarchical order. Their parents must have been loaded before they can be loaded.
- Multiple SSAs are permitted in a load operation, although they serve no function. If multiple SSAs are present, the path defined by the multiple SSAs is checked and must be a valid path, or the call is rejected. The last SSA in a group of multiple SSAs must be unqualified.
- If segments without keys are loaded, they are loaded in the data base in the same order in which their ISRT calls are processed.

3. *Status Codes Returned:*

   LB The segment already exists in the data base; this was an attempt to load a duplicate segment. This status code is only associated with segment types with key fields.

   LC The segment being loaded is out of sequence by key.

   LD No parent exists for this segment: this status code usually indicates that the segment types being loaded are out of sequence.

   LE When multiple SSAs are specified in a load call, the segment types specified by the multiple SSAs are out of sequence.

   NI Segment inserted contains a secondary index source field which is a duplicate of one already reflected in the secondary index. The segment just inserted should be deleted. (Expect an NE status code on the DLET call.)

   NO Segment being loaded is a duplicate of the secondary index source segment.

   V1 Invalid length for a variable length segment.

   ␣␣ Segment was loaded successfully.

   Other For all other status codes the program should probably be terminated. (See “Status Code Summary'' in this chapter for a complete list of all status codes.)

4. *Position Pointer:* In a load operation, the position pointer always points to the next available space following the last segment successfully loaded. The next segment loaded is placed in that space.
5. *Parentage:* Since only ISRT calls can be issued to a data base being loaded, parentage has no significance and is not set.

ISRT (Add To An Existing Data Base)

1. *Function:* A new segment is randomly or sequentially inserted or added to an existing data base from the I/O area after it has been built there. The PSB generated for each individual program by the DBA determines if the ISRT is to add to an existing data base or to load a new data base.
2. *SSAs and Keys:*
   - For each segment being added, at least one unqualified SSA must be specified that defines the type of segment being added.
   - Before the ISRT call is issued, the segment being inserted must be built in the I/O area and, if it has a key, its correct key must be placed in the proper location in the I/O area.
   - If the segment being inserted is a root segment, it is inserted in its correct place in the data base as determined by its key taken from the I/O area.
   - If the segment being inserted is not a root segment, but its immediate parent has just been inserted, it too may be inserted as soon as it is built in the I/O area by merely specifying its unqualified SSA in the ISRT call.
   - When inserting a logical child segment through the physical path, the I/O area must contain the logical parent's concatenated key, followed by the logical child segment.
   - When inserting a logical child segment through the logical path, the I/O area must contain the physical parent's concatenated key followed by any logical child data.
   - If only one unqualified SSA is specified for the segment being inserted, the position pointer must point to the right place in the data base so that the segment's logical location in the data base can be found by searching forward or backward in the current record.
   - Segments that have no keys are always inserted following the last segment of the same type unless the RULES parameter for the DBD generation for this data base states FIRST or HERE, in which case the stated rule applies. Segments having equal keys are also treated in this manner.
   - If a complete set of qualified SSAs is specified for the segment being inserted, it is inserted at its proper location in the data base.
   - If multiple SSAs are specified for an insertion, they may be a mixture of qualified and unqualified SSAs. However, the last SSA must be unqualified.
   - If unqualified SSAs are used to establish a path to an insertion, the first occurrence of the segment type satisfies the SSA, providing the higher level path to it is correct.
   - If SSAs are omitted, the current position in the data base is used to develop implied SSAs. If the current position in the data base is not correct because higher level SSAs have changed the position in the data base, then implied unqualified SSAs are developed for the first occurrence of the segment type that falls within the newly established path.

3. *Status Codes Returned:*

   GE The path specified with multiple SSAs was not found. This status code is normally associated with one or more qualified SSAs that define the path to the insertion. However, it can be returned when only unqualified SSAs are used, if no segment of the parent segment type exists.

   II The segment already exists. This is generally a duplicate segment indication, although it could occur if an ISRT call were issued for a segment without first establishing the proper path. The segment could possibly match a segment with the same key in another hierarchy or record.

   IX An attempt was made to insert either a logical child segment or a concatenated segment. In the case of the logical child segment, the corresponding logical/physical parent segment does not exist. In the case of the concatenated segment, either the insert rule was physical and the logical/physical parent does not exist, or the insert rule is virtual and the key of the logical/physical parent in the I/O area does not match the concatenated key of the logical/physical parent.

   NI Segment inserted contains a secondary index source field which is a duplicate of one already reflected in the secondary index. The segment just inserted should be deleted. (Expect an NE status code on the DLET call.)

   V1 Invalid length for a variable length segment.

   bb The segment was inserted in the proper place as specified by the given or implied SSAs.

   Other For all other status codes the program should probably be terminated. (See “Status Code Summary'' in this chapter for a complete list of all status codes.)

4. *Position Pointer:* After a successful ISRT call, the position pointer points to the segment immediately following the segment just inserted. This could be another segment of the same type if the segment was inserted in the middle of a group of the same segment types by its key, or to the next segment type in the hierarchy if the segment was inserted as the last segment of the group.

5. *Parentage:* Previously established parentage is not changed or affected providing the segments being inserted are inserted below the lowest level parent in the established parentage path. If a segment is inserted outside the currently established path, then parentage is destroyed and must be reestablished before GNP calls can be issued.

CHKP (Checkpoint)

1. *Function:* For DL/I batch application programs, the CHKP call causes a checkpoint record to be written on the DL/I log as an aid in restart processing. The data base buffers will be written to secondary storage and a checkpoint log record, with a user-supplied unique checkpoint identification, will be written to the DL/I log. CHKP can be issued at any appropriate points in a program as determined by the programmer and can be issued by batch, MPS, or DL/I online tasks.

   See the section “RQDLI Commands for DB Access” in Chapter 1 for details concerning RPG II.

   In case of a failure in a batch environment, DL/I the backout utility may be run to back out data base changes occurring since the last checkpoint. The utility will not back out data base changes committed before the last checkpoint. If there are no checkpoint records on the log for the PSB of interest, backout will proceed back to the scheduling record for this PSB.

   For MPS and online tasks running with CICS/VS journalling active, a CHKP call is in effect a CICS/VS sync point call with the exception that the task's PSB scheduling status is not changed.

   Therefore, if the task has a scheduled PSB in effect at the time the CHKP call is issued, a PSB scheduling call is not required after the CHKP call. In fact, a scheduling call issued under these circumstances would cause a scheduling error. (See “Checking the Response to a DL/I Call in a CICS/VS Environment" in Chapter 3.)

   It is recommended that DL/I logging not be used for MPS and online tasks. With DL/I logging active, a CHKP call causes data base buffers to be written to secondary storage and a checkpoint log record to be written to the DL/I log, as in the batch environment. However, these functions are not usable for performing backout because batch backout cannot be used in an online environment. Backout for an MPS or online DL/I task can only be performed using CICS/VS dynamic transaction backout, which requires that CICS/VS journalling be active.

   The I/O area, when used for a CHKP call, is the area from which DL/I obtains the eight-character checkpoint identification. (See the format of the DL/I call under DL/I Batch Program Call in Chapter 1.) The 8-character checkpoint identification may be any value. However, an EBCDIC character string is recommended because the value is used in messages to the operator and/or programmer. For batch and MPS tasks, the message issued by DL/I each time an application program issues a CHKP call, DLZ105I, contains the checkpoint identification, which should be noted and saved, because it may be required to aid in backout and restart processing. For online tasks, DLZ105I is not written. Therefore, online application programs must include their own provisions for identifying where restart is to begin following a CHKP call.

   DL/I provides a facility to restart MPS batch programs. This facility supports the use of VSE checkpoint/restart with DL/I checkpoints. For additional information on this facility, see “MPS Restart Facility.”

2. *SSAs*: SSAs have no effect during the CHKP call.

3. *Status Codes Returned:*

   XD An error occurred when the data base buffers were being written to secondary storage. When using MPS Restart, this status code will not be issued; instead, error message DLZ131I will be issued and the program will be abnormally terminated.

   XH Data base logging is not active.

   XR A combined checkpoint failure occurred during MPS Restart processing. This status code is issued when a VSE checkpoint is not taken before a DL/I CHKP call, when the VSE checkpoint prior to a DL/I checkpoint fails, or when a CICS/VS temporary storage error occurs during DL/I CHKP processing.

   bb A checkpoint was successfully taken.

4. *Position Pointer:* Current position in the data base is lost upon return from a CHKP call. If the PSB being used has more than one PCB defined, position is lost across all PCBs and not just the PCB specified in the checkpoint call. Position must be reestablished before issuing GET NEXT calls after a CHKP call unless you wish to restart at the beginning of the data base.

5. *Parentage:* Parentage must be reestablished after a CHKP call.

MPS Restart Facility

In order to provide a restart capability for MPS batch users, DL/I supports the use of VSE checkpoint/restart with DL/I checkpoint calls. A VSE checkpoint writes a copy of the partition in which the checkpoint was issued to disk or tape. The VSE RSTRT job control statement reloads this copy into the partition and passes control to the restart address that was specified when the VSE checkpoint was issued. For COBOL and PL/I programs using their respective VSE checkpoint interfaces, this corresponds to the next program statement after the one which invokes the VSE checkpoint.

MPS Restart provides the following functions:

- Combined Checkpoint Verification
  MPS Restart verifies that a VSE checkpoint is issued immediately before each DL/I CHKP call. This is called a combined checkpoint. A VSE checkpoint may be issued in PL/I and COBOL by using the checkpoint interfaces provided by those languages. It is recommended that the checkpoint ID returned by a VSE checkpoint be used as the checkpoint ID on the following DL/I CHKP call in PL/I and Assembler language programs. This is not possible in COBOL because the returned checkpoint ID is not available to the application program. Using the VSE checkpoint ID on the DL/I CHKP call provides a cross reference between the VSE and DL/I checkpoint messages issued to SYSLOG.

  See the “Program Examples" section for examples on how to code combined checkpoints.

- MPS Batch Reinitialization
  The first DL/I call performed following a VSE restart must be a DL/I CHKP call. This will be the normal sequence when VSE checkpoints are placed immediately before each DL/I checkpoint. DL/I checkpoint processing will automatically determine that a VSE restart has occurred and reinitialize the MPS batch environment. Following a successful reinitialization, control will be returned from the CHKP call to the application program as if from a normal checkpoint and the program may continue processing.

- Checkpoint ID Notification and Verification
  Besides the normal SYSLOG messages issued by VSE and DL/I when checkpoints are taken, a message containing the checkpoint identifier (ID) of the last successful combined checkpoint will be issued when a failure occurs. For individual job failures, the message is issued at the time of the failure. For system-wide failures, it is issued when MPS is started again in the online partition. The checkpoint ID contained in this message must be specified as a parameter on the VSE RSTRT job control statement when restarting an MPS batch job. MPS Restart will verify whether the checkpoint ID used for restart is the correct checkpoint ID. If it is not, DL/I will indicate the correct checkpoint ID which must be used and cancel the job, allowing you to restart the job, using the correct checkpoint ID.

  For additional information on using the MPS Restart facility, see *DL/I DOS/VS Data Base Administration*.

Restrictions on Using VSE Checkpoint/Restart

Certain restrictions exist on the use of VSE checkpoint/restart which you should be aware of:

- VSAM files must be closed before a VSE checkpoint is issued (DL/I data bases used by MPS programs are in the online partition and are not affected by this restriction).
- RPG II does not support a VSE checkpoint interface. RPG II users will have to write their own interface in another programming language (Assembler language, for example) if the MPS Restart facility is to be used with RPG programs.
- VSE Restart cannot be used to restart programs that failed because of program logic errors. This is because a copy of the program, exactly as it was before it failed, is loaded back into the partition during a restart.

  For details on these and other restrictions, see *VSE/Advanced Functions Macro Reference*, SC24-5211, and *VSE/Advanced Functions Macro User's Guide*, SC24-5210.

General Programming Techniques and Suggestions

1. In general, all calls should contain qualified SSAs whenever applicable and wherever possible.
2. Do not omit SSAs, qualified or unqualified, in a multiple SSA call if it can be avoided. This promotes flexibility and control as the application and the data base grow or change:
   - If the hierarchical structure of the data base is changed, the specification of all SSAs ensures the integrity of the program's access to the newly structured data base.
   - If use of the multiple positioning feature is added to the program at a later time, full specification of all SSAs would then be required.
   - Specifying all SSAs is a sound programming practice from a documentation and debugging point of view.
3. Try not to construct the logic of the program and the structure of the calls in a manner that is heavily dependent on the hierarchical structure of the data base.
4. In a call parameter list, the parameter count, if specified, determines how many parameters are actually used. This allows a common call routine to be written for several types of calls, and the count field can be varied for each call issued.
5. It is usually a better programming practice to use qualified SSAs to retrieve the segments required in the application instead of requesting the DBA to incorporate sensitivity in the PSB to eliminate segments not required.
6. It is possible to construct SSAs in a GN call that could force DL/I to search to the end of the data base without retrieving any segment. This could be especially true during program testing. If a large, multivolume data base is being accessed, such an operation could waste a significant amount of processing time. To prevent this waste, use the GNP or GU calls wherever possible or practical, especially when using qualified SSAs with relational operations other than the equal operator.
7. When using multiple SSAs in a call, provide an SSA for the root segment qualified for the key field and using the equal operator wherever possible, so that unnecessary and time consuming processing of the entire data base can be avoided. The same applies when secondary indexing (see Chapter 4) is used, except that the qualification must be on the indexed field and not the key field.
8. The use of GU (as opposed to GN) will probably provide more flexibility for future application program and data base change.
9. The GU call can be used when processing a data base sequentially, although it usually requires more processing time than a comparable GN.
10. Remember that the GN call can retrieve a segment from a different logical record than was pointed to prior to issuing the call. A GA status code is returned only if no SSA was specified. Sometimes it is good practice to save the original root key and make a comparison after a GN call.
11. A repeated GN call with one unqualified SSA retrieves all occurrences of that segment type in the data base until the end is reached.
12. Remember that parentage cannot be set or reset with a GNP call.

13. Since the GET HOLD calls do not increase processing time, and since the use of a REPL or DLET call following a GET HOLD is optional, it is sometimes more convenient programming to issue all GET calls with the HOLD option.
14. Remember that the deletion of a parent also deletes all of its children in the same call, even if they are not sensitive. This is the only nonpath call that affects multiple segments. If any information is required from those children, they must be retrieved before the parent is deleted.
15. The ISRT call to load a data base initially is only used once to build the data base. Thereafter, the “loading” of additional segment types is accomplished through use of the ISRT (add) call.
16. Most comprehensive data bases are loaded in stages by segment type or by groups of segment types. This requires multiple programs to accomplish the loading. All programs after the first are actually add-type programs and may have serious performance problems. These add-type programs used in a data base loading procedure must be coordinated and reviewed with the DBA to ensure that they perform adequately through the use of the various facilities available to the DBA.
17. Consider loading or building the data base initially as a sequential data base and having the DBA convert it to its ultimate access method once it is built.
18. Of all the calls that should be fully qualified, the ISRT (add) is the most important. If not fully qualified, the ISRT call could insert the segment in a position completely different from that which was originally intended by the programmer.

Problem Determination

The following is a brief discussion of steps that you, as an application programmer, can take when your program fails to run, abnormally terminates, or gives incorrect results.

Initialization Errors

Before your program receives control, DL/I must have correctly loaded and initialized a nucleus and control blocks. If you suspect a problem in this area, consult your system programmer or data base administrator. There are aids available to them that will help them to determine if a problem does exist and to isolate it. Check to see whether there have been any recent changes to DBDs, PSBs, and the control blocks generated from them.

Execution Errors

If initialization errors do not seem to be present, you should check the following:

1. The output from the compiler.
   - All error messages should be resolved.
2. The output from the linkage editor.
   - Are all external references resolved?
   - Have all necessary modules been included?
   - Was the Language Interface module (DLZLI000) correctly autolinked?
   - Is the correct entry point specified?
3. Your job control statements.
   - Is the information correct that describes the files that contain the data bases? See your data base administrator.
   - Are you using the UPSI bit settings correctly?
   - Have you included the SIZE parameter in the EXEC statement and is its value large enough to include your program?
   - Have you included a DLI parameter statement in the correct format?
4. Your program.
   - Are the fields in your PCB masks correctly declared?
   - Have you saved and restored registers correctly if using Assembler Language?
   - Does register 1 point to a fullword parameter list prior to issuing a DL/I call in Assembler Language?
   - If using a high-level language, are the literals you are using for arguments in DL/I calls producing the results you expect? For instance, in PL/I, is the parameter count being generated as a halfword rather than a fullword, and is the function code producing the required four-byte field?

- If you need help in producing and interpreting a dump, see your system programmer.
- Make full use of the information in the PCB if your program is producing incorrect results. For more detailed information about the status codes, see the status code summary below.

Status Code Summary

After processing a DL/I call, or RQDLI command in RPG II, control is returned to the application program at the next sequential instruction following the call. DL/I places a status code in the status code field of the PCB.

Figure 2-1 on page 2-18 provides a list of DL/I status codes and is given as a quick reference for the DL/I application programmer. These status codes are discussed in more detail below.

STATUS CODE	DATA BASE CALLS											DESCRIPTION
	GU GHU	GN GHN	GNP GHNP	DLET	REPL	ISRT (LOAD)	ISRT (ADD)	CHKP	CALL COMPLETED	ERROR IN CALL or CONVERSION	I/O or SYSTEM ERROR	
AB	X	X	X	X	X	X	X			X		SEGMENT I/O AREA REQUIRED, NONE SPECIFIED IN CALL
AC	X	X	X			X	X			X		HIERARCHICAL ERROR IN SSAs
AD										X		INVALID FUNCTION PARAMETER
AH						X	X			X		CALL REQUIRES SSAs, NONE PROVIDED
AI	X	X	X	X	X	X	X				X	DATA MANAGEMENT OPEN ERROR
AJ	X	X	X	X	X	X	X			X		INVALID SSA QUALIFICATION FORMAT OR COMMAND CODE
AK	X	X	X			X	X			X		INVALID FIELD NAME IN CALL
AM	X	X	X	X	X	X	X			X		CALL FUNCTION NOT COMPATIBLE WITH PROCESSING OPTION OR SEGMENT OR PATH SENSITIVITY
AO	X	X	X	X	X	X	X				X	I/O ERROR
DA					X					X		SEGMENT KEY FIELD HAS BEEN CHANGED
DJ				X	X					X		NO PRECEDING SUCCESSFUL GET HOLD CALL
DX				X						X		VIOLATED DELETE RULE
GA		★	★						X			CROSSED HIERARCHICAL BOUNDARY INTO HIGHER LEVEL (RETURNED ONLY ON CALLS WITH NO SSA SPECIFIED)
GB		★										END OF DATA SET, LAST SEGMENT REACHED
GE	★	★	★				★					SEGMENT OR PARENT SEGMENT NOT FOUND
GK		★	★						X			DIFFERENT SEGMENT TYPE AT SAME LEVEL RETURNED (RETURNED ON UNQUALIFIED CALLS ONLY)
GP			X							X		A GNP CALL AND NO PARENT ESTABLISHED, OR REQUESTED SEGMENT LEVEL NOT LOWER THAN PARENT LEVEL
II							★					SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE OR IS NON-UNIQUE
IX							X			X		VIOLATED INSERT RULE
KA	X	X	X	X	X	X	X			X		NUMERIC TRUNCATION ERROR DURING CONVERSION
KB	X	X	X	X	X	X	X			X		CHARACTER TRUNCATION ERROR DURING CONVERSION
KC	X	X	X	X	X	X	X			X		INVALID PACKED/ZONED DECIMAL CHARACTER DURING CONVERSION
KD	X	X	X	X	X	X	X			X		TYPE CONFLICT DURING CONVERSION
KE					X					X		REPLACE VIOLATION
LB						★						SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE OR IS NON-UNIQUE
LC						★						KEY FIELD OF SEGMENTS OUT OF SEQUENCE
LD						★						NO PARENT FOR THIS SEGMENT HAS BEEN LOADED
LE						★						SEQUENCE OF SIBLING SEGMENT NOT THE SAME AS DBD SEQUENCE
NA					X					X		DATA IN SEARCH OR SUBSEQUENCE FIELD HAS BEEN CHANGED
NE				X	X		X		X		X	INDEX MAINTENANCE CANNOT FIND SEGMENT
NI				X	X	X	X				X	INDEX MAINTENANCE UNABLE TO OPEN INDEX DATA BASE
					X	X	X			X		DUPLICATE KEY FOUND FOR INDEX DATA BASE
NO				X	X	X	X				X	I/O ERROR
						X						INSERTION OF DUPLICATE SECONDARY INDEX POINTER SEGMENT
RX					X					X		VIOLATED REPLACE RULE
V1					X	X	X			X		INVALID LENGTH FOR VARIABLE LENGTH SEGMENT
XD								X				ERROR DURING DATA BASE BUFFER WRITE OUT
XH								X				DATA BASE LOGGING NOT ACTIVE
XR								X				ERROR DURING CHECKPOINT PROCESSING FOR MPS RESTART
bb	★	★	★	★	★	★	★	★	★			CALL COMPLETED SUCCESSFULLY

★ Indicates status code that could be expected as normal situation.

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

Figure 2-1. DL/I Status Codes

Two categories of status codes are listed in the table. The circled status codes indicate situations that could be expected to occur during the execution of an application program. The circled status codes should be treated as normal situations rather than as errors. The results of the call are perfectly valid but the data retrieved may not be what the application program expects. Immediately following the call statement the application program should test for those circled status codes that apply to the function just requested. See Figure 2-2 on page 2-25, Figure 2-3 on page 2-26, and Figure 2-4 on page 2-27 for examples.

To handle all of the other status codes, it is recommended that a standard error routine be made available by the DBA, that will print as much information as possible prior to termination of the program. This would include all fields in the PCB, the I/O area, etc. Most of the status codes in this category are usually encountered during the debugging of an application program. The standard error routine could be included in each program using COBOL COPY, PL/I %INCLUDE, or /INSERT in RPG II (see Appendix A).

The following list gives more detailed information on each individual status code.

Note: When the first character of any of the following status codes is an N, this indicates that an error occurred during an internal DL/I call issued by indexing maintenance.

AB **Error in call.**

Explanation: On a data base call, segment I/O area is required but was not specified in the call.

Action: Correct program.

AC **Error in call.**

Explanation: SSA(s) contains an error in hierarchical sequence.

Possible causes:

1. No segment name equal to that specified in SSA found within scope of this PCB.
2. Level at which this SSA appears is out of sequence with that specified by the PCB.
3. Two segments of the same level are specified in the same call.

Action: Correct program.

AD **Error in call.**

Explanation: An invalid function parameter was supplied.

Action: Correct program.

AH **Error in call.**

Explanation: No SSA(s) was specified in call. Call type requires at least one SSA; none was specified.

Action: Specify SSA in call. Correct program.

AI **I/O, system, or user error.**

Explanation: Open error.

Possible causes:

1. Error in job control statements.
2. Data base being opened for other than load mode and has not been loaded.

Action: Check job control statements; ensure that filename is the same as the name specified on the DATASET card of the DBD. The segment name area in the PCB has the filename of the file which could not be opened.

AJ **Error in call.**

Explanation: SSA qualification format was invalid.

Possible causes:

1. Invalid relational operators
2. Missing right parenthesis
3. DLET call has qualified SSAs
4. REPL call has qualified SSAs
5. ISRT call has the last SSA qualified
6. Field in the SSA has the wrong length
7. Invalid command codes
8. Invalid Boolean connector.
9. Call interface SSA is longer than 304 bytes is being used in the ISC environment.
10. Column 9 of the SSA contains a value other than a blank, '*', or '('.
11. Path insert with logical child.

Action: Correct program.

AK **Error in call.**

Explanation: Possible causes are: the field name in the SSA parameter in a call statement does not correspond to any field name specified in the DBD, or if using boolean qualification, an invalid boolean operator was used.

Action: Correct program.

AM **Error in call.**

Explanation: Call function not compatible with processing option or segment sensitivity.

Action: Correct program, PSB, or system definition.

Possible causes:

1. Processing option of L and call function is not insert.
2. DLET, REPL, or ISRT call without corresponding segment sensitivity.
3. DLET or ISRT call for the root of a secondary data structure or any of its physical parent segment types in the physical data base.
4. A secondary index is processed as a data base itself and attempts are made to delete or insert a segment or to replace system-maintained fields.
5. Command code D used for a path retrieval call without path sensitivity.
6. An attempt was made to unload a secondary index using the HD unload utility.

AO **I/O error.**

Explanation: There is a SAM or VSE/VSAM I/O error.

Action: Check and correct.

DA **Error in call.**

Explanation: Segment key field has been changed on a REPL call.

Action: Correct.

DJ **Error in call.**

Explanation: No previous successful GET HOLD call.

Action: Check and correct.

DX **Error in call.**

Explanation: Violated delete rule. Review delete rule in SEGM statement of DBDGEN in the *DL/I DOS/VS Resource Definition and Utilities*.

Action: Correct the program.

GA **Call is completed.**

Explanation: Crossed hierarchical boundary into higher level. (See “Call Functions" earlier in this chapter.) This status code is returned on unqualified GN calls only.

Action: None required.

GB **Call is not completed.**

Explanation: This is the end of the file; the position beyond the last segment is reached.

Action: None required.

GE **Call is not completed.**

Explanation: Segment has not been found (GET call). For an ISRT call, the parent of the segment to be inserted was not found.

Action: None required.

GK **Call is completed.**

Explanation: Different segment type at same level returned. This status code is returned on unqualified GN calls only.

Action: None required.

GP **Error in call.**

Explanation: No parent established (or parent deleted) for this GNP call, or the requested segment level is not lower than the parent level.

Action: None required.

II **Call is not completed.**

Explanation: The segment that the user tried to insert already exists in the data base.

Possible causes:

- A segment with an equal physical twin sequence field already exists for the parent.
- A segment with an equal logical twin sequence field already exists for the parent.
- A logical parent has a logical child pointer, but the logical child does not have a logical twin pointer and the segment being inserted is the second logical child for the logical parent.

- A physical parent has a physical child pointer but the physical child does not have a twin pointer (POINTER=NOTWIN was specified on the SEGM macro for the physical child segment) and another segment of the same type already exists under the parent.

Action: Correct the error.

IX Call is not completed.

Explanation: Violated the insert rule. Review the insert rule in SEGM statement of DBDGEN in the *DL/I DOS/VS Resource Definition and Utilities*.

Possible causes:

- Insert of logical child (insert rule of logical parent is physical) and the logical parent does not exist.
- Insert of logical child and logical parent (insert rule is logical or virtual) and the logical parent does not exist and, in the user I/O area, the key of the logical parent does not match the corresponding key in the concatenated key in the logical child.
- Insert of logical child (insert rule of logical parent is virtual, and logical parent exists) and, in the user I/O area, the key in the logical parent does not match the corresponding key in the concatenated key in the logical child.

Action: Correct the program.

KA Numeric truncation error.

Explanation: During automatic conversion of a numeric field from one format to another format, an intermediate or final field was not large enough to contain the significant digits in the ‘from’ field.

Action: Correct program.

KB Character truncation error.

Explanation: During automatic length conversion of a character field, the ‘to' field was not large enough to contain all the non-blank characters moved from the “from' field. The field is moved left justified.

Action: Correct program.

KC Invalid packed decimal or zoned decimal format.

Explanation: During automatic conversion, a ‘from' field character was encountered that is not a valid packed decimal or zoned decimal character.

Action: Correct invalid character.

KD Type conflict for conversion.

Explanation: This code should occur only if the user has supplied a field exit routine and a field-to-field conversion was requested that was not supported by DL/I.

Action: Execute user's field exit routine.

KE NOREPL violation.

Explanation: The user attempted to modify a field that was not replace sensitive (REPLACE=NO was specified in the SENFLD statement for PSBGEN). The call is not completed.

Action: Correct program or specify REPLACE=YES.

LB **Call is not completed.**

Explanation: The segment that the user tried to load already exists in the data base.

Possible causes:

- A segment with an equal physical twin sequence field already exists for the parent.
- A physical parent has a physical child pointer but the physical child does not have a twin pointer (POINTER=NOTWIN was specified on the SEGM macro for the physical child segment) and another segment of the same type already exists under the parent.
- A segment with an equal logical twin sequence field already exists for the parent.

Action: None required.

LC **Call is not completed.**

Explanation: Key field of segments is out of sequence.

Action: Check and correct.

LD **Call is not completed.**

Explanation: No parent has been loaded for this segment.

Action: Check and correct.

LE **Call is not completed.**

Explanation: Sequence of sibling segments is not the same as the sequence in the DBD.

Action: Check and correct.

NA **Call is not completed.**

Explanation: The user tried to replace data in an index source segment that is used in search or subsequence fields of the index pointer segment, while the secondary index is used as the processing sequence.

Action: Correct the program.

NE **Error in some previous insert call or system error. The INSERT, DELETE, or REPLACE call was completed, as if the NE status code were a warning.**

Explanation: The user tried to delete or replace an index source segment or to insert an index source segment that had not been physically removed because of logical relationship requirements, and the corresponding index pointer segment could not be found.

Possible causes:

- During some previous insert call an index source segment was inserted with data in search and subsequence fields equal to an already existing index source segment. An NI status code had been returned with that call.
- Some error had occurred during reorganization of the data base.

Action:

- For DLET call, none. The index source segment which produced the duplicate key is now removed.
- For REPL call, user determined. For example, delete the segment and reinsert it with proper search and subsequence data.

NI **VSE/VSAM open error or duplicate key for index data base.**

Explanation: Check the error message printed on the system log device to get detailed information on the error.

Possible causes for being unable to open the index data base are:

- Error in job control statements.
- Control interval size, keylength, or relative key position specified in the VSE/VSAM DEFINE macro do not match the values specified for DBD generation.
- The processing option was L or LS but the data base was not empty or the data base was empty and the processing option was not L or LS.

A possible cause for having a duplicate key is that an index source segment was inserted with data in search and subsequence fields equal to an already existing index source segment. The index source segment is inserted, the index pointer segment is not inserted.

Action: Delete the segment and insert it with a unique key. For a subsequent delete call for the index source segment two different situations have to be distinguished:

1. If the index source segment just inserted and the one with the same search and subsequence fields point to different index target segments, the delete call returns an NE status code.
2. If the two index source segments point to the same index target segment, the first delete call for one of the index source segments removes the index pointer segment created by the first inserted index source segment. The second delete call for the remaining index source results in an NE status code.

NO **I/O error.**

Explanation: An I/O error occurring during processing of an index, either in the index or indexed data base. This status code is also returned when attempting an insertion of a duplicate secondary index pointer segment at Load time.

Action: Check and correct.

RX **Error in call.**

Explanation: Violated replace rule. Review replace rule in SEGM statement of DBDGEN in the *DL/I DOS/VS Resource Definition and Utilities*.

Action: Correct the program.

V1 **Program error.**

Explanation: Invalid length for variable length segment. The LL field of the variable length segment is either too large or too small. The length of the segment must be equal or less than the maximum length specified in the DBD. The length must be long enough to include the entire sequence field.

Action: Correct the program.

XD **I/O error.**

Explanation: An error occurred when the data base buffers were being written out to secondary storage during processing of a checkpoint (CHKP) call.

Action: Check and correct.

XH **Call is not completed.**

Explanation: Data base logging was inactive during checkpoint (CHKP) call processing.

Action: Ensure that data base logging is active during checkpoint call processing.

XR **Error During Checkpoint Processing for MPS Restart**

Explanation: For application programs using the MPS Restart facility, a DL/I checkpoint was not taken for one of the following reasons:

1. A VSE checkpoint was not taken before the DL/I CHKP call.
2. The VSE checkpoint failed.
3. A Temporary Storage error occurred during DL/I CHKP processing.

Action: If the status code was returned because of failure or error, then correct the situation described in the accompanying error message, otherwise, make sure that a VSE checkpoint is coded before each DL/I CHKP in the program.

ßß **Call completed.**

Explanation: Your call was completed.

Action: Proceed.

Abnormal Termination Messages

DL/I also issues execution time error messages. For an explanation of these messages and the required action, consult the *DL/I DOS/VS Messages and Codes*.

```
DLITPLI:   PROCEDURE (MAST_PCB_PTR) OPTIONS (MAIN);
           .
           .
DCL        01 MAST_PCB         UNALIGNED BASED (MAST_PCB_PTR),
           02 ---              ,
           02 ---              ,
           02 STAT_CODE        CHAR(2),
           02 ---              ,
           02 ---              ,
           02 ---              ;
           .
           .
           CALL PLITDLI (COUNTS,FUN_GN,MAST_PCB,SEG_IO,SSA1,SSA2);
           IF STAT_CODE='  ' THEN GOTO CONT;
           IF STAT_CODE='GE' THEN GOTO NOT_FOUND;
           IF STAT_CODE='GB' THEN GOTO END_OF_DB;
           GOTO ERROR;
CONT:      .
           .
NOT_FOUND: .
           .
END_OF_DB: .
           .
ERROR:     .
           .
           END DLITPLI;
```

Figure 2-2. PL/I Error Processing Routine Example

```
IDENTIFICATION DIVISION.
       .
       .
LINKAGE SECTION.
01     PCB-AREA.
       02 ---  .
       02 ---  .
       02 STATUS-CODE  PICTURE   X(02).
       02 ---  .
       02 ---  .
       02 ---  .
PROCEDURE DIVISION.
       ENTRY 'DLITCBL' USING PCB-AREA.
       .
       .
       CALL 'CBLTDLI' USING FUNC-GU,PCB-CU,PCB-AREA,SEG-IO,
                   SSA1,SSA2.
       IF STATUS-CODE = SPACES GO TO CONT.
       IF STATUS-CODE = 'GE' GO TO NOT FOUND.
       GO TO ERROR.
CONT.  .
       .
       .
NOT-FOUND.
       .
       .
ERROR. .
       .
       .
```

Figure 2-3. COBOL Error Processing Routine Example

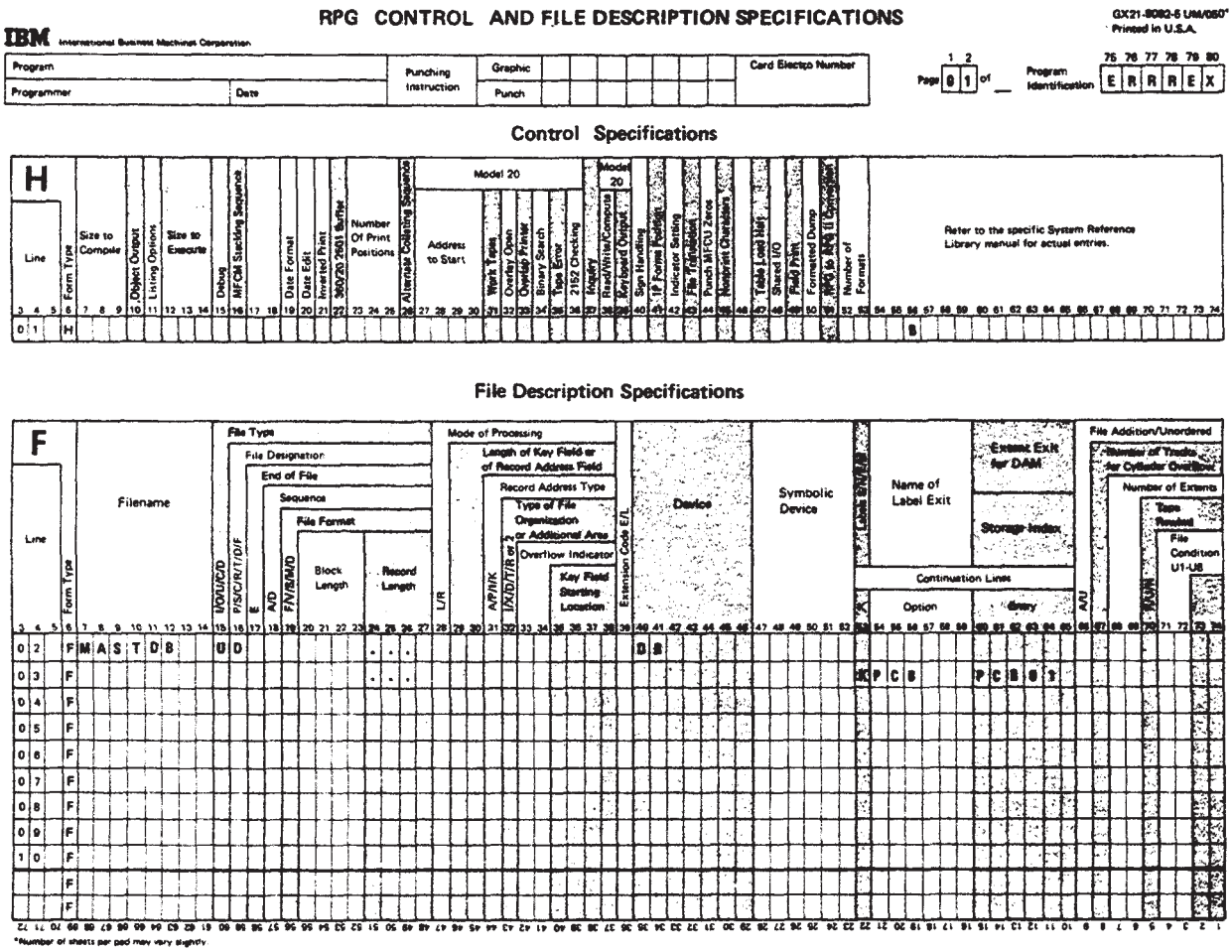

Figure 2-4 (Part 1 of 3). RPG II Error Processing Routine Example

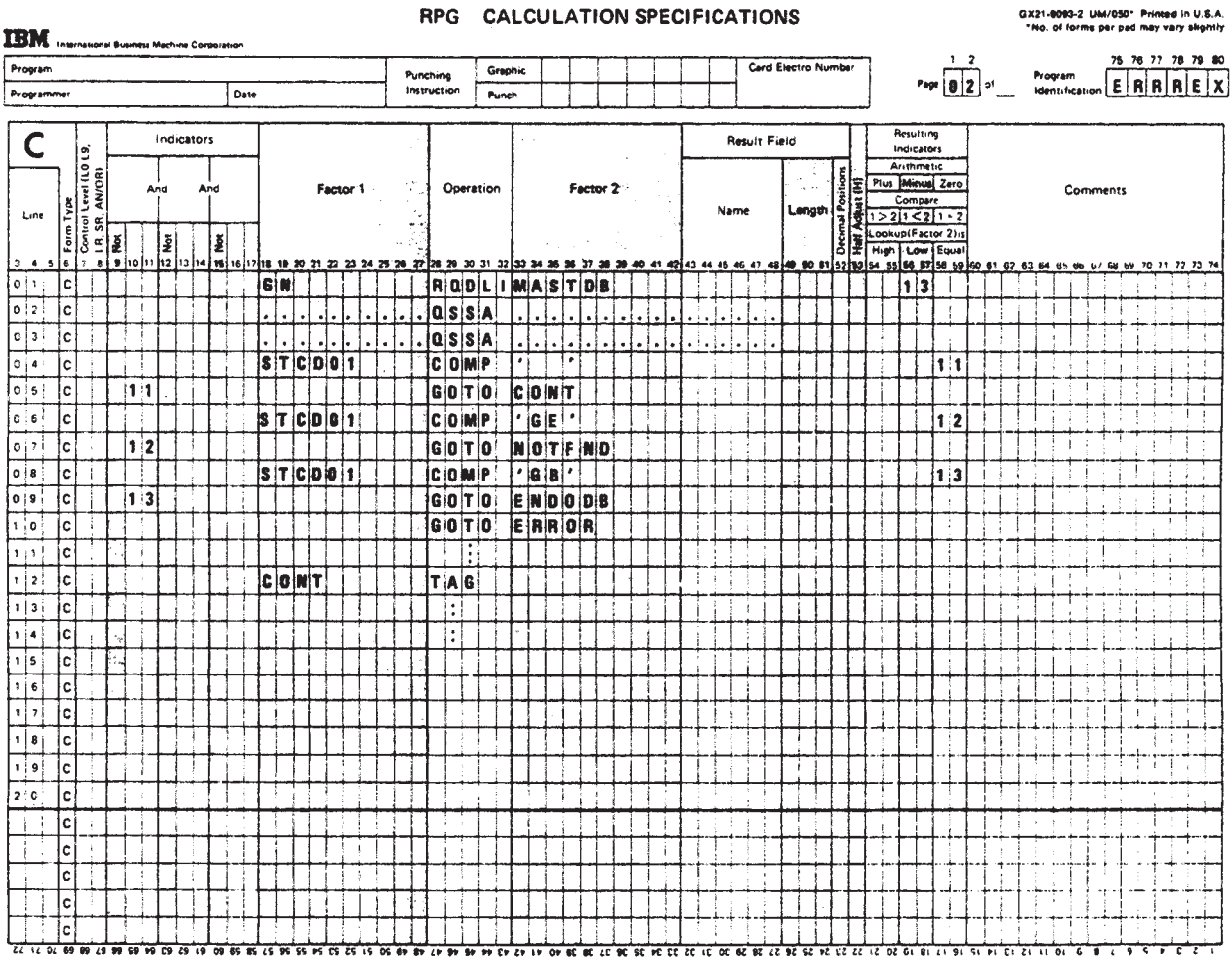

Figure 2-4 (Part 2 of 3). RPG II Error Processing Routine Example

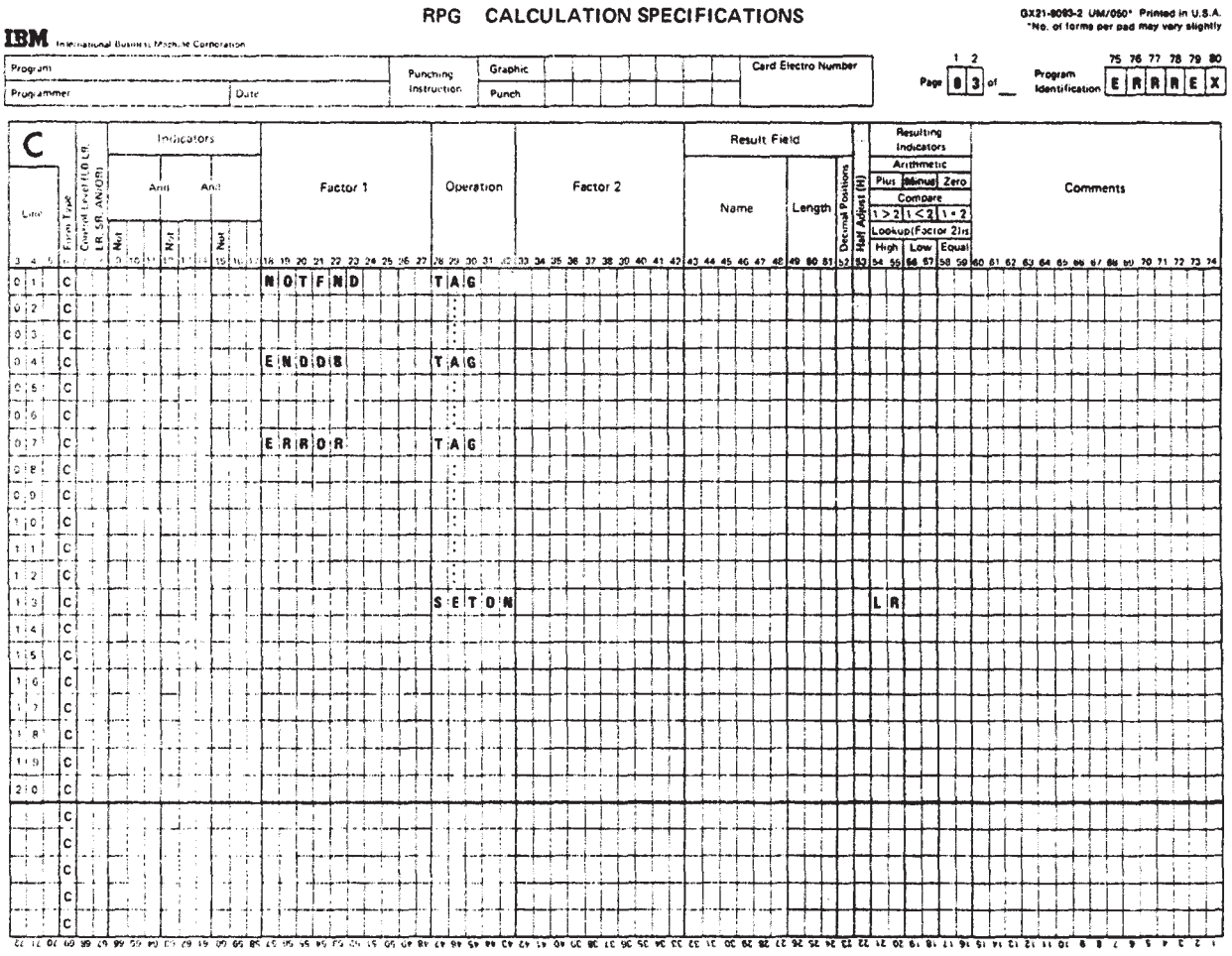

Figure 2-4 (Part 3 of 3). RPG II Error Processing Routine Example

Chapter 3. Online Programming Considerations

Before attempting to write a CICS/VS-DL/I program you should be familiar with DL/I DOS/VS batch programming concepts and Customer Information Control System/Virtual Storage (CICS/VS) programming fundamentals. References to the prerequisite publications are contained in the preface to this manual.

A programmer in a CICS/VS-DL/I environment accesses data bases in the same manner as in the batch environment. Since the CICS/VS-DL/I user may share access to DL/I data bases with other applications programs, the user has additional responsibilities when writing a CICS/VS-DL/I program. This chapter discusses these additional programming requirements and considerations. Where nothing to the contrary is stated, batch and online requirements are the same.

The CICS/VS application programmer requests DL/I services by issuing a DL/I call, or RQDLI command in RPG II, just as it is done in a batch environment. All call function codes described for batch use are valid in the online environment.

In a CICS/VS environment, a single copy of a program can service multiple transactions concurrently. Because of this multi-thread environment capability, all DL/I associated storage areas such as PCB pointers, SSAs, and segment I/O areas that are uniquely identified with a transaction must be located in storage that is also uniquely identified with the transaction. This storage must either be obtained from CICS/VS Dynamic Storage or be defined in the transaction work area.

The five steps to request DL/I services are:

1. Obtain addresses of PCBs for use by this transaction by issuing a DL/I call with 'PCB ' as the function code. This is described below, under "Obtaining the Address of the PCB: The Scheduling Call."
2. Acquire working storage for Assembler language programs; for COBOL, PL/I, and RPG II this is automatically handled. For Assembler there are three ways to reserve storage for the count field, function code field, I/O area, SSAs, and parameter list.
   a. The recommended approach is to reserve these in the transaction work area.
   b. You may issue one CICS/VS GETMAIN macro to hold all these areas.
   c. The least efficient method is to issue one CICS/VS GETMAIN macro for each area.

   As with all CICS/VS GETMAIN operations, storage account areas must be considered when techniques b and c are used.
3. Set the parameter count and function code as in the batch environment.
4. Furnish the PCB address provided by the 'PCB ' call previously issued.
5. Issue the call.

Obtaining the Address of the PCB: The Scheduling Call

Before accessing DL/I data bases, a CICS/VS program must issue a special DL/I call to initiate scheduling of a PSB. It is the responsibility of the programmer to determine the results of this call, details of which are given later under "Checking the Response to a DL/I Call in a CICS/VS Environment". The format of the scheduling call is:

For COBOL:

```
CALL 'CBLTDLI' USING [parm-count,] call-function
     [,psbname [,uibparm]].
```

For PL/I:

```
CALL PLITDLI (parm-count,call-function[,psbname [,uibparm]]);
```

For Assembler:

```
CALLDLI {ASMTDLI},([parm-count,]call-function[,psbname [,uibparm]])
        {CBLTDLI}
```

parm-count
: is the name of a binary fullword containing the parameter count. This count equals 1, 2, or 3 depending on whether or not psbname and uibparm are specified. The parm-count parameter is optional in Assembler and COBOL.

call-function
: is the name of the field containing the 4-character function 'PCB '.

psbname
: is the name of the 8-byte field containing the 1- to 7-character PSB generation name, right padded with blanks, that the application program accesses. If the uibparm is not used, this parameter is optional, and, if omitted, the default is the first PSB name associated with the application program name in the DL/I application control table generation.

  Note: In order to indicate that a default PSB is to be scheduled when the uibparm is specified in COBOL, PL/I, or Assembler, a psbname of '*b' must be used.

uibparm
: is the name of a fullword to which DL/I returns the address of the User Interface Block. Use of this parameter is optional, but desirable in COBOL, PL/I, and Assembler programs using the CICS/VS command language. RPG II requires use of the User Interface Block. The User Interface Block (UIB) is a control block used to pass to the user the address of the PCB list and the scheduling and termination response and error codes. If this parameter is omitted, the PCB list address and response and error codes will be returned in fields in the CICS/VS task communication area (TCA).

For RPG II:

```
PCB         RQDLI                  in
[PSBNAME    ELEM          psb-name]
SET         ELEM          BUIB
```

in
: indicator required in pos 56-57

psb-name
: is the name of the PSB to be scheduled. It can be specified either as a var-name (a variable psb-name containing the psbname as a character string, as shown in [---] above) or as an alphameric literal constant

```
PSBNAME     ELEM      'PSBNAM1'
```

: where PSBNAM1 is the name of the PSB.

BUIB
: refers to a field in DFHDUM that is the base of DLIUIB. User must specify such a field in DFHDUM to establish the addressability of PCBs after a scheduling call. (See Figure 3-7 on page 3-24.)

If no PSB name is explicitly specified, the Translator will generate a PSB name of '*b', which will cause the first PSB to be scheduled by default.

After a successful scheduling call, the field UIBPCBAL, or TCADLPCB if the UIB is not used, or UAPCBL in RPG II, contains the address of a PCB list. The PCB list consists of a series of 4-byte addresses that point to the PCBs within the PSB that has been scheduled. The last address in the list is indicated by the high order bit being 1.

Releasing a PSB in a CICS/VS Application Program: The Termination Call

The TERM call is the mechanism through which a CICS/VS transaction communicates to DL/I that all modifications made to the data bases by the transaction to this point are committed and cannot be backed out.

When scheduling intent is used, the TERM call also releases any scheduling intent caused by the PSB currently scheduled by the transaction.

Conversational programs should release the PSB before writing output onto a terminal so that other transactions can use the PSB while the conversational program is waiting for a response. Before issuing any other DL/I calls requesting DL/I access to a data base, however, the application program must again schedule the PSB using a scheduling call. If necessary, position in the data base must also be reestablished.

When program isolation is used, the TERM call also releases any DL/I resources currently enqueued by the transaction.

To release a PSB, the program issues a call of the following format:

For COBOL:

```
CALL 'CBLTDLI' USING [parm-count,] call-function.
```

For PL/I:

```
CALL PLITDLI (parm-count,call-function);
```

For Assembler:

```
CALLDLI {ASMTDLI},([parm-count,]call-function)
        {CBLTDLI}
```

Note: The contents of registers 1, 14, and 15 are altered.

parm-count
: is the name of a fullword containing a binary value of 1.

call-function
: is the name of a 4-byte field containing the value 'TERM' or 'T' and three blanks.

For RPG II:

```
TERM RQDLI
```

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

Checking the Response to a DL/I Call in a CICS/VS Environment

In a CICS/VS transaction, after every DL/I call, the transaction should check the status of the DL/I-CICS/VS interface. There are three types of DL/I calls that are processed by the DL/I-CICS/VS interface: scheduling, termination, and data base. The following paragraphs describe how to check the status of the interface after each type of call. The check of the DL/I-CICS/VS interface should occur before checking the DL/I status code returned in the PCB.

Include code immediately following the call to examine the field UIBFCTR, or TCAFCTR (TCAFCRC in ANS COBOL) if the UIB is not used, or UFCTR in RPG II, and, based on its contents, transfer control if necessary to an exception-handling routine. The possible response codes in these fields are:

Scheduling Call

Condition	**Response Code**
NORESP - Normal Response	X‘00’
INVREQ - Invalid Request	X‘08’
NOTOPEN - Not open	X‘0C’

NORESP
: indicates that the requested function was completed normally and that the field UIBPCBAL, or TCADLPCB if the UIB is not used, or UAPCBL in RPG II, contains the address of the PCB list.

INVREQ
: indicates that the field UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II, contains one of the error codes that are listed together with their explanations below:

  X‘01’ PSB name, as provided in the scheduling call, is not in the PSB directory.

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

  X‘05’ The PSB could not be initialized by DL/I online initialization.

  X‘06’ The PSB in the scheduling call is not defined in the program's application control table entry, or is too long, or is not delimited by a blank.

  X‘07’ A TERM call was issued when the task has already been terminated.

  X‘08’ A data base CALL was issued when the task was not scheduled.

  X‘09’ An MPS batch program attempted to issue a PCB call for a read-only PSB or for a non-exclusive PSB if program isolation is active.

  X‘FF’ The DL/I interface has been terminated or DL/I initialization failed.

NOTOPEN
: indicates that one or more DBD entries associated with this PSB are stopped or that a scheduling conflict with an MPS-scheduled task has occurred. Stopped means that the data base is not available for use because of an initialization error or an I/O error, or because it is closed. Conflict with an MPS-scheduled task means that a task running under MPS in a batch partition has:

1. Exclusive control of a segment type (PROCOPT=E) to which the PSB your task is trying to schedule is sensitive.
2. Sensitivity to a segment type to which the PSB your task is trying to schedule has specified exclusive control (PROCOPT=E).
3. Update sensitivity to a segment type (PROCOPT=I, D, R, or A) to which the PSB your task is trying to schedule also has update sensitivity and program isolation is not being used.

Field UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II, contains one of the error codes listed with their explanation below:

X‘01’ One or more DBD entries associated with the PSB are stopped.
X‘02’ A scheduling conflict with a currently active MPS batch partition occurred.

Note: A termination call is not required after an unsuccessful scheduling call because no PSB resources were acquired.

Termination Call

Condition	**Response Code**
NORESP - Normal Response	X‘00’
INVREQ - Invalid Request	X‘08’

NORESP
indicates that the DL/I resources have been released.

INVREQ
indicates that the field UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II, contains one of the error codes that are listed together with their explanations below:

X‘07’ TERM requested but task not scheduled.
X‘FF’ The DL/I interface has been terminated or DL/I initialization failed.

Note: For TERM calls, only the first (high-order) digit is considered.

Data Base Call

Condition	**Response Code**
NORESP - Normal Response	X‘00’
INVREQ - Invalid Request	X‘08’

NORESP
indicates that the DL/I interface completed the call.

INVREQ
indicates that the field UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II, contains one of the error codes that are listed together with their explanations below:

X‘08’ A DL/I call was made but the task has not scheduled a PSB.
X‘FF’ The DL/I interface has been terminated or DL/I initialization failed.

If a DL/I task abnormal termination occurs during online processing, control is not returned to the application program and the transaction is terminated with a CICS/VS message. In that message, the numeric part of the code that follows the word ABEND corresponds to the numeric portion of the applicable DL/I message number as listed in *DL/I DOS/VS Messages and Codes*. The code normally begins

with D but it begins with E if the termination cannot be noted on the transient data destination CSMT.

MPS (Multiple Partition Support) Considerations

An online program will receive a return code from a PCB call if it conflicts with an MPS batch job, instead of waiting. In this case, UIBFCTR, or TCAFCTR (TCAFCRC in ANS COBOL) if the UIB is not used, or UFCTR in RPG II, will contain X'0C' and UIBDLTR, or TCADLTR if the UIB is not used, or UDLTR in RPG II, will contain X'02'. When the data base is not open, the fields will contain X'0C' and X'01', respectively.

If any online tasks must wait for a resource owned by a batch MPS task, the MPS task will be informed on the next and all subsequent calls until a DL/I checkpoint is issued. (Note that making the resource available through some other action makes no difference. The passback indicates that a wait was required, not necessarily that a task is currently waiting.) This condition is flagged by setting the high-order bit of the first byte of the "JCB Address" field in the PCB to a one (X'80'). The application program must test for this condition by examining the field in the PCB mask that is reserved for DL/I. In COBOL, it is labeled "RESERVE-DL/I"; in PL/I, "RESERVE_DL/I"; in RPG II, "RESRij"; and in Assembler, "DBPCBRSV". See the section "Program Communication Block (PCB) Mask" in Chapter 1.

MPS batch jobs not using program isolation or MPS Restart are permitted to issue PCB and TERM calls, or PCB and TERM RQDLI commands in RPG II. This allows tasks conflicting with the batch job to run before the batch job completes. However, the following restrictions apply:

1. The first PCB call is issued automatically by DL/I so before an MPS batch job issues its first PCB call, it must issue a TERM call.
2. The PSB name used by an MPS batch job in a PCB call must always be the one specified in the DL/I parameter statement. The PCB addresses are the same as at the start of the application program. These addresses should be used after a PCB call.
3. The format of the PCB and TERM calls are the same as in online execution except the CALL macro is used instead of CALLDLI.
4. The user must not issue PCB calls and TERM calls for a read-only PSB.
5. There is no feedback information passed to the program. The MPS batch program request handler intercepts the return code, and if it is non-zero, it will ABEND the batch job.
6. After an MPS batch job has successfully issued its own PCB call, it is considered to be an online task from a scheduling viewpoint.

Notes:

1. If PCB and TERM calls are used by MPS batch jobs, jobs must not be run in the normal DL/I batch environment or in any environment using Program Isolation or MPS Restart. Also, the use of these calls is not upward compatible with IMS/VS; that is, these calls are not permitted in batch IMS/VS.
2. DL/I application programs can access a data base that is resident on another CICS/VS system via the CICS/VS Intersystem Communication Support. If an MPS batch application is to access a remote data base using Intersystem Communication or MPS Restart, the program must not issue PCB or TERM

calls. PCB calls would receive an abnormal return code of X'08' in TCAFCTR (TCAFCRC in ANS COBOL), or UFCTR in RPG II; and X'09' in TCADLTR, or UDLTR in RPG II. If issued in the MPS Restart environment, an error message will be issued and the batch partition will be cancelled.

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

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

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

Issuing the DL/I Call in a CICS/VS Environment

DL/I data base services are available to CICS/VS application programs through call statements, or RQDLI commands in RPG II. The call statement formats for ANS COBOL and PL/I are similar. For Assembler language application programs, a CALLDLI macro instruction is used. The general formats of the DL/I calls are as follows:

For COBOL:

```
CALL 'CBLTDLI' USING [parm-count,] call-function, db-pcb-name,
                       i/o-area[, ssa...].
```

For PL/I:

```
CALL PLITDLI (parm-count,call-function,db-pcb-name,i/o-area[,ssa...]);
```

For RPG II:

The format of RQDLI commands in RPG II is the same as in the batch environment (see Chapter 1, "RQDLI commands for DB Access").

For Assembler:

```
CALLDLI {ASMTDLI}[,([parm-count,]call-function,db-pcb-name,
        {CBLTDLI}
                       i/o-area[,ssa...]) ]
```

parm-count
: is the name of a binary fullword containing the parameter count. For COBOL and Assembler it is optional.

call-function
: is the name of the field containing the 4-character DL/I call function desired.

db-pcb-name
: is the name of the PCB (or DSECT if Assembler).

i/o-area
: is the name of the I/O area.

ssa...
: are the names of the SSAs; these parameters are optional.

Notes:

1. If no parameters are specified in an Assembler language CALLDLI macro instruction, register 1 is assumed to contain the address of a parameter list.
2. In Assembler language, the following format may be used as an alternative:

   ```
   CALLDLI  {ASMTDLI}, MF=(E, {(register)} )
            {CBLTDLI}         { address  }
   ```

   Register contains the address of the parameter list. Address is the address of the parameter list.

   Register 13 must contain the address of a 72-byte user-provided save area. The CALLDLI macro alters the contents of registers 1, 14, and 15.
3. If the application program makes a DL/I data base call without previously making a successful scheduling call, a 1-byte response code (X'08') is placed in the field UIBFCTR, or TCAFCTR (TCAFCRC in ANS COBOL) if the UIB is

not used, or UFCTR in RPG II, indicating an invalid request. If the call is accepted, the field is set to binary zeros. However, the user must still check the DL/I PCB status code.

Online Application Coding Examples

The following examples assume the application programmer has a thorough understanding of CICS/VS coding requirements and techniques. The examples, therefore, only illustrate the use of the DL/I portions of the application programs.

DL/I Requests in an ANS COBOL Program

The PCB addresses must be obtained upon program entry by issuing a scheduling call. After CICS/VS returns control to the application program, the programmer moves the contents of UIBPCBAL, or TCADLPCB if the UIB is not used, to the BLL pointer which is the base for the layout of the PCB pointers in the Linkage Section. The programmer then moves the addresses of the PCBs to their BLL pointers to provide the base addresses for the PCBs. When this has been done, the program is in the same state as a DL/I DOS/VS batch application program in which the following statement has been executed.

```
ENTRY 'DLITCBL' USING PCB1,PCB2.
```

Figure 3-1 gives examples of writing DL/I requests in an ANS COBOL program when the UIB is being used. Figure 3-2 on page 3-12 gives examples when the UIB is not used. Only a few of the possible combinations of operands are shown.

```
WORKING-STORAGE SECTION.
01  SEGMENT-AREA.
    02 KEY PICTURE X(8).
    02 NAME PICTURE X(20).
    .
    .
01 SSA.
    02 SEGMENT-NAME PICTURE X(8).
    02 LEFT-PAR PICTURE X(1).
    02 FIELD-NAME PICTURE X(8).
    02 RO PICTURE X(2).
    02 COMP-VALUE PICTURE X(n).
    02 RIGHT-PAR PICTURE X(1).
77 FUNCTION PICTURE X(4).
77 PSBNAME PICTURE X(8) VALUE 'CBOLPSB '.
    .
    .
LINKAGE SECTION.
01  BLLCELLS.
    02 FILLER  PICTURE S9(8) COMP.
    02 UIB-PTR  PICTURE S9(8) COMP.
    02 B-PCB-PTRS  PICTURE S9(8) COMP.
    02 B-PCB1  PICTURE S9(8) COMP.
    02 B-PCB2  PICTURE S9(8) COMP.
01 UIB COPY UIB.
```

Figure 3-1 (Part 1 of 2). Online COBOL Application Program Examples (UIB used) (CICS/VS Command Language Environment).

```
* THE FOLLOWING IS THE EXPANSION OF UIB
01  UIB.
    02 UIBPCBAL  PICTURE S9(8) COMP.
    02 UIBFCTR  PICTURE X(1).
    02 UIBDLTR  PICTURE X(1).
01 PCB-PTRS.
    02 PCB1-PTR  PICTURE S9(8) COMP.
    02 PCB2-PTR  PICTURE S9(8) COMP.
01  PCB1.
    02 DBD1-NAME PICTURE X(8).
    .
    .
01  PCB2.
    02 DBD2-NAME PICTURE X(8).
    .
    .
PROCEDURE DIVISION.
    .
    .
* DO DL/I SCHEDULING
    MOVE 'PCB ' TO FUNCTION.
    CALL 'CBLTDLI' USING FUNCTION, PSBNAME, UIB-PTR.
    IF UIBFCTR NOT EQUAL TO LOW-VALUES GO TO SCHEDULING-ERROR.
    MOVE UIBPCBAL TO B-PCB-PTRS.
    MOVE PCB1-PTR TO B-PCB1.
    MOVE PCB2-PTR TO B-PCB2.
    .
    .
* SET UP FOR GU CALL
    MOVE 'ROOT' TO SEGMENT-NAME.
    MOVE '(' TO LEFT-PAR.
    MOVE 'KEY' TO FIELD-NAME.
    MOVE '= ' TO RO.
    MOVE KEYI TO COMP-VALUE.
    MOVE ')' TO RIGHT-PAR.
    MOVE 'GU' TO FUNCTION.
    CALL 'CBLTDLI' USING FUNCTION, PCB1, SEGMENT-AREA, SSA.
    IF UIBFCTR NOT EQUAL TO LOW-VALUES GO TO INTERFACE-ERROR.
    IF STATUS-CODE IN PCB1 = 'GE' GO TO NOT-FOUND.
    IF STATUS-CODE IN PCB1 NOT EQUAL TO ' ' GO TO CALL-ERROR.
    .
    .
```

Figure 3-1 (Part 2 of 2). Online COBOL Application Program Examples (UIB used) (CICS/VS Command Language Environment).

```
WORKING STORAGE SECTION.
77  PCB-FUNCTION PICTURE X(4) VALUE 'PCB '.
77  PSBNAME PICTURE X(8) VALUE 'CBOLPSB '.
77  GU-FUNCTION PICTURE X(4) VALUE 'GU '.
LINKAGE SECTION.
01  DFHBLLDS COPY DFHBLLDS.
    02  B-PCB-PTRS PICTURE S9(8) COMPUTATIONAL.
    02  B-PCB1 PICTURE S9(8) COMPUTATIONAL.
    02  B-PCB2 PICTURE S9(8) COMPUTATIONAL.
01  DFHCSADS COPY DFHCSADS.
01  DFHTCADS COPY DFHTCADS.
* TWA DEFINITIONS
    02 SEGMENT-AREA.
       03 KEY PICTURE X(8).
       03 NAME PICTURE X(20).
       .
       .
    02 SSA.
       03 SEGMENT-NAME PICTURE X(8).
       03 LEFT-PAR PICTURE X(1).
       03 FIELD-NAME PICTURE X(8).
       03 RO PICTURE X(2).
       03 COMP-VALUE PICTURE X(n).
       03 RIGHT-PAR PICTURE X(1).
01  PCB-PTRS.
    02 PCB1-PTR PICTURE S9(8) COMPUTATIONAL.
    02 PCB2-PTR PICTURE S9(8) COMPUTATIONAL.
01  PCB1.
    02 DBD1-NAME PICTURE X(8).
    .
    .
01  PCB2.
    02 DBD2-NAME PICTURE X(8).
    .
    .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.
    .
    .
* DO DL/I SCHEDULING
    CALL 'CBLTDLI' USING PCB-FUNCTION, PSBNAME.
    IF TCAFCRC NOT EQUAL TO LOW-VALUES GO TO SCHEDULING-ERROR.

    MOVE TCADLPCB TO B-PCB-PTRS.
    MOVE PCB1-PTR TO B-PCB1.
    MOVE PCB2-PTR TO B-PCB2.
    .
    .
```

Figure 3-2 (Part 1 of 2). Online COBOL Application Program Examples (UIB not used) (Macro Language Environment).

```
* SET UP FOR GU CALL
    MOVE ‘ROOT’ TO SEGMENT-NAME.
    MOVE ‘(’ TO LEFT-PAR.
    MOVE ‘KEY’ TO FIELD-NAME.
    MOVE ‘= ’ TO RO.
    MOVE KEYI TO COMP-VALUE.
    MOVE ‘)’ TO RIGHT-PAR.
    CALL ‘CBLTDLI’ USING GU-FUNCTION, PCB1, SEGMENT-AREA, SSA.
    IF TCAFCRC NOT EQUAL TO LOW-VALUES GO TO INTERFACE-ERROR.
    IF STATUS-CODE IN PCB1 = ‘GE’ GO TO NOT-FOUND.
    IF STATUS-CODE IN PCB1 NOT EQUAL TO ‘ ’ GO TO CALL-ERROR.
    .
    .
```

Figure 3-2 (Part 2 of 2). Online COBOL Application Program Examples (UIB not used) (Macro Language Environment).

DL/I Requests in a PL/I Program

The PCB addresses must be obtained upon program entry by issuing a scheduling call. When CICS/VS returns control to the application program, the base address of a structure of PCB pointers is in UIBPCBAL, or TCADLPCB if the UIB is not used. The PL/I programmer must move the address to the based variable for his declared structure of PCB pointers. He then loads the pointers to all PCBs from this structure. When this has been done, the program is in the same state as a DL/I DOS/VS batch application program in which the following statement has been executed.

```
DLITPLI: PROCEDURE (pcbname1,...) OPTIONS (MAIN);
```

The PL/I programmer may then make DL/I requests. Examples of these are shown in Figure 3-3 when the UIB is used and in Figure 3-4 on page 3-16 when the UIB is not used.

```
TEST:  PROC OPTIONS (MAIN, RE-ENTRANT);
% INCLUDE DLIUIB;
/* PRODUCES THE FOLLOWING:                    */
/*                                            */
/* DCL UIBPTR POINTER;                        */
/* DCL 1 UIB BASED (UIBPTR),                  */
/*        2 UIBPCBAL PTR,                     */
/*        2 UIBFCTR CHAR(1),                  */
/*        2 UIBDLTR CHAR(1);                  */
/*                                            */
DCL 1 PCB_PTRS BASED (UIBPCBAL),
        2 PCB1_PTR POINTER,
        2 PCB2_PTR POINTER;
DCL B_PCB1 POINTER;
DCL B_PCB2 POINTER;
DCL 1 PCB1 BASED (B_PCB1),
        2 DBD_NAME CHAR (8),
        .
        .
        .
DCL 1 PCB2 BASED (B_PCB2),
        2 DBD_NAME CHAR (8),
        .
        .
        .
DCL 1 SEGMENT_AREA,
        2 KEY CHAR (8),
        2 NAME CHAR (20),
        .
        .
        .
```

Figure 3-3 (Part 1 of 2). Online PL/I Application Program Examples (UIB used) (CICS/VS Command Language Environment).

```
DCL 1 SSA,
        2 SEGMENT_NAME CHAR (8),
        2 LEFT_PAR CHAR (1),
        2 FIELD_NAME CHAR (8),
        2 RO CHAR (2),
        2 COMP_VALUE CHAR (n),
        2 RIGHT_PAR CHAR (1);
DCL PCB_FUNCTION CHAR (4) STATIC INIT ('PCB ');
DCL GU_FUNCTION CHAR (4) STATIC INIT ('GU ');
DCL PSBNAME CHAR (8) STATIC INIT ('PLIPSB ');
DCL COUNT FIXED BIN (31);
        .
        .
        .
/*DO DL/I SCHEDULING                          */
        COUNT=3;
        CALL PLITDLI (COUNT, PCB_FUNCTION, PSBNAME, UIBPTR);
        IF UIBFCTR ¬= _'00000000'B THEN GO TO SCHEDULING_ERROR;
        B_PCB1 = PCB1_PTR;
        B_PCB2 = PCB2_PTR;
        .
        .
        .
/* SET UP FOR GU CALL                           */
        SEGMENT_NAME = 'ROOT';
        LEFT_PAR = '(';
        FIELD_NAME = 'KEY';
        RO = '= ';
        COMP_VALUE = KEYI;
        RIGHT_PAR = ')';
        COUNT=4;
        CALL PLITDLI (COUNT, GU_FUNCTION, PCB1, SEGMENT_AREA, SSA);
        IF UIBFCTR ¬= _'00000000'B THEN GO TO INTERFACE_ERROR;
        IF PCB1.STATUS_CODE = 'GE' THEN GO TO NOT_FOUND;
        IF PCB1.STATUS_CODE ¬= ' ' THEN GO TO STATUS_ERROR;
        .
        .
        .
```

Figure 3-3 (Part 2 of 2). Online PL/I Application Program Examples (UIB used) (CICS/VS Command Language Environment).

```
TEST: PROC OPTIONS (MAIN, RE-ENTRANT);
%INCLUDE DFHCSADS;                      /*CSA DEFINITION */
%INCLUDE DFHTCADS;                      /*TCA DEFINITION */
          2 DUMMY CHAR(1);  /*DUMMY VARIABLE TO  ADD SEMICOLON TO LAST
                            DECLARTION IN DFHTCADS, AS THIS IS NOT
                            SUPPLIED, TO ALLOW FOR TWA DEFINITION. */
    DCL B_PCB_PTRS POINTER;
    DCL 1  PCB_PTRS BASED (B_PCB_PTRS),
           2 PCB1_PTR POINTER,
           2 PCB2_PTR POINTER;
    DCL B_PCB1 POINTER;
    DCL B_PCB2 POINTER;
    DCL 1  PCB1 BASED (B_PCB1),
           2 DBD_NAME CHAR (8),
           .
           .
           .
    DCL 1  PCB2 BASED (B_PCB2),
           2 DBD_NAME CHAR (8),
           .
           .
           .
    DCL 1  SEGMENT_AREA,
           2 KEY CHAR (8),
           2 NAME CHAR (20),
           .
           .
           .
    DCL 1  SSA,
           2 SEGMENT_NAME CHAR (8),
           2 LEFT_PAR CHAR (1),
           2 FIELD_NAME CHAR (8),
           2 RO CHAR (2),
           2 COMP_VALUE CHAR (n),
           2 RIGHT_PAR CHAR (1);
    DCL PCB_FUNCTION CHAR (4) STATIC INIT ('PCB ');
    DCL GU_FUNCTION CHAR (4) STATIC INIT ('GU ');
    DCL PSBNAME CHAR (8) STATIC INIT ('PLIPSB ');
        DCL COUNT FIXED BIN (31);
        .
        .
        .
```

Figure 3-4 (Part 1 of 2). Online PL/I Application Program Examples (UIB not used) (Macro Language Environment).

```
/* DO DL/I SCHEDULING                          */
        COUNT=2;
        CALL PLITDLI (COUNT, PCB_FUNCTION, PSBNAME);
        IF TCAFCTR ¬= '00000000'B THEN GO TO SCHEDULING_ERROR;
        B_PCB_PTRS= TCADLPCB;
        B_PCB1 = PCB1_PTR;
        B_PCB2 = PCB2_PTR;
        .
        .
        .
/* SET UP FOR GU CALL                          */
        SEGMENT_NAME = 'ROOT';
        LEFT_PAR = '(';
        FIELD_NAME = 'KEY';
        RO = '= ';
        COMP_VALUE = KEYI;
        RIGHT_PAR = ')';
        COUNT=4;
        CALL PLITDLI (COUNT, GU_FUNCTION, PCB1,SEGMENT_AREA, SSA);
        IF TCAFCTR ¬= '00000000'B THEN GO TO INTERFACE_ERROR;
        IF PCB1.STATUS_CODE = 'GE' THEN GO TO NOT_FOUND;
        IF PCB1.STATUS_CODE ¬= ' ' THEN GO TO STATUS_ERROR;
        .
        .
        .
```

Figure 3-4 (Part 2 of 2). Online PL/I Application Program Examples (UIB not used) (Macro Language Environment).

DL/I Requests in an Assembler Language Program

The application programmer must first obtain the PCB addresses. The examples in Figure 3-5 and Figure 3-6 on page 3-20 show the options available to the application programmer in a few of the acceptable combinations. Figure 3-5 gives examples when the UIB is used and Figure 3-6 on page 3-20 when the UIB is not used. Note that the application program must be quasi-reentrant.

```
DFHEISTG          DSECT
UIBPTR            DS            A
SEGAREA           DS            0CLn
KEY               DS            CL4
NAME              DS            CL20
                  .
                  .
                  .
SSA               DS            0CLn            DL/I SSA
SEGNAME           DS            CL8
LPAR              DS            CL1
FLDNAME           DS            CL8
RO                DS            CL2
COMPVAL           DS            CLn
RPAR              DS            CL1
                  .
                  .
                  .
                  DLIUIB
* PRODUCES THE FOLLOWING:
*
* DLIUIB           DSECT
* UIB              DS            0F          EXTENDED CALL USER INTFC BLK
* UIBPCBAL         DS            A           PCB ADDRESS LIST
* UIBRCODE         DS            0XL2        DL/I INTERFACE RETURN CODES
* UIBFCTR          DS            X           RETURN CODE
* UIBDLTR          DS            X           ADDITIONAL INFORMATION
*                  DS            0F          LEN IS FULL WORD MULTIPLE
* UIBLEN           EQU           *-UIB       LENGTH OF UIB
START             DFHEIENT
                  .
                  .
                  .
*   DO DL/I SCHEDULING
                  CALLDLI       ASMTDLI,(PCB,PSBNAME,UIBPTR)
                  USING         UIB,R7
                  L             R7,UIBPTR
                  CLI           UIBFCTR,X'00'
                  BNE           SCHERROR
                  L             R4,UIBPCBAL
                  USING         PCBADRS,R4
                  L             R5,PCB1ADR
                  USING         PCB1,R5
                  L             R6,PCB2ADR
                  USING         PCB2,R6
```

Figure 3-5 (Part 1 of 2). Online Assembler Language Application Program Examples (UIB used) (CICS/VS Command Language Environment)

```
                    .
                    .
                    .
                    .
   *  SET UP FOR GU CALL
                    MVC             SEGNAME,=CL8'ROOT'
                    MVI             LPAR,C'('
                    MVC             FLDNAME,=CL8'KEY'
                    MVC             RO,=C'= '
                    MVC             COMPVAL,KEYI
                    MVI             RPAR,C')'
                    CALLDLI         ASMTDLI (GU,PCB1,SEGAREA,SSA)
                    CLI             UIBFCTR,X'00'
                    BNE             INTERROR
                    CLC             PCB1STC,=C'GE'
                    BE              NOTFOUND
                    CLC             PCB1STC,=C' '
                    BNE             STCERROR
                    .
                    .
                    .
                    .
PCB                 DC              CL4'PCB'
GU                  DC              CL4'GU'
PSBNAME             DC              CL8'ASOLPSB '
PCBPTRS             DSECT
PCB1ADR             DS              A             ADR OF 1ST PCB IN PSB
PCB2ADR             DS              A             ADR OF 2ND PCB IN PSB
*
*  CONTINUE FOR AS MANY PCBS IN PSB
*
PCB1                DSECT
PCB1DBDN            DS              CL8             DBD NAME
PCB1LEV             DS              CL2             SEGMENT LEVEL
PCB1STC             DS              CL2             STATUS CODE
PCB1PRO             DS              CL4             PROCESSING OPTIONS
                    DS              F               RESERVED
PCB1SFD             DS              CL8             SEGMENT NAME
PCB1KFDL            DS              F               CURRENT LENGTH OF KEY
*                                                   FEEDBACK AREA
PCB1NSS             DS              F               NO OF SENSITIVE
*                                                   SEGMENTS IN PCB
PCB1KFD             DS              CL255           KEY FEEDBACK AREA
*
*  CONTINUE FOR AS MANY PCBS IN PSB
*
PCB2                DSECT
PCB2DBD             DS              CL8
                    .
                    .
                    .
```

Figure 3-5 (Part 2 of 2). Online Assembler Language Application Program Examples (UIB used) (CICS/VS Command Language Environment)

```
EXAMPLE  CSECT
R0       EQU      0
R1       EQU      1
R3       EQU      3
R4       EQU      4
R9       EQU      9
R12      EQU      12
R13      EQU      13
         BALR     R3,R0            LOAD BASE REGISTER (R3)
         USING    *,R3              ...AND TELL ASSEMBLER
         USING    DFHTCADS,R12     TELL ASSEMBLER ABOUT TCA
         USING    DFHCSADS,R13     ...AND CSA ADDRESSABILITY
*         .
*      ESTABLISH ADDRESSABILITY TO OTHER CICS/VS AREAS
*      AS REQUIRED BY THE APPLICATION PROGRAM
*         .
*        -----    SET UP AND ISSUE SCHEDULING (PCB) CALL
*                 (MF=E FORM OF CALLDLI MACRO DEMONSTRATED)
         LA       R1,COUNT            SET DL/I COUNT PARAMETER
         ST       R1,COUNTADR         ...ADR IN CALL PARM LIST
         LA       R1,PCB              GET ADR OF PCB FUNCTION CODE
         ST       R1,FUNADR           ...AND STORE IT IN PARM LIST
*        -----    OPTIONALLY SPECIFY NAME OF PSB TO BE SCHEDULED
         LA       R1,PSBNAME          GET ADR OF NAME OF PSB TO SCHED
         ST       R1,PCBADR           ...AND STORE IT IN PARM LIST
         MVC      COUNT,=F'2'       SET PARM COUNT = 2
*                                     IF PSB NAME WAS NOT SPECIFIED
*                                     ...COUNT SHOULD BE SET TO ONE
         LA       R1,DLIPARMS         POINT R1 AT PARM LIST
         ST       R13,CSASAVE         SAVE CSA ADR PRIOR TO MF=E
*                                     ...CALLDLI MACRO FORMAT USAGE
         LA       R13,CALLSAVE        PUT ADR OF SAVE AREA IN R13
*                                     ...PRIOR TO USING MF=E CALLDLI
*                                     ...MACRO FORMAT (CSA ADR LOST)
         CALLDLI  ASMTDLI,MF=(E,(1))  ISSUE PCB CALL (MF=E FORMAT)
         L        R13,CSASAVE         RECOVER CSA ADR AFTER MF=E
*                                     ...CALLDLI MACRO FORMAT USAGE
*        -----    CHECK SUCCESS OF SCHEDULING CALL
         CLI      TCAFCTR,X'00'     CALL SUCCESSFUL?
         BNE      SCHERROR            ...NO, GO DETERMINE PROBLEM
*        -----    SCHEDULE CALL OK, ESTABLISH ADDRESSABILITY TO PCBS
         L        R9,TCADLPCB         GET ADR OF PCB ADDRESSES
         USING    PCBADRS,R9          ...AND TELL ASSEMBLER
         L        R4,PCB1ADR          GET ADR OF 1ST PCB IN PSB
         USING    PCB1,R4             ...AND TELL ASSEMBLER
*        .
```

Figure 3-6 (Part 1 of 4). Online Assembler Language Application Program Examples (UIB not used) (Macro Language Environment)

```
*     ESTABLISH ADDRESSABILITY TO THE REMAINING PCBS IN THE PSB
*     AND CONTINUE WITH APPLICATION PROGRAM LOGIC
*        .
*       -----     INITIALIZE SSAS
        MVC       COMCODE1,=C'*--'    SET NULL COMMAND CODE IN 1ST SSA
        MVI       RP1,C')'            ...AND ENDING RIGHT PAREN
*        .
*     CONTINUE TO INITIALIZE THE REMAINING SSAS
*        .
*        -----    SET UP TO RETRIEVE A SEGMENT
         MVC      SEGNAME1,=CL8'ROOT' PUT SEG NAME IN SSA
         MVI      LP1,C'('            MAKE NAME QUALIFIED
         MVC      KEYNAME1,=CL8'SEQ'  ...PUT KEY FIELD NAME
         MVC      RO1,=C'= '          ...RELATIONAL OPERATOR
         MVC      KEY1,=CL5'00001'    ...AND KEY FIELD VALUE IN SSA
         MVC      COUNT,=F'4'         INDICATE 4 PARMS USED IN CALL
         CALLDLI  ASMTDLI,(COUNT,GU,PCB1,SEGIO,SSA1)
*        -----    CHECK FOR CALL ACCEPTANCE
         CLI      TCAFCTR,X'00'     WAS CALL ACCEPTED?
         BNE      CALLERR             ...NO, GO DETERMINE REASON
*         .
*        CALL WAS ACCEPTED.  CHECK DL/I PCB STATUS CODE
*        AND CONTINUE APPLICATION PROGRAM LOGIC
*         .
         DFHPC    TYPE=RETURN
*        -----    DL/I CALL ERROR ROUTINES
CALLERR  DS       0H
SCHERROR DS       0H
*         .
*      AT THIS POINT THE PROGRAM CAN DETERMINE THE REASON FOR
*      THE ERROR BY EXAMINING THE FIELD 'TCADLTR'.  IN MOST
*      CASES A CICS/VS ABEND SHOULD BE ISSUED
*         .
         DFHPC    TYPE=RETURN
*        -----    DL/I ONLINE FUNCTION CODE CONSTANTS
*                 (COULD BE A COPY BOOK)
PCB      DC       CL4'PCB'
GU       DC       CL4'GU'
GHU      DC       CL4'GHU'
GN       DC       CL4'GN'
GHN      DC       CL4'GHN'
GNP      DC       CL4'GNP'
GHNP     DC       CL4'GHNP'
REPL     DC       CL4'REPL'
ISRT     DC       CL4'ISRT'
DLET     DC       CL4'DLET'
TERM     DC       CL4'TERM'
*        -----    MISCELLANEOUS PROGRAM CONSTANTS
PSBNAME  DC       CL8'PIPSBA1 ' NAME OF PSB TO BE SCHEDULED
TWALEN   DC       A(TWASTOP-TWASTART) LENGTH OF TWA REQUIRED
*         .
```

Figure 3-6 (Part 2 of 4). Online Assembler Language Application Program Examples (UIB not used) (Macro Language Environment)

```
*      OTHER PROGRAM CONSTANTS
*         .
*        -----    CICS/VS DSECTS
         COPY DFHCSADS
         COPY DFHTCADS
*        -----    TWA STARTS HERE
TWASTART EQU      *
CSASAVE  DS       F                 CSA ADR SAVE AREA FOR MF=E CALLS
CALLSAVE DS       18F               REG SAVE AREA FOR MF=E CALLS
DLIPARMS DS       0F                DL/I CALL PARM LIST
*                                   FOR USER CREATED CALL PARM LISTS
COUNTADR DS       A                 ADR OF PARM COUNT VALUE
FUNADR   DS       A                 ADR OF FUNCTION CODE
PCBADR   DS       A                 ADR OF PCB USED WITH CALL
IOADR    DS       A                 ADR OF SEGMENT I/O AREA USED
SSA1ADR  DS       A                 ADR OF 1ST SSA USED IN CALL
SSA2ADR  DS       A                 ADR OF 2ND SSA USED IN CALL
*         .
*         .
*        -----    SSAS (COULD BE COPY BOOKS)
SSA1     DS       0CL29
SEGNAME1 DS       CL8               SEGMENT NAME
COMCODE1 DS       CL3               COMMAND CODE AREA OF SSA
LP1      DS       CL1               LEFT PAREN'('
KEYNAME1 DS       CL8               SEGMENT KEY FIELD NAME
RO1      DS       CL2               RELATIONAL OPERATOR
KEY1     DS       CL6               KEY FIELD VALUE
RP1      DS       CL1               SSA ENDING RIGHT PAREN
*        -----    MISCELLANEOUS WORKING STORAGE AREAS
COUNT    DS       F                 NUMBER OF PARMS IN LIST
SEGIO    DS       0CL40             A SEGMENT I/O AREA (COPY BOOK?)
ROOTKEY  DS       CL6               ROOT KEY FIELD IN ROOT SEGMENT
*         .
*     DEFINITIONS OF OTHER FIELDS IN SEGMENT
*         .
*         .
*     DEFINITIONS OF OTHER WORKING STORAGE AREAS
*     REQUIRED BY PROGRAM
*         .
TWASTOP  EQU      *                 END OF TWA
*        -----    DSECT USED TO ESTABLISH ADDRESSABILITY TO PCBS
*                 (COULD BE A COPY BOOK FOR EACH PSB IN INSTALLATION)
PCBADRS  DSECT
PCB1ADR  DS       A                 ADR OF 1ST PCB IN PSB
PCB2ADR  DS       A                 ADR OF 2ND PCB IN PSB
*         .
```

Figure 3-6 (Part 3 of 4). Online Assembler Language Application Program Examples (UIB not used) (Macro Language Environment)

```
*     CONTINUE FOR AS MANY PCBS IN PSB
*         .
PCB1     DSECT    (COULD BE CONTINUATION OF PSB COPY BOOK)
PCB1DBDN DS       CL8               DBD NAME
PCB1LEV  DS       CL2               LEVEL FEEDBACK
PCB1STC  DS       CL2               STATUS CODE
PCB1PRO  DS       CL4               PROCESSING OPTIONS
         DS       F                 RESERVED
PCB1SFD  DS       CL8               SEGMENT NAME FEEDBACK
PCB1KFDL DS       F                 CURRENT LENGTH OF KEY FEEDBACK
PCB1NSS  DS       F                 NUMBER OF SENSITIVE SEGMENTS
PCB1KFD  DS       CL255             KEY FEEDBACK AREA
*         .
*     CONTINUE FOR AS MANY PCBS IN PSB
*         .
*         .
*     COPY OTHER CICS/VS DSECTS AS REQUIRED BY PROGRAM
*         .
          END       EXAMPLE
```

Figure 3-6 (Part 4 of 4). Online Assembler Language Application Program Examples (UIB not used) (Macro Language Environment)

RQDLI Commands in an RPG II Program

The following lists all the peculiarities for RPG II applications using DL/I under CICS/VS.

1. File Description Specifications for DB-files may be specified and have the same format as in a batch environment. Their implication on the RQDLI command for standard data transfer is the same.

   The RQDLI commands have the same format as those specified for the batch environment in Chapter 1.

   Exception: For a scheduling call (func-name=PCB), additionally a SET option and a PSB-name option are supported.

2. `*ENTRY PLIST for DL/I under CICS/VS`

   In addition to the PARMs required by CICS/VS, additional parameters for DLIUIB, the PSB, and PCBs have to be specified by the user. The bases for those parameters must also be specified in DFHDUM, in the same order as in the `*ENTRY PLIST`.

   An example of an online RPG II application program is given in Figure 3-7.

```
I*THE LAYOUT IN DFHDUM MUST EXACTLY CORRESPOND TO THE
I*LAYOUT OF THE *ENTRY PLIST STARTING WITH DFHDUM
IDFHDUM      DS
I                                        1   4 SELPTR
I                                        5   8 BUIB
I                                        9  12 BPSB
I                                       13  16 BPCB01
I                                       17  20 BPCB02
I
IPCB01       DS
I .............
IPCB02       DS
I .............
I* USER MUST SPECIFY THE PROPER LAYOUT OF THE PCBS
I* PSB DEFINES THE ADDRESSLIST OF THE PCBADDRESSES
IPSB         DS
I                                        1   4 PCB01P
I                                        5   8 PCB02P
I/INSERT  .DLIUIB
I* THE FOLLOWING DATA STRUCTURE FOR THE UIB CONTROL BLOCK WILL
I* BE INSERTED FROM THE LIBRARY BY THE TRANSLATOR
IDLIUIB      DS
I
 1 4 UPCBAL
I                                         5   5 UFCTR
I                                         6   6 UDLTR
I                                        ...............
I*  END OF THE INSERTED UIB CONTROL BLOCK
C*  USER HAS TO SPECIFY AT LEAST THE PARAMETERS AFTER DFHDUM
```

Figure 3-7 (Part 1 of 3). Online RPG II Application Program Examples

```
C           *ENTRY     PLIST
C                      PARM           DFHEIB
C                      PARM           DFHCOM
C                      PARM           DFHDUM
C                      PARM           DLIUIB
C                      PARM           PSB
C                      PARM           PCB01
C                      PARM           PCB02
C*  BEFORE ACCESSING THE DATA BASE THE FOLLOWING
C*  STATEMENTS MUST BE CODED BY THE USER
C           PCB        RQDLI                    13
C           SET        ELEM           BUIB
C           PSBNAME    ELEM  'PSBNAM1'
C*      PSBNAME OPTION MUST BE SPECIFIED IN
C*      AN ELEM STATEMENT
C* THIS ESTABLISHES THE ADDRESSABILITY OF THE DATA STRUCTURE
C* DLIUIB AND ITS FIELDS
C* WITH THE HELP OF A MOVE STATEMENT THE ADDRESS OF THE PCBADDRESS
C* LIST IS PUT INTO THE BASE OF THE DS DEFINING THE ADDRESSLIST
C                      MOVE UPCBAL  BPSB
C                      CALL 'ILNSAD'
C* CHECK CICS/VS INTERFACE RESPONSE
C                     TESTB'01234567'UFCTR       10
C   10                GOTO NORESP
C                     TESTB'4'       UFCTR            10
C                     TESTB'5'       UFCTR            11
C   10N11             GOTO INVREQ
C   10 11             GOTO NOTOPEN
C* DATA STRUCTURE CONTAINING THE ADDRESS LIST OF THE PCBS IS
C* NOW ADDRESSABLE.IN THIS CASE PSBNAM1 CONTAINS ONLY
C* PCB01
C* ESTABLISH THE ADDRESSABILITY FOR PCB01 BY MOVE STATEMENT
C* FOLLOWED BY CALL TO ILNSAD
C                      MOVE PCB01P     BPCB01
C                      CALL 'ILNSAD'
C* NOW THE USER CAN ACCESS THE DATA BASE PCB01
C           GU         RQDLI                    13
C           PCB        ELEM           PCB01
C           INTO       ELEM           IOAREA200
C*      CONTINUE WITH PROGRAM
C           TERM       RQDLI
C* NOW PCB01 CAN NO LONGER BE ADDRESSED
C...........
C* BEFORE ACCESSING A NEW DATA BASE THE FOLLOWING
C* STATEMENTS MUST BE CODED BY THE USER
C           PCB        RQDLI                    13
C           SET        ELEM           BUIB
C           PSBNAME    ELEM   'PSBNAM2'
C*      PSBNAME OPTION MUST BE SPECIFIED IN
C*      AN ELEM STATEMENT
```

Figure 3-7 (Part 2 of 3). Online RPG II Application Program Examples

```
C* THIS ESTABLISHES THE ADDRESSABILITY OF THE DATA STRUCTURE
C* DLIUIB AND ITS FIELDS
C                     MOVE UPCBAL     BPSB
C                     CALL 'ILNSAD'
C* DATA STRUCTURE CONTAINING THE ADDRESS LIST OF THE PCBS IS
C* NOW ADDRESSABLE.IN THIS CASE PSBNAM2 CONTAINS ONLY
C* PCB02
C* ESTABLISH THE ADDRESSABILITY FOR PCB02 BY MOVE STATEMENT
C* FOLLOWED BY CALL TO ILNSAD
C                     MOVE PCB01P     BPCB02
C                     CALL 'ILNSAD'
C* NOW THE USER CAN ACCESS THE DATA BASE PCB02
C           GU        RQDLI                   13
C           PCB       ELEM            PCB02
C   .............
C           TERM      RQDLI
C* NOW USER CAN NO LONGER ACCESS PCB02
C* BEFORE ACCESSING A NEW DATABASE THE FOLLOWING
C* STATEMENTS MUST BE CODED BY THE USER
C           PCB       RQDLI                   13
C           SET       ELEM            BUIB
C           PSBNAME   ELEM   'PSBNAM3'
C* PSBNAME OPTION MUST BE SPECIFIED IN
C* AN ELEM STATEMENT (PSBNAM3 SCHEDULES PCB01 AND PCB02)
C* THIS ESTABLISHES THE ADDRESSABILITY OF THE DATA STRUCTURE
C* DLIUIB AND ITS FIELDS
C                     MOVE UPCBAL     BPSB
C                     CALL 'ILNSAD'
C* DATA STRUCTURE CONTAINING THE ADDRESS LIST OF THE PCBS IS
C* NOW ADDRESSABLE.IN THIS CASE PSBNAM3 CONTAINS
C* PCB01 AND PCB02
C* ESTABLISH THE ADDRESSABILITY FOR BOTH BY MOVE STATEMENTS
C* FOLLOWED BY CALL TO ILNSAD
C                     MOVE PCB01P     BPCB01
C                     MOVE PCB02P     BPCB02
C                     CALL 'ILNSAD'
C* NOW THE USER CAN ACCESS THE DATA BASES PCB01 AND PCB02
C           GU        RQDLI                   13
C           PCB       ELEM            PCB02
C       .............
C* CONTINUE WITH PROGRAM
```

Figure 3-7 (Part 3 of 3). Online RPG II Application Program Examples

DL/I Application Program Coding in a CICS/VS Command Language Environment

The following steps should be observed when coding a DL/I application program in a CICS/VS command language environment:

1. The DL/I I/O area, SSA, COUNT, and FUNCTION should be coded in the WORKING-STORAGE SECTION of a COBOL program and in AUTOMATIC storage of a PL/I program.
2. In application programs not using the UIB, addressability must be established to the TCA to execute the Schedule and TERM calls. To establish this addressability in COBOL, the copy books DFHBLLDS, DFHCSADS, and DFHTCADS must be in the LINKAGE SECTION. Also, the COBOL statement

MOVE CSACDTA TO TCACBAR is required to prime the BLL cell for the TCA. To establish this addressability in PL/I, the copy books DFHCSADS and DFHTCADS must be included in the program before the Schedule call.

3. The list of PCB addresses and PCBs must be coded in the LINKAGE SECTION of a COBOL program and in BASED storage of a PL/I program.

If these three steps are followed, there is no need to code any CICS/VS macros and therefore, no need to execute the macro-level language preprocessor. See Figure 3-1 on page 3-10, Figure 3-3 on page 3-14, and Figure 3-5 on page 3-18 for examples.

CICS/VS Trace Table Entries for DL/I DOS/VS

The following describes the entries placed by DL/I DOS/VS in the CICS/VS trace table. For a complete description of the CICS/VS trace table, refer to the *CICS/VS Application Programmer's Reference Manual.* For a description of the terms and acronyms used in the internal control blocks, refer to *DL/I DOS/VS Diagnostic Guide.*

	Entry ID	Type of Request	Reserved	TCA ID Number	Trace Information
Bytes	0	1-2	3-4	5-7	8-15

	Resource Name	R14 Contents	Time of Day
Bytes	16-23	24-27	28-31

Byte	Meaning
0	Trace code X'F8'indicating DL/I trace entry.
1-2	A code indicating the type of request that was made. There are three type-of -request codes: 'S' - occurring at the completion of a scheduling call. 'D' - occurring at the completion of a data base call. 'T' - occurring at the completion of a termination call.
3-4	Reserved
5-7	The 3-byte task id field contains the CICS/VS transaction id (packed decimal 1-99999)
8-15	An eight-byte field containing data unique to each type of request. The details for each type of request are shown below. Two trace entries are written in the trace table for 'S' and 'D' requests. One trace entry is written for 'T' requests.
16-23	An eight-byte field containing the resource name. This field is blank for DL/I
24-27	A four-byte field containing an address of a DL/I control block or module. Contents of this field are unique to each type of request. The details for each type are shown below.
28-31	Unsigned binary integer representing time-of-day in units of 32 microseconds.

Trace Information

'S' Type of Request:


```
PSBNAME = name of the PSB being scheduled
L/R     = Type of data base
          ' ' = local data base
          '+' = remote data base
          '*' = local and remote data base
```

Mnemonic Status Code = six-character mnemonic status code representing an exit condition as shown below:

Mnemonic	Exit Cond	Meaning
SUCESS NO	00 00	Successfully scheduled PSB not in directory (PDIR) Task
PSB RESCHD	08 01	already scheduled PSB initialization error PSB not in
PSBERR	08 03	program ACT entry Illegal MPS scheduling call DL/I not
PSBATH	08 05	active Data base not opened or stopped Scheduling conflict
ILLEGAL	08 06	with MPS task Undefined return code.
DLIDWN	08 09	
NOTOPN	08 FF	
MPSCFL	0C 01	
UNDEFN	0C 02	
	XX XX	

Exit cond = see complete list of exit conditions in *DL/I DOS/VS Diagnostic Guide.*

'D' Type of Request:

Data Base Name = name of the DMB being accessed.

Call Function Code = see list of call function codes in Chapter 2
DL/I status = see list of DL/I status codes in Chapter 2
Exit cond Exit cond = see complete list of exit conditions in *DL/I DOS/VS Diagnostic Guide*

Note: If a data base call is issued when the task is not scheduled or if any DL/I call is issued when the DL/I is not active, the data base name is replaced by '*ERROR**', andthe PCB status is replaced by '=>'. In the first case, the DL/I status code is X'0808'. In the second case, theDL/I status code is X'08FF'.

'T' Type of Request:

Why Term Used = a six-character description code:

Code	Description
ABEND	The task ended abnormally while scheduled
CANCEL	The task was canceled by operator intervention
DEADLK	The task was ended abnormally by DL/I to resolve a program isolation scheduling deadlock
GETVIS	The task was ended abnormally by DL/I due to a lack of space for program isolation queue elements
RETERM	TERM call issued when task was not scheduled
SYNCPT	Termination was caused by a CICS/VS sync point call
USER	User issued a DL/I TERM or 'T ' call

Exit cond = see complete list of exit conditions in *DL/I DOS/VS Diagnostic Guide.*

Register 14 Contents Field

'S' Type of Request

Entry Description
#1 Address of DL/I System Contents Directory (SCD)
#2 Address of CICS/VS TCA for task

'D' Type of Request

Entry Description
#1 Address of DL/I call parameter list
#2 Address of PCB

'T' Type of Request

Entry Description
#1 Address of DL/I Program Request Handler

Chapter 4. Optional DL/I Programming Functions

This chapter describes several additional capabilities which DL/I makes available to application programs:

- Command codes
- Variable length segments
- Multiple positioning
- Secondary indexing
- Field level sensitivity.
- DL/I System and GSCD calls

These optional facilities provide the experienced user with more powerful and sophisticated techniques for organizing and processing data base structures.

The advantages and disadvantages should be evaluated before making a decision to use these features in an application, since the application program in question, other applications, and the overall DL/I system may be affected. Multiple positioning requires earlier planning for PSB generation; secondary indexing requires both PSB generation and DBD generation planning and implementation, and can have significant performance considerations; field level sensitivity requires both PSB and DBD generation planning and implementation; and the system calls are specific to DL/I and are not supported by IMS/VS.

You should be familiar with the information presented in chapters 1 through 3 of this manual before attempting any of the optional facilities described in this chapter. Additionally, for secondary indexing, you should be familiar with the related information in *DL/I DOS/VS Data Base Administration.*

Command Codes

In Chapter 1, the basic function of the segment search argument (SSA) was defined as identifying a specific data base segment called by an application program. The rules governing the use of SSAs by each DL/I functional call are covered in that discussion. The rules governing the use of qualified and unqualified SSAs in RPG II are also discussed in Chapter 1, “DL/I Application Program for RPG II.”

The command codes are an optional addition to the SSA and provide specification of functional variations applicable to the call function. The command codes and their meanings are:

Code **Meaning**

L Under the parent already established, retrieve the last occurrence of this segment type that satisfies the qualification statement; or, if unqualified, retrieve the last occurrence of this segment type under its parent. If command code ‘L’ is used at the root level, it is disregarded. When used with ISRT calls, the command code only applies to segments with a nonunique sequence field or with no sequence field and with RULES=(,FIRST) or RULES=(,HERE), in which case the rule is overridden.

F Start with the first occurrence of this segment type under its parent in attempting to satisfy this level of the call. With single positioning, it is possible to either back up to the first occurrence of the segment type on which position is established or to back up to the first occurrence of a segment defined earlier in the hierarchy but in the same path as the one on which position is established. With multiple positioning, the ‘F’ command code can be effective in any prior path where position has been established.

When used with ISRT calls, the command code only applies to segments with a nonunique sequence field or with no sequence field and with RULES=(,HERE), in which case that rule is overridden.

Command code ‘F’ is disregarded if it is used at the root level or with GU or GHU calls, in the latter case because a GU or GHU call leads to the same result.

D For retrieval calls with multiple SSAs, move the segment which satisfies the particular SSA in which this command code is present to the user's I/O area. This allows the retrieval in a single call of multiple segments in a hierarchical path. This type of call is referred to as a path call. The first through the last segment retrieved are concatenated in the user's I/O area. Intermediate SSAs may be present without the ‘D’ command code. If so, these segments are not moved to the user's I/O area. The segment name in the PCB is that of the lowest level segment retrieved, or the last level satisfied in the call in case of a not-found condition. Higher level segments having the ‘D’ command code are placed in the user's I/O area even in the not-found case. The ‘D’ is not necessary for the last SSA in the call, since the segment which satisfies the last level is always moved to the user's I/O area. Processing option P must be specified in the PSB generation for any segment for which command code ‘D’ is to be used for retrieval calls. Note that the way in which segments are retrieved is not affected by the ‘D’ command code. The only effect is to move all segments with the ‘D’ command code into the I/O area.

For INSERT calls, the ‘D’ command code allows the insertion in a single call of multiple segments in a hierarchical path. The first segment type in the path to be inserted is designated by the ‘D’ command code. The SSAs for lower level segments in the path need not have the ‘D’ command code set. This use of the ‘D’ command code is not permitted if a logical child segment (see *DL/I DOS/VS Data Base Administration*) is present in the path.

N When a REPLACE call follows a path retrieval call, it is assumed that all segments in the path are being replaced. If any of the segments have not been changed, and, therefore, need not be replaced, the ‘N’ command code may be set at their levels to inform DL/I not to attempt to replace the segments at these levels of the path.

Q The ‘Q’ command code causes DL/I to lock the segment(s) returned by the call to prevent modification by another task.

It provides a facility which permits segments to be enqueued (locked) when the application needs to examine a number of segments and, at the same time, prevent any of them from being modified while the others are being examined. The application can obtain the segments using the ‘Q’ command code and then retrieve them again with the assurance that none of them can be modified until the application terminates or issues a checkpoint.

In IMS, the ‘Q’ command code is always followed by a one-byte field. In order to be compatible with this format, DL/I requires that the ‘Q’ command code always be followed by the character ‘A’ (as:QA). This means that the *second* byte after the ‘Q’ must contain another command code, a left parenthesis, or a blank. For example, when used in combination with command code ‘D’ either of the following forms of coding would be acceptable:

```
*QAD
*DQA
```

Note: The ‘Q’ command code will be ignored by DL/I unless the segment for which it was specified is actually returned to the user (that is: it was used in an SSA with command code ‘D’ or in the lowest level SSA).

U The ‘U’ command code prevents the position from being moved from a segment during a search of its hierarchical dependents. It indicates that no occurrences of the segment type specified in the SSA (other than the segment type upon which position is already established) under the current occurrences of the parent segment type will be used to satisfy the call. If the segment has a unique sequence field, use of this code is equivalent to qualifying the SSA such that it is equal to the current value of the key field.

If position is moved to a level above that at which the ‘U’ code is issued when a call is being satisfied, the code has no effect for the segment type whose parent changed position. In short, if no dependents exist for the segment at which the ‘U’ code is specified, the next higher level segment will be retrieved to satisfy the call. While looking for dependents in this new path, the ‘U’ would then be ignored. Except for key feedback information, there will be no indication that the segment returned is not a dependent of the segment where position was originally held.

The ‘U’ code is especially useful when dependents that are unkeyed or nonunique keyed segments are being processed. The position on a specific occurrence of an unkeyed or nonunique keyed segment can be held by use of this code.

The ‘U’ command is disregarded if it is used at the lowest level SSA in the call or if the SSA is qualified. If used with command code ‘F’ or ‘L’, the ‘U’ command code is disregarded at that level and all lower levels of SSAs for that call. Also, if the SSA or data base position is such that the command code can not be used, it will be ignored.

V The ‘V’ command code is the same as the ‘U’ command code, except that the command code is automatically set at all higher levels in the call. This means that DL/I, in attempting to satisfy this call, cannot move from the existing position at the level at which the ‘V’ is specified, unless the command code is disregarded. See the ‘U’ command code for the condition under which it will be disregarded.

The codes L, D, and Q or F, D, and Q may be used in the same SSA, in any order.

Figure 4-1 indicates which command codes are applicable to which functions. If the command code is used with a function where it is not applicable, it has no effect but no error status code is returned.

Figure 4-1. Command Code Applicability by Function

Command Code	GU GHU	GN GHN	GNP GHNP	DLET	REPL	ISRT	CHKP
D	A	A	A	D	D	A	D
F	D	A	A	D	D	A	D
L	A	A	A	D	D	A	D
N	D	D	D	D	A	D	D
Q	A	A	A	D	D	D	D
U	A	A	A	D	D	A	D
V	A	A	A	D	D	A	D

A = Applicable
D = Disregarded

Figure 4-2 on page 4-5 shows the format of the SSA with the command code.

If command codes are used the ninth character of the SSA must be an asterisk, followed by one or more command codes. These may be used with either a qualified or unqualified SSA.

Note: For IMS compatibility, a blank or ‘C’ appearing in position 10 with an asterisk (*) in position 9 will also be accepted. The asterisk will be ignored and the ‘C’ or blank will act as a normal delimiter and any qualification will be accepted.

If no command code is required for a particular SSA then the ‘(’ or ‘b’ must appear in position 9 as described in Chapter 1, or a null (dummy) command code of one or more dashes must begin in position 10.

Note: The null command code may be used to keep the format of SSAs consistent throughout an application program. During execution of the program a given SSA can then be used for different purposes at different points in the program by replacing the null command code with the appropriate command code characters.

QUALIFIED SSA

	(optional)			(optional)												
elements	Segment Name	Com- mand Codes		BOOLEAN STATEMENT												
				Begin Qualif.	Qualification Statement # 1			Boolean Operator	Qualification Statement # 2			Boolean Operator	Qualification Statement # n			End Qualif.
contents	Name of Segment Type	•	Code Char- acters	'('	Field Name	RO	Compar. Value	'&' '|' or '+' '|'	Field Name	RO	Compar. Value	'&' '|' or '+' '|'	Field Name	RO	Compar. Value	')'
number of bytes	8	1	Var.	1	8	2	1 to 255	1	8	2	1 to 255	1	8	2	1 to 255	1

UNQUALIFIED SSA

	(optional)			
elements	Segment Name	Com- mand Codes		Ending Delim- iter
contents	Name of Segment Type	•	Code Char- acters	'('
number of bytes	8	1	Var.	1

Figure 4-2. Format of SSAs with Command Codes

Variable Length Segments

DL/I DOS/VS can process variable length segments. This feature is intended for use in processing variable length text of descriptive data. A complete discussion of this feature can be found in the *DL/I DOS/VS Data Base Administration*.

Variable length segments are supported for HDAM and HIDAM data bases only. The application program processes variable length segments in the same way as fixed length segments, except that the manipulation of the length field is the responsibility of the application programmer.

The format of the variable length segment in the user input/output area contains, in the first 2 bytes, the binary value describing the segment size, followed by user data. This 2-byte field describes the segment length as the user sees it, including the size field itself. The minimum value valid in this field is 4.

Segment retrieval, including path call, follows normal retrieval rules. If all or any part of a field is not present because the segment is shorter than the fields defined within it, DL/I calls using SSAs qualified by these fields produce a GE status code. After the segment has been accessed, replacement of existing data may occur with a REPL call. If the segment length has not changed, a one-for-one replacement takes place. If the length of the data changes, either increasing or decreasing, the user must change the value in the segment size field. For an insert operation, segment size is placed in the appropriate field, followed by the corresponding quantity of data, and the ISRT call is issued. Specification in the size field at execution time of a value less than either 4, or any length greater than 4 that was specified as minimum during DBD generation, is ignored. In that case the segment is written with a length equal to the greater of 4 or the specified minimum, but the segment size field remains unaltered.

Since the segment size field is actually part of the segment, all starting positions must be related to the first position of the variable length segment, not the start of the user data. The existence, alignment, and content of any defined data fields which follow the required 2-byte binary field describing the segment length are the responsibility of the user.

Multiple Positioning With DL/I Calls

Two alternatives are provided by DL/I regarding the current position in the data base, namely single or multiple positioning. This option is specified during PSB generation.

When *single positioning* is specified for a PCB, DL/I maintains only one position in that data base for that PCB. This is the position that is used in attempting to satisfy all subsequent GN (GET NEXT) calls.

If *multiple positioning* is specified, DL/I maintains a unique position in each hierarchical path in the data base, that is, for different segment types under the same parent.

With single positioning, whenever a segment is obtained, position for all dependent segments and all segments on the same level is cleared. The position clearing procedure is done for not-found conditions also. With multiple positioning, whenever a segment is obtained, position for all dependent segments is cleared, but position for segments at the same level is maintained. The control blocks in either case are the same (multiple positioning does not require more storage). There is no significant performance difference, even though in some cases multiple positioning requires slightly more processing time.

DL/I attempts to satisfy GN calls from the existing position by analyzing segments in a forward direction only. Since multiple positioning allows position to be maintained at each level in all hierarchical paths under the current parent position rather than at each level in only one hierarchical path, the GN call is satisfied using the existing position established on the path of the hierarchy in which the GN call is qualified. If the GN call is not qualified, DL/I uses the position established on a path by the prior call.

The only effect multiple positioning has on GU (GET UNIQUE) and INSERT calls is when these calls have missing SSAs in the hierarchical path. The missing levels are completed by the system according to the rules for GET calls.

DELETE and REPLACE calls are not affected by single or multiple positioning. Rather, the effect is on the GET HOLD calls, as described above, since a GET HOLD call must be issued prior to a DELETE or REPLACE call.

The following examples compare the results of single and multiple positioning, using the data base in Figure 4-3 on page 4-8.

Call Sequence	Result with Single Positioning	Result with Multiple Positioning
Example 1:		
GU A (KEY=A1)	A1	A1
GN B	B11	B11
GN C	C11	C11
GN B	B21	B12
GN C	C21	C12
Example 2:		
GU A (KEY=A1)	A1	A1
GN C	C11	C11
GN B	B21	B11
GN B	B22	B12
GN C	C21	C12
Example 3:		
GU A (KEY=A1)	A1	A1
GN B	B11	B11
GN C	C11	C11
GN D	D111	D111
GN E	E111	E111
GN B	B21	B12
GN D	D221	D112
GN C	C under next A	C12
GN E	E under next A	E121

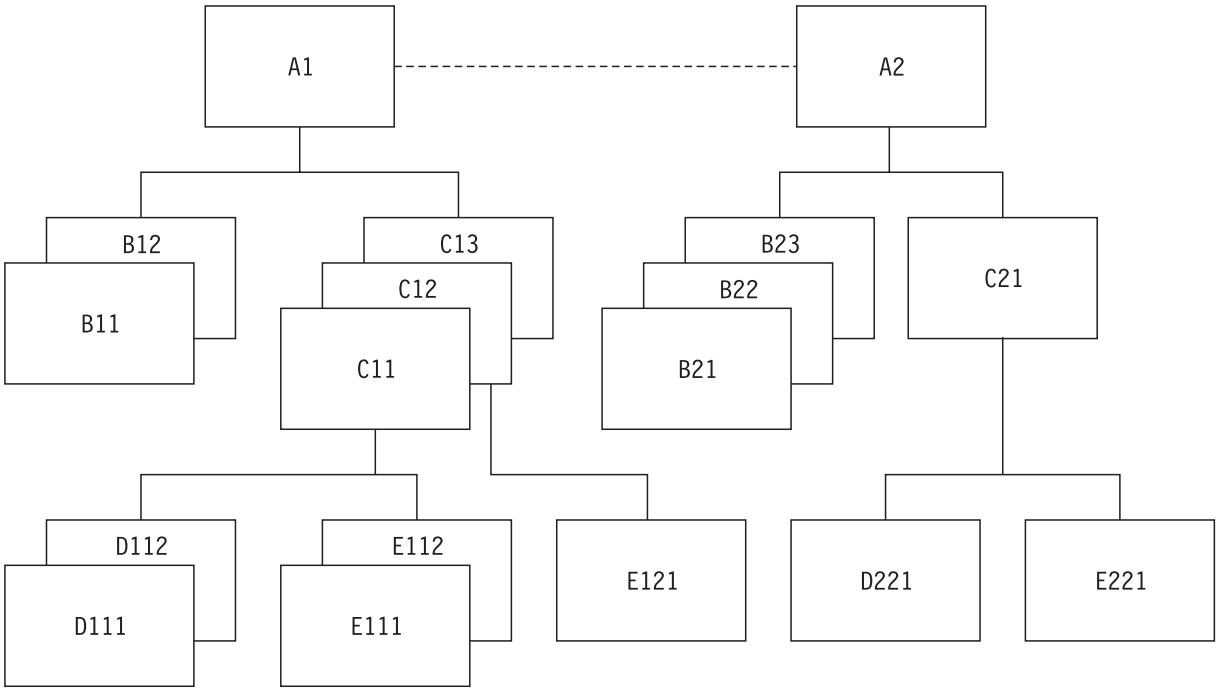

Figure 4-3. Assumed Data Base to Illustrate Single and Multiple Positioning

Use of Multiple Positioning

Multiple positioning may be useful for application program development in two general types of situations.

1. *Increased Data Independence:* Multiple positioning allows the user to develop application programs using GN or GNP (GET NEXT WITHIN PARENT) calls and INSERT and GU calls with missing levels in a manner independent of the relative order of segment types defined at the same level in the data base structure. Hence, if performance could be improved by changing the relative order of segment types, and all application programs that access the segment types use multiple positioning, then the change could be made with no impact on previously produced application programs. It should be noted, however, that this ability depends on the proper use of the calls relevant to multiple positioning (GN, GNP and incompletely specified INSERT and GU calls). Multiple positioning also presents an increased responsibility for the application programmer to keep track of all positions maintained by DL/I. Other alternatives exist to decrease an application program's exposure to future changes, for instance increased use of explicit call specifications when possible. These alternatives may require additional application program coding. Such trade-offs must be determined in the user's own environment.
2. *Parallel Processing of Dependent Segment Types:* When an application program needs to process dependent segment occurrences in parallel, that is, to switch alternately from one dependent segment type to another under a parent, the program may specify multiple positioning. An alternative parallel processing technique would be to give the program two or more PCBs using the same data base. Under this alternative, the program processes the data base as though it were two or more different data bases. This approach may be more useful if the way a segment is updated depends on the analysis of other subsequent segments. The use of multiple PCBs may decrease the number of GET HOLD calls required, but may increase the number of other calls required to maintain proper positioning in two or more data bases. Internal control block processing also increases with each added PCB. The decision to choose multiple positioning or multiple PCBs for application programming must be evaluated in the user's own environment.

It should be emphasized strongly that multiple positioning uses position differently from single positioning. If an application program changes from one option to the other, the user must not assume the same results will be produced. An application program must be developed for one alternative or the other.

Mixing Calls With and Without Segment Search Arguments and Multiple Positioning

The multiple positioning feature is intended to be used for DL/I requests which specify SSAs, therefore providing for parallel processing and increased data independence. However, retrieval calls without SSAs also may be used when multiple positioning is specified to accomplish a sequential retrieval of segment occurrences independent of segment types, if the following considerations are observed:

1. Certain restrictions apply when GET calls without SSAs are mixed with DL/I requests that specify SSAs in processing a single data base record.

   Example (using Figure 4-3):

CALL	**Result with Multiple Positioning**
GU A(KEY=A1)	Retrieves A1
GN C	Retrieves C11
GN B	Retrieves B11
GN B	Retrieves B12
GN	Unpredictable

   The GN calls may not attempt to retrieve occurrences of the C segment type because a position has already been established on this segment type using the multiple positioning feature. The result of the call is unpredictable.

2. When segment types have previously been processed with GET calls not specifying SSAs, a position is established on the last retrieved segment type and its parent (hierarchical path). Multiple positions are no longer maintained.

CALL	**Result with Multiple Positioning**
GU A(KEY=A1)	Retrieves A1
GN C	Retrieves C11
GN B	Retrieves B11
GN C	Retrieves C12
GN	Retrieves E121
GN B	Unpredictable

   Multiple positions on B are no longer maintained. The result of the GN B call is unpredictable.

It should be noted that although the mixed use of GET calls with and without SSAs in processing a single logical data base record may be valid for some types of parallel processing, it may decrease the degree of data independence created by the use of multiple positioning. The implications of the two restrictions stated above should be carefully considered before application programming is based upon mixed use of retrieval with and without SSAs within a single data base record. If possible, GET calls without SSAs should be limited to GNP calls to avoid potentially inconsistent retrieval situations.

Secondary Indexing

Secondary indexing is a data base structuring technique which ordinarily would concern only the data base administrator of a DL/I installation.

However, in those installations which employ secondary indexing, two factors make it desirable that the experienced application programmers have some familiarity with the secondary indexing facility. First, secondary indexes are used to establish alternate entries to physical or logical data bases for application programs. The existence of a secondary index on a segment can affect the manner in which DL/I processes the SSAs for that segment. Second, secondary indexes can be processed as data bases themselves.

A complete discussion of secondary indexing can be found in the *DL/I DOS/VS Data Base Administration,* that addresses the access methods and the design and implementation (as opposed to processing) aspects of data base structures. This is the necessary context for a discussion of secondary indexing, and the application analyst or programmer who is interested in this facility is referred to that document for the information. The discussion which follows simply summarizes the characteristics of secondary indexing and describes the effect of secondary indexing on data base processing.

An index data base is an auxiliary data base used to locate data in a HDAM, HD randomized, or HIDAM data base. HIDAM and HD primary index always have one INDEX data base which is called a primary index and which indexes only on the sequence field of the root segment. All other indexes are secondary indexes, and they may index segment types at any level of the data base structure including root segments. HSAM, HISAM and INDEX data bases cannot be indexed.

Logical data bases can have secondary indexes, that is, secondary indexes existing for a physical data base that participates in a logical relationship can be used when accessing the logical data base. However, if a segment other than the root segment of the physical data base is indexed and accessed through this index, the application program cannot be sensitive to any logically related segments.

Unlike primary indexes as used with HIDAM, secondary indexes can:

- Index any field or combination of fields (not necessarily contiguous) in a segment of a HIDAM, HD, or HDAM data base at any level.
- Carry, in addition to the indexed data from the indexed data base and pointers, other source data which consists of system-maintained copies of data from the indexed data base.
- Include user-maintained data in addition to the system-maintained data.
- Be created as sparse indexes through a system-provided function for suppressing the creation of an index entry for certain data base records by allowing user options and/or entries.
- Be processed as data bases themselves, in addition to serving as alternative access paths to a data base. When processed as data bases, only GET and REPLACE calls are allowed, with the REPLACE calls only being used to change user-maintained data.

A secondary index can be used:

- To sequentially process all or a part of a data base in an order which is different from its primary processing sequence.
- To sequentially process a data base as if its structure had been inverted, that is, the data base appears to be a differently structured data base.
- To randomly retrieve and process single segments faster than with the primary addressing scheme, if the secondary index provides a unique identification of the requested segment.
- As a data base itself in order to perform scan processing in the index rather than in the indexed data base.
- To access a segment in a data base based on data located in one of its dependent segments in the same physical data base.

An indexed field may be referenced in an SSA, using a specific field name, that is, the name specified in the XDFLD statement during DBD generation, with the result that the index rather than the data base is inspected in order to satisfy a call. In this case the secondary index must be designated as main processing sequence of the data base. This is done by the PROCSEQ operand in the PSB generation for an application program. Depending on the level of the indexed segment, this may involve an automatic structure inversion of the indexed data base. The inverted structure must be reflected in the PSB control statements. Any HDAM, HIDAM, or HD data base can be inverted and processed using a secondary index, provided the applicable restrictions are observed. This processing mode requires careful design considerations. The names of the indexed fields must be defined with appropriate DBD statements. DL/I will ensure that the user is sensitive to the index source segment, that is, the indexed data, before allowing use of the referenced index in an SSA.

Field Level Sensitivity

The field level sensitivity feature makes it possible for the user to specify only those fields in the physical definition of a given segment that are to be included in his application's view of that segment, while remaining insensitive to the other fields in the segment. The locations of the chosen fields within the new view of the segment may also be different from their locations within the physical definition. This makes it possible for different application programs to have entirely different views of the same segment. This specification is done at PSB generation time and causes DL/I to automatically map the chosen fields from the physical segment into the user's view, during execution.

In addition, the following capabilities are made available through field level sensitivity support:

- *Virtual fields* - it is possible to identify fields in the user's view of a segment that don't exist in the physical segment.
- *Automatic data format conversion* - it is possible to have DL/I automatically change the format of the physical data to another chosen format in the user's view, for a particular application program.
- *User field exit routine* - it is possible to specify a user-written routine that will be given control each time a given field is retrieved or stored.
- *Dynamic segment expansion* - it is possible to add fields to a segment in the user's view, without reloading the data base or re-compiling other application programs that access the segment.

Virtual Fields

At PSB generation time, fields may be identified in a user's view of a segment that do not exist in the physical segment. At the same time, an initial value may be assigned to the field and/or the name of a user-written routine, that can be used to create the field, may be specified. If both an initial value and a routine are specified, DL/I will insert the initial value in the user's view of the field before the routine is called during a GET procedure for the field. If a routine is specified, it will be called on both GETs and PUTs involving the field. See “User Field Exit Routine” later in this chapter for further detail.

Automatic Data Format Conversion

If the type of data to be maintained in a given field was defined during DBD generation, that data can be automatically converted to another type for a particular application program. This is accomplished through field level sensitivity support by specifying a different data type in the SENFLD macro at PSB generation time, for the user's view of the field. The data types that may be specified are:

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

The automatic conversions that are supported are:

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

Conversion of data types E, D, and L is not supported.

Notes:

1. Conversion status codes
   Errors detected during the process of automatic data format conversion will set status codes as follows:

   When significant digits are lost during the conversion of a numeric field because the “to.” field was too small, status code “KA” is set.

   When non-blank characters are lost during the conversion of a character field because the “to” field was too small, status code “KB” is set.

   When a “from” field character is detected as being an invalid packed decimal or zoned decimal character, status code “KC” is set.

   Status code “KD” is returned when the user has supplied a field exit routine (see User Field Exit Routine, below) and a conversion was requested that is not supported by DL/I.

2. For more detailed information, see *DL/I DOS/VS Resource Definition and Utilities.*

User Field Exit Routine

At PSB generation time, the name of a user-written routine may be specified. This must be the name by which the routine is cataloged in the DOS/VSE core image library. The routine will be given control whenever the associated field is referenced in either a GET or a PUT access.

The routine may be used to perform data type conversions not supported by DL/I, to create values for virtual fields, or to correct a conversion problem. If the user routine corrects the problem, it should reset the status code to blank. Setting the code to a non-blank will result in the termination of the request with a status code of Kx, where ‘x’ is the code set by the user routine.

Dynamic Segment Expansion

Fields may be added to a segment in the physical data base without unloading and reloading the data base, and without re-compiling other application programs that access the segment. This is accomplished by fulfilling certain requirements during DBD generation and PSB generation. For a complete discussion, see the *DL/I DOS/VS Resource Definition and Utilities.*

Further Field Sensitivity Considerations

- SSAs - Field information supplied in an SSA should be in the format of the user's view of the field. The field identified in the SSA, and any subfields that the application is sensitive to, will be converted to the physical view before the compare is done. Any fields overlapping either end of the field identified in the SSA will not be converted.
- Key feedback area - The information returned in the key feedback area will be unconverted.

DL/I System and DSCD Calls

The DL/I system calls can be used to control the DL/I system. These system calls are supported only for CICS/VS online programs written in assembler language and use the DL/I CALLDLI macro interface.

The GSCD (get SCD) call can be used to obtain buffer pool statistics in a batch environment. This call returns the address of the system contents directory SCD control block, which contains pointers to the major control blocks and entry point addresses of the primary DL/I modules. The GSCD call may be issued in an assembler language or PL/I program.

For more information about the DL/I system and GSCD calls, see *DL/I DOS/VS Data Base Administration*.

Appendix A. /INSERT Statement in RPGII

/INSERT may be used to include data structures, program pieces, etc. from source statement libraries. The inserted text must be *untranslated* source and must not itself contain /INSERT statements.

Note: For RPG II the /INSERT statement is a facility of the Translator and, therefore, the inserted text is untranslated source. For COBOL, PL/I, and Assembler, COPY or INCLUDE is a language facility and, therefore, the inserted text is translated source.

Figure A-1. Format of the /INSERT Statement

Position	Contents
1-5	see the publication, *DOS/VS RPG II Language*
6	H|F|E|L|I|C|O
7-13	/INSERT
14	blank
15	sublibrary-name|blank
16	•
17-24	book-name
25-49	blank
50-74	comment
75-80	see the publication, *DOS/VS RPG II Language*

sublibrary-name
Name of the sublibrary from which the insertion should be made. If no sublibrary is specified the name is defaulted to R.

book-name
Name of sublibrary member to be inserted.

Function of the /INSERT Statement
Inserts the contents of the book specified by book-name, from the sublibrary specified by sublibrary-name, in place of the /INSERT statement.

Glossary

A number of terms and phrases used in describing DL/I DOS/VS are either new to most readers, or have new meanings. To improve the readability and your understanding of this and other DL/I DOS/VS publications, the significant and important terms are defined in this Glossary. Some of the definitions refer to the representative DL/I hierarchical structure shown in Figure X-1.

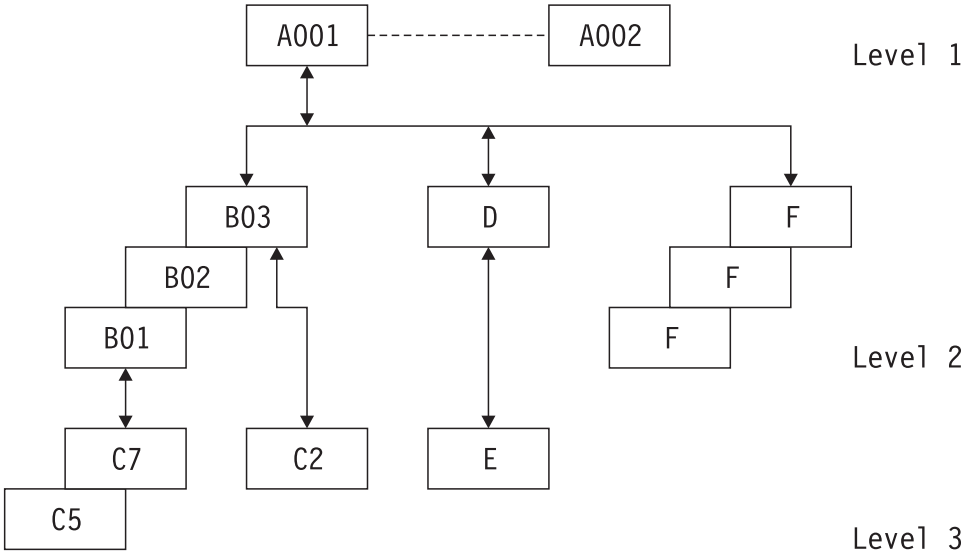

Figure X-1. Representative DL/I Hierarchical Structure

ACB. (1) Application control blocks (DL/I). (2) Access method control block (VSAM).

ACBGEN. Application control block generation.

access method control block (ACB).. A control block that links a program to a VSAM data set.

access method services.. A multifunction utility program that defines VSAM data sets (or files) and allocates space for them, and lists data set records and catalog entries.

ACT. Application control table.

addressed direct access. In systems with VSAM, the retrieval or storage of a data record indentified by its relative byte address, independent of the record's location relative to the previously retrieved or stored record. (See also *keyed direct access, addressed sequential access, keyed sequential access,* and *relative byte address.)*

addressed sequential access. The retrieval or storage of a VSAM data record relative to the previously retrieved or stored record. (See also *keyed sequential access, addressed direct access,* and *keyed direct access.)*

aggregate. See *data aggregate.*

anchor point (AP). See *root anchor point.*

application control blocks. The control blocks created from the output of DBDGEN and PSBGEN, e.g., a DMB of an internal PSB created by the ACB utility program.

application control block generation (ACBGEN). The process by which application control blocks are created.

application control table (ACT). A DL/I online table describing those CICS application programs that utilize DL/I.

argument. (1) (ISO)[1] An independent variable. (2) (ISO)[1] Any value of an independent variable . (3) Information, such as names, constants, or variable values included within the parentheses in a DL/I command.

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1.

attribute. A property of an entity expressing a value. Synonymous with *field*.

backout. The process of removing all the data base updates performed by an application program that has terminated abnormally. See also *dynamic backout.*

batch checkpoint/restart. The facility that enables batch processing programs to synchronize checkpoints and to be restarted at a user-specified checkpoint.

batch processing. A processing environment in which data base transactions requested by applications are accumulated and then processed periodically against a data base.

Boolean operator. (1) (ISO)[1] An operator, each of the operands of which and the result of which, take one of two values. (2) An operator that represents symbolically relationships, such as AND, OR, and NOT, between entities.

business process. A defined function of a business enterprise usually interrelated through information requirements with other business processes. For example, personnel management is the business process responsible for employee welfare from pre-hire through retirement. It is related to the accounting business process through payroll.

CA. Control area.

call. (1) (ISO)[1] The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (2) (ISO)[1] In computer programming, to execute a call. (3) The instruction in the COBOL, PL/I, or Assembler program that requests DL/I services. For RPG II, see *RQDLI command.* See also *command.*

control area (CA). A collection of control intervals. Used by VSAM to distribute free space.

checkpoint. A time at which significant system information is written on the system log, and optionally, the system shut down.

child. Synonymous with *child segment.*

child segment. A segment one level below the segment which is its parent, with a direct path back up to the parent. Depending on the structure of the data base, a parent may have many children; however, a child has only one parent segment. Referring to Figure G-1:

- All the B, D, and F segments are children of A-001.
- C-5 and C-7 are children of B-01 (and A-001) but not children of the other B segments.
- B-02 has no children.

See also *logical child* and *physical child.*

CI. Control interval.

command. The statement in DL/I High Level Programming Interface (HLPI) that requests services for application programs written in COBOL or PL/I. See also *call.*

command code. An optional addition to the SSA that provides specification of a function variation applicable to the call function.

concatenated key. The key constructed to access a particular segment. It consists of the key fields, including that of the root segment and successive children down to the accessed segment.

control interval (CI). (1) A fixed length amount of auxilary storage space in which VSAM stores records and distributes free space. (2) The unit of information transmitted to or from auxillary storage by VSAM.

data aggregate. A group of data elements that describe a particular entity. Synonymous with *segment.* See also *data element.*

data base (DB). (1) (ISO)[1] A set of data, part of the whole of another set of data, and consisting of at least one file, that is sufficient for a given purpose or for a given data processing system. (2) A collection of data records comprised of one or more data sets. (3) A collection of interrelated or independent data items stored together without unnecessary redundancy to serve one or more applications. See *physical data base* and *logical data base.*

data base administration (DBA). The tasks associated with defining the rules by which data is accessed and stored. The typical tasks of data base administration are outlined in *DL/I DOS/VS Data Base Administration.*

data base administrator (DBA). The person in an installation who has the responsibility (full or part time) for technically supporting the use of DL/I.

data base description (DBD). A description of the physical characteristics of a DL/I data base. One DBD is generated and cataloged in a core image library for each data base that is used in the installation. It defines the structure, segment keys, physical organization, names, access method, devices, etc., of the data base.

data base integrity. The protection of data items in a data base while they are available to any application program. This includes the isolation of the effects of concurrent updates to a data base by two or more application programs.

data base organization.. The physical arrangment of related data on a storage device. DL/I data base organizations are hierarchical direct (HD) and hierarchical sequential (HS). See *hierarchical direct organization* and *hierarchical sequential organization.*

data base record. A collection of DL/I data elements called segments hierarchically related to single root segments. Referring to Figure G-1, A-001, B-01, C-5, C-7, B-02, B-03, C-2, D, E, F, F, F constitute a data base record.

data base reorganization. The process of unloading and reloading a data base to optimize physical segment adjacency, or to modify the DBD.

data communication (DC). A program that provides terminal communications and automatic scheduling of application programs based on terminal input. For example, CICS/DOS/VS.

data dictionary. (1) A centralized repository of information about data, such as its meaning, relationship to other data, usage, and format. (2) A program to assist in effectively planning, controlling, and evaluating the collection, storage, and use of data. For example, DOS/VS DB/DC Data Dictionary.

data element. The smallest unit of data that can be referred to. Synonymous with *field*. See also *data aggregate.*

data field. Synonymous with *field.*

data independence. (1) The concept of separating the definitions of logical and physical data such that application programs do not depend on where or how physical units of data are stored. (2) The reduction of application program modification in data storage structure and access strategy.

data management block (DMB). The data management block is created from a DBD by the application control blocks creation and maintenance utility, link edited, and cataloged in a core image library. The DMB describes all physical characteristics of a data base. Before an application program using DL/I facilities can be run, one DMB for each data base accessed, plus a PSB for the program itself, must be cataloged in a core image library. The DMBs and the associated PSB are automatically loaded into main storage from the core image library at the beginning of the application program execution (their loading is controlled by the parameter information supplied to DL/I at the beginning of program execution).

data set. A named organized collection of logically related records. They may be organized sequentially, as in the case of DOS/VSE SAM, or in key entry sequence, as in the case of VSE/VSAM. Synonymous with *file.*

data set group (DSG). A control block linking together a data base with the data sets comprising this DL/I data base.

DB. Data base.

DBA. (1) Data base administration. (2) Data base administrator.

DBD. Data base description.

DBDGEN. Data base description generation -- the process by which a DBD is created.

DB/DC. Data base/data communication.

DC. Data communication.

dependent segment. A DL/I segment that relies on at least the root segment (or on another segment at a level immediately above its own) for its full hierarchical meaning. Synonymous with *child segment.*

destination parent. The physical or logical parent segment reached by the logical child path.

device independence. The concept of writing application programs such that they do not depend on the physical characteristics of the device on which data is stored.

DIB. DL/I interface block.

direct access. The retrieval or storage of a VSAM data record independent of the record's location relative to the previously retrieved or stored record. (See also *address direct access* and *keyed direct access*). Contrast with *sequential access.*

distributed data. The ability of DL/I application programs to access a data base that is resident on another processor.

distributed free space. See *free space.*

DL/I interface block (DIB). Variables automatically defined in an application program using HLPI to receive information passed to the program by DL/I during execution. Contrast with *PCB mask.*

DMB. Data management block.

DSG. Data set group.

DTF. Define the file -- a control block that connect a program to a SAM data set.

dynamic backout. A process that automatically cancels all activities performed by an application program that terminates abnormally.

entity. A item about which information is stored. It has properties that can be recorded. Information about an entity is a record.

entry sequenced data set (ESDS). A VSAM data set whose records are physically in the same order as they were put in the data set. It is processed by addressed direct access or addressed sequential access and has no index. New records are added at the end of the data set.

ESDE. Entry sequenced data set.

exclusive intent. The scheduling intent type that prevents an application program from being scheduled concurrently with another application program. See *scheduling intent.*

FDB. field description block.

field. (1) (ISO)[1] In a record, a specified area used for a particular catagory of data, for example, in which a salary rate is recorded. (2) a unique or nonuique area (as defined during DBDGEN) within a segment that is the smallest unit of data that can be referred to. (3) any designated portion of a segment. (4) see also *key field.*

field level sensitivity. The ability of an application program to access data at the field level. See *sensitivity.*

file. (ISO)[00] A set of related records treated as a unit. See also *data set.*

forward. Movement in a direction from the beginning of the data base to the end of the data base, accessing each record in ascending root key sequence, and accessing the dependent segments of each root segment from top to bottom and from left to right. Referring to Figure G-1, forward accessing of all the segments shown would be in the following sequence: A-001, B-01, C-5, C-7, B-02, B-03, C-2, D, E, F, F, F, A-002.

free space. Space available in a VSAM data set for inserting new records. The space is distributed throughout a key sequenced data set (KSDS) or left at the end of an entry sequenced data set (ESDS). Synonymous with *distributed free space.*

free space anchor point. A fullword at the beginning of a control interval pointing to the first free space element in this CI.

free space element. In HD data bases, the portions of direct access storage not occupied by DL/I segments are called and marked as free space elements.

FSA. free space anchor point.

FSE. free space element.

HD. Hierarchical direct.

HDAM. Hierarchical direct access method.

HIDAM. Hierarchical indexed direct access method

HIDAM index. A data base that consists of logical DL/I records, each containing an image of the key field of a HIDAM root segment. A HIDAM index data base consists of one VSAM KSDS (keyed sequenced data set).

hierarchic sequence. The sequence of segment occurrences in a data base record defined by traversing the hierarchy from top to bottom, front to back, and left to right.

hierarchical direct access method (HDAM). Provides for direct access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a randomizing routine. An HDAM data base consists of one VSAM entry sequence data set (ESDS).

hierarchical direct organization. An organization of DL/I segments of a data base that are related by direct addresses and may be accessed through an HD randomizing routine or an index.

hierarchical indexed direct access method (HIDAM). Provides for indexed access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a HIDAM index data base. A HIDAM data base consists of one VSAM Entry Sequenced Data Set (ESDS) and its associated index.

hierarchical indexed sequential access method (HISAM). Provides for indexed access to a DL/I data base. A HISAM data base consists of one VSAM key sequenced data set (KSDS) and one VSAM entry sequenced data set (ESDS).

hierarchical sequential access method (HSAM). The segments of a DL/I HSAM physical data base record are arranged in sequential order with the root segments followed by the dependent segments. HSAM data bases are accessed by the DOS/VSE sequential access method (SAM).

hierarchical sequential organization. An organization of DL/I segments of a data base that are related by physical adjacency.

hierarchy. (1) An arrangement of data segments beginning with the root segment and proceeding downward to dependent segments. (2) A“tree” structure.

high level programming interface (HLPI). A DL/I facility providing services to application programs written in either COBOL or PL/I Optimizer language through commands.

HISAM. Hierarchical indexed sequential access method.

HLPI. High level programming interface.

HS. Hierarchical sequential.

HSAM. Hierarchical sequential access method.

index data base. An ordered collection of DL/I index entries (segments) consisting of a key and a pointer used by VSAM to sequence and locate the records of a key sequenced data set (KSDS). Organized as a balanced tree of levels of index.

index data set. Synonymous with *index data base.*

index pointer segment. The segment that contains the data and pointers used to index the index target segments.

index record. A system-created collection of VSAM index entries that are in collating sequence by the key in each of the entries.

index segment. The segment in the index data base that contains a pointer to the segment containing data (the indexed segment). Synonymous with *index pointer segment.*

index set. The set of VSAM index levels above the sequence set. An entry in a record in one of these levels contains the highest key entered in an index record in the next lower level and a pointer that indicates the record's physical location.

index source segment. The segment containing the data from which the indexing segment is built.

index target segment. The segment pointed to by a secondary index entry, that is, by an index pointer segment.

indexed segment. A segment that is located by an index. Synonymous with *index target segment.*

intersection data. Any user data in a logical child segment that does not include the logical parent's concatenated key.

inverted file. In information retrieval, a method of organizing a cross-index file in which a key identifies a record. The items pertinent to that key are indicated.

key. (1) (ISO)[1] One or more characters within a set of data that contains information about that set, including its identification. (2) The field in a segment used to store segment occurrences in sequential order. (3) A field used to search for a segment. See *primary key* and *secondary key.* (4) Synonymous with *key field* and *sequence field.*

Note: A segment may or may not have a key, that is, a sequence field. All root segments, except for HSAM and simple HSAM data bases, must have keys. DL/I ensures that multiple segments of the same type that have keys are maintained in strict ascending sequence by key. The key may be located anywhere within a segment; it must be in the same location in all segments of the same type within a data base. The maximum sizes for keys are 236 alphameric characters for root segments and 255 for all dependent segments. Keys provide a convenient way to retrieve a specific occurrence of a segment type, maintain the uniqueness and sequential integrity of multiples of the same segment type, and determine under which segment of a group of multiples new dependent segments are to be inserted. Keys should normally be prescribed for all segment types; the exceptions being if there will never be multiples of a particular type or if a particular segment type will never have dependents.

key field. The field is a segment used to store segment occurrences in sequential ascending order. A key field is also a search field. Synonymous with *key* and *sequence field.*

key sequenced data set (KSDS). A VSAM file whose records are loaded in key sequence and controlled by an index. See also *keyed direct access* and *keyed sequential access.*

keyed direct access. The retrieval or storage of a data record by use of an index that relates the record's key to its physical location in the VSAM data set, independent of the record's location relative to the previously retrieved or stored record. See also *addressed direct access, keyed sequential access,* and *addressed sequential access.*

keyed sequential access. The retrieval or storage of a VSAM data record in its collating sequence relative to the previously retrieved or stored record, by the use of an index that specifies the collating sequence of the records by key. See also *addressed sequential access, keyed direct access,* and *keyed sequential access.*

KSDS. Key sequenced data set.

level. (1) (ISO)[1] The degree of subordination of an item in a hierarchic arrangement. (2) Level is the depth in the hierarchical structure at which a segment is located. Roots are always the highest level and the segments at the bottom of the structure are the lowest level. The maximum number of levels in a DL/I data base is 15. For purposes of documentation and reference, the levels are numbered from 1 to 15, with

the root segments being level number 1. Referring to Figure G-1:

- Three levels are shown.
- The A segments (roots) are at the highest level (Level 1).
- The C and E segments are at the lowest level (Level 3).

local system. (1) A specific system in a multisystem environment. Contrast with *remote system.* (2) The system in a multisystem environment on which the application program is executing. The local application may process data from data bases located on both the same (local) system and another (remote) system.

local view. A description of the data that an individual business process requires. See *system view.*

logical. When used in reference to DL/I components, logical means that the component is treated according to the rules of DL/I rather than physically as it may exist, or as it may be organized, on a physical storage device. For example, a logical DL/I record (a root segment and all of its dependent segments grouped) might be contained on several physical records or blocks on a storage device, and because of prior insertions and deletions, the segments might be in a different physical sequence than that by which they are retrieved logically for the application program by DL/I.

logical child. A pointer segment that establishes an access path between its physical parent and its logical parent. It is a physical child of its physical parent; it is a logical child of its logical parent. See also *logical parent* and *logical relationship.*

logical data base. A data base composed of one or more physical data bases representing a hierarchical structure derived from relationships between data segments that can be different from the physical structure.

logical data base record. (1) A set of hierarchically related segments of one or more segment types. As viewed by the application program, the logical data base record is always a hierarchic tree structure of segments. (2) All of the segments that exist hierarchically dependent on a given root segment, and that root segment.

logical data structure. A hierarchic structure of segments that is not based solely on the physical relationship of the segments. See also *logical relationships.*

logical parent. The segment a logical child points to. A logical parent segment can also be a physical parent. See also *logical child* and *logical relationship.*

logical relationship. A user defined path between two segments; that is, between logical parent and logical child, which is independent of any physical path. Logical relationships can be defined between segments in the same physcial data base hierarchy or in different hierarchies.

logical twins. All occurrences of one type of logical child with a common logical parent. Contrast with *physical twin.* See also *twin segment.*

MPS. Multiple partition support

multiple partition support (MPS). Multiple partition support provides a centralized data base facility to permit multiple applications in different partitions to access DL/I data bases concurrently. MPS follows normal DL/I online conventions in that two programs cannot both update the same segment type in a data base concurrently. (With program isolation, two programs can concurrently update the same segment type; however, they cannot concurrently update the same segment. See *program isolation.*) However, two or more programs can retrieve from a data base while another program updates it. If one program has exclusive use of a data base, no other program can update it or retrieve from it.

multiple SSA. A series of segment search arguments (SSAs) included in a DL/I call to identify a specific segment or path. See also *segment search argument.*

object segment. The segment at the lowest hierarchical level specified in a particular command. See also *path call.*

online. A operating environment in which DL/I is used with CICS/DOS/VS (or another data communication program) to permit end-users of application programs to access and store information in a data base through terminals.

option. A command keyword used to qualify the requested function.

parent. Synonymous with *parent segment.*

parent segment. (1) A segment that has one or more dependent segments. Contrast with *child.* (2) A parent is the opposite of a child, or dependent segment, in that dependent segments exist directly beneath it at lower levels. A parent may also itself be a child. Referring to Figure G-1:

- A-001 is the parent of all B, C, D, E, and F segments.
- D is a parent of E, yet a child of A.
- B-02 is not a parent.
- None of the level 3 segments are parents.

parentage. Extablishment in a program of a particular parent as the beginning point for the use of the get next in parent (GNP) or get hold next in parent (GHNP) functions. Parentage can only be established by issuing successful GU, GHU, GN, or GHN calls, or GET UNIQUE or GET NEXT commands.

PATH. The chain of segments within a record that leads to the currently retrieved segment. The formal path contains only one segment occurrence from each level from the root down to the segment for which the path exists. The exact path for each retrieved segment is returned in the following fields of the PCB:

Field 2	Segment hierarchy level indicator
Field 6	Segment name feedback area
Field 7	Length of key feedback area
Field 9	Key feedback area, containing the concatenated keys in the path.

Referring to Figure G-1:

- The path to C-5 is A-001, B-01.
- The path to C-7 is the same as the path to C-5.
- There is no path to A-002 because it is a root segment.

path call. (1) The retrieval or insertion of multiple segments in a hierarchical path in a single call, by using the D command code and multiple SSAs. (2) The retrieval, replacement, or insertion of data for multiple segments in a hierarchical path in a single command, by using the FROM or INTO options specifying an I/O area for each parent segment desired. The object segment is always retrieved, replaced, or inserted.

PCB. Program communication block.

PCB mask. A skeleton data base PCB in the application program by which the program views a hierarchical structure and into which DL/I returns the results of the application's calls.

physical child. A segment type that is dependent on a segment type defined at the next higher level in the data base hierarchy. All segment types, except the root segment, are physical children because each is dependent on at least the root segment. See also *child segment.*

physical data base. An ordered set of physical data base records.

physical data base record. A physical set of hierarchically related segments of one or more segment types.

physical data structure. A hierarchy representing the arrangement of segment types in a physical data base.

physical parent. A segment that has a dependent segment type at the next lower level in the physical data base hierarchy. See also *parent.*

physical segment. The smallest unit of accessible data.

physical twins. All occurrences of a single physical child segment type that have the same (single occurrence) physical parent segment type. Contrast with *logical twins.* See also *twin segment.*

PI. Program isolation

pointer. A physical or symbolic identifier of a unique target.

position pointer. For most call functions, a position pointer exists before, during, and after the completion of the function. The pointer indicates the next segment in the data base that can be retrieved sequentially. It is normally set by the successful completion of the call function. Referring to Figure G-1:

- If A-001 has just been retrieved, it points to B-01.
- If a new segment C-6 has just been inserted, it points to C-7.
- If the D segment has been deleted (E will be deleted along with it), it points to the first F segment.
- If the last F segment has just been retrieved, it points to A-002.

During PSB generation, it is possible to specify either single or multiple positioning.

primary key. The data element, or combination of data elements, within a segment that uniquely identifies an occurrence of that segment. See *key* and *secondary key.*

program communication block (PCB). Every data base accessed in an application program has a PCB associated with it. The PCB actually exists in DL/I and its fields are accessed by the application program by defining their names within the application program as follows:

COBOL	The PCB names are defined in the linkage section.
PL/I	The PCB names are defined under a pointer variable.
Assembler	The PCB names are defined in a DSECT.
RPG II	The PCB names are automatically generated by the translator, or may be defined by the user.

There are nine fields in a PCB:

1. Data base name

2. Segment hierarchy level indicator
3. DL/I results status code
4. DL/I processing options
5. Reserved for DL/I
6. Segment name feedback area
7. Length of key feedback area
8. Number of sensitive segments
9. Key feedback area.

Program Isolation (PI). A facility that isolates all data base activity of an application program from all other application programs active in the system until that application program commits, by reaching a synchronization point, that the data it has modified or created is valid.

This concept makes it possible to dynamically backout the data base activities of an application program that terminates abnormally without affecting the integrity of the data bases controlled by DL/I. It does not affect the activity performed by other application programs processing concurrently in the system.

program specification block (PSB). A PSB is generated for each application program that uses DL/I facilities. The PSB is associated with the application program for which it was generated and contains a PCB for each data base that is to be accessed by the program. Once it is generated, the PSB is cataloged in a core image library, and subsequently processed by a utility along with the associated DBDs to produce the updated PSB and DMBs; all of these are cataloged in a core image library for subsequent use by the application program during execution.

PSB. Program specification block

PSBGEN. PSB generation -- the process by which a program specification block is created.

qualified call. A DL/I call that contains at least one segment search argument (SSA). See also *segment search argument.*

qualified segment selection. The identification of a specific occurrence of a given segment type in a command, by using the WHERE option in the command for the desired segment. Contrast with *qualified SSA.*

qualified SSA. A qualified segment search argument contains both a segment name that identifies the specific segment type, and segment qualification that identifies the unique segment within the type for which the call function is to be performed. See also *segment search argument* and *multiple SSA.*

RAP. Root anchor point.

RBA. Relative byte address.

read-only intent. The scheduling intent type that allows a program to be scheduled with any number of other programs except those with exclusive intent. No updating occurs. See *scheduling intent.*

record. A data base record is made up of at least a unique root segment, and all of its dependent segments. See *data base record.*

relative byte address (RBA). The displacement of a stored record or control interval from the beginning of the storage space allocated to the VSAM data set to which it belongs.

remote system. In a multisystem environment, the system containing the data base that is being used by an application program resident on another (local) system Contrast with *local system.*

root anchor point (RAP). A DL/I pointer in an HDAM control interval that points to a root segment or a chain of root segments.

root segment. The highest level (level 1) segment in a record. A root segment must have a key unless the organization is HSAM or simple HSAM. The sequence of the root segments constitutes the fundamental sequence of the data base. There can be only one root segment per record. Dependent segments cannot exist without a parent root segment but a root segment can exist without any dependent segments.

RQDLI COMMAND. The instruction in the RPG II program used to request DL/I services.

scheduling intent. An application program attribute defined in the PSB that specifies how the program should be scheduled if multiple programs are contending for scheduling. See *exclusive intent, read-only intent,* and *update intent.*

search field. In a given DL/I call, a field that is referred to by one or more segment search arguments (SSAs).

secondary index. Secondary indexes can be used to establish alternate entries to physical or logical data bases for application programs. They can also be processed as data bases themselves. See also *secondary index data base.*

secondary index data base. An index used to establish accessibility to a physical or logical data base by a path different from the one provided by the data base definition. It contains index pointer segments.

secondary key. A data element, or combination of data elements, within a segment that identifies -- and is used to locate -- those occurrences of the segment that have a property named by the key. See *key* and *primary key.*

segment. A segment is a group of similar or related data that can be accessed by the application program with one I/O function call. There may be a number of segments of the same type within a record.

segment name. A segment name is assigned to each segment type. Segment names for the different segment types must be unique within a data base. The segment name is used by the application programmer when constructing a qualified or unqualified SSA prior to issuing a call for a specific segment. Synonymous with *segment type.*

segment occurrence. One instance of a set of similar segments.

segment search argument (SSA). Describes the segment type, or specific segment within a segment type, that is to be operated on by a DL/I call. See also *multiple SSA, qualified SSA,* and *unqualified SSA.*

segment selection. The specifying of parent and object segments by name in a command. Selection may be either qualified or unqualified. Contrast with *segment search argument.*

segment type. A user-defined category of data. Referring to Figure G-1, there are six different types of segments; A through F.

Different segment types may have different lengths, but within each single type, all segments must be the same length (unless variable length segments have been specified by the DBA). Synonymous with *segment name.*

sensitivity. (1) A DL/I capability that ensures that only data segments or fields predefined as C<DC<Dsensitive" are available for use by a particular application program. The sensitivity concept also provides a degree of control over data security, inasmuch as users can be prevented from accessing particular segments or fields from a logical data base. (2) Sensitivity to the various segments and fields that constitute a data base is controlled, on a program-by-program basis, when the PSB for each program is generated. For example, a program is said to be sensitive to a segment type when it can access that segment type. When a program is not sensitive to a particular segment type, it appears to the program as if that segment type does not exist at all in the data base. Segment sensitivity applies to types of segments, not to specific segments within a type, and to all segment types in the path to the lowest level sensitive segment type.

sequence field. Synonymous with *Key field.*

sequence set. The lowest level of a VSAM index. It immediately controls the order of records in a key sequenced data set (KSDS). A sequence set entry contains the key of the highest keyed record stored in a control interval of the data set and a pointer to the control interval's physical location. A sequence set record also contains a pointer to the physical location of each free control interval in the fan-out of the record.

sequential processing. Processing or searching through the segments in a data base in a forward direction (see also *forward*).

simple HISAM. A hierarchical indexed sequential access method data base containing only one segment type.

source segment. A segment containing the data used to construct the secondary index pointer segment. See also *secondary index data base.*

SSA. Segment Search Argument

status code. Each DL/I request for service returns a status code that reflects the exact results of the operation. The first operation that a program should perform immediately following a DL/I request is to test the status code to ensure that the function requested was successful. Following a command, the status code is returned in the DIB at the label DIBSTAT. Following a call, the status code is returned in field 3 of the PCB.

sync(h) point. Synonymous with *synchronization point.*

synchronization point. A logical point in time during the execution of an application program where the changes made to the data bases by the program are committed and will not be backed out. Synonymous with *sync point* or synch point.

A synchronization point is created by:

- a DL/I CHECKPOINT command or CHKP call
- a DL/I TERMINATE command or TERM call
- a CICS/VS synch point request
- an end of task (online) or an end of program (MPS-batch).

system view. A conceptual data structure that integrates the individual data structures associated with local views into an optimum arrangement for physical implimentation as a data base. See *local view.*

transaction. A specific set of input data that triggers the execution of a specific process or job.

twin segments. All child segments of the same segment type that have a particular instance of the same parent type. See also *physical twins* and *logical twins.*

twins. Synonymous with *twin segments.*

unqualified call. A DL/I call that does not contain a segment search argument.

unqualified segment selection. The identification of a given segment type in a command without specifying a particular occurrence of that segment type (without using the WHERE option). As a general rule, unqualified segment selection retrieves the first occurrence of the specified segment type. Contrast with *unqualified SSA.*

unqualified SSA. An unqualified SSA contains only a segment name that identifies the specific type of segment for which the I/O function is to be performed. As a general rule, the use of an unqualified SSA retrieves the first occurrence of the specified type of segment. See also *segment search argument.*

update intent. The scheduling intent type that permits application programs to be scheduled with any number of other programs except those with exclusive intent. See *scheduling intent.*

Communicating Your Comments to IBM

IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)
Application Programming:
CALL and RQDLI Interfaces
Version 1 Release 7

Publication No. SH12-5411-06

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of the book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF form and either send it postage-paid in the United States, or directly to:

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

- If you prefer to send comments by FAX, use this number:
  - (Germany): 07031-16-3456
  - (Other countries): (+49)+7031-16-3456

- If you prefer to send comments electronically, use this network ID:

  INTERNET: s390id@de.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

**IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)
Application Programming:
CALL and RQDLI Interfaces
Version 1 Release 7**

Publication No. SH12-5411-06

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH12-5411-06

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

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

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/S390-50
Program Number: 5746-XX1

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH12-5411-06