SH20-9046-3
File No. S370/4300-50

IBM System/370 Low-Level Code/ Continuity Check in Data Language/I DOS/VS

Program Reference and Operations Manual

Program Product

Program Number 5746-XX1

IBM

Fourth Edition (December 1983)

This edition, SH24-9046-3, 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 versions and modifications until otherwise indicated in new editions or Technical Newsletters. It supersedes SH20-9046-2 and Technical Newsletter SN24-5681. Changes are continually made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, consult the latest edition of *IBM System/370 and 4300 Processors Bibliography*, GC20-0001, for editions that are applicable and current.

Summary of Changes

For a list of changes, see page iii.

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

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

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

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

IBM Corporation
Dept. 812BP
1133 Westchester Avenue
White Plains, NY, 10604 U.S.A.

or

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Summary of Changes

Summary of Changes
for SH20-9046-3

This revision contains various performance improvements such as information on defining VSAM control intervals and DL/I buffers through use of the BLOCK parameter in the DATASET statement, and the HDBFR parameter in the DL/I statement. This edition also contains miscellaneous additions, corrections, and improvements.

Summary of Changes
for SH20-9046-2
as updated by SN24-5681

This Technical Newsletter updates titles and/or order numbers for DL/I DOS/VS and other publications referenced in this book.

Summary of Changes
for SH20-9046-2

This major revision contains further additions, corrections, and improvements, such as notes regarding loss of the application program's position in the data base through use of the same PCB by LLC/CC in DL/I, organization of the parts data base in either HIDAM or HDAM, use of the RULES parameter in the SEGM statement, and the replacement of job input stream examples.

Summary of Changes
for SH20-9046-1

This edition was a major revision of SH20-9046-0, and included Technical Newsletter SN20-5051. This revision contained miscellaneous additions, corrections and improvements.

Preface

This manual is intended for application programmers who want to use the services of Low-Level Code/Continuity Check (LLC/CC) in Data Language/I Disk Operation System/Virtual Storage (DL/I DOS/VS). It describes the functions and the operation of the system, and contains all the information required to generate and execute LLC/CC in DL/I DOS/VS.

Low-level codes are used primarily in the manufacturing industry to indicate the lowest level at which a particular part number is found in all product structure trees. The product structures must not contain any loops. Therefore, a continuity check is applied to ensure proper assembly-to-subassembly continuity.

Low-Level Code/Continuity Check in DL/I DOS/VS provides a callable subroutine that assigns low-level codes to parts in a manufacturing industry parts data base. LLC/CC is used primarily by programs that maintain bills of material.

This manual has three basic parts. The first part contains an "Introduction" and a "General Description" where the problem areas are defined and the solutions LLC/CC offers are presented.

The second part, containing chapters on "Data Base Description," "Invocation of LLC/CC IN DL/I," "Operational Procedures," and "Error Messages and Return Codes," describes the program structure and the functions of LLC/CC in DL/I DOS/VS, and its data base.

The third part contains "Installation Requirements" and the LLC/CC system requirements.

Related Publications

The reader is assumed to have a working knowledge of the functions and the facilities of DL/I DOS/VS and should be familiar with the contents of the following publications:

- *DL/I DOS/VS Application and Data Base Design*, SH24-5022.
- *DL/I DOS/VS Resource Definition and Utilities*, SH24-5021.
- *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*, SH12-5411.
- *DL/I DOS/VS Messages and Codes*, SH12-5414.
- *DL/I DOS/VS Recovery/Restart Guide*, SH24-5030.

Contents

Figures

Chapter 1. Introduction

This manual describes the functions and facilities of Low-Level Code/Continuity Check in Data Language/I DOS/VS which will be abbreviated by LLC/CC in DL/I.

This manual is designed to meet the requirements of the application programmer. It begins with descriptions of the capabilities of the program, or processing aspects, and of the required data bases. It presents information on how to use the services of LLC/CC in DL/I, how to generate and operate the programs, and how to handle error messages and return codes. The manual concludes with a series of sections dealing with performance specifications, programming systems, system requirements, etc.

Low-level coding and continuity checking are established techniques for production planning and control in the manufacturing industry.

Low-level coding is generally used to facilitate the planning of material requirements, from end items down to raw materials and purchased parts. Continuity checking assures the continuity of the related bills of material. In other words, continuity checking prevents a part from being accidentally contained in itself. If continuity is violated, a loop is produced, and low-level codes cannot be assigned. A peculiarity of common low-level coding is the reverse order of the numeric value. While an end item is logically at the highest level in the hierarchy, its low-level code is zero. The lower the level within the hierarchy of a bill of material, the higher is the numeric value of the code.

LLC/CC in DL/I can also be applied to problems outside the manufacturing industry. All problems that can be expressed as directed graphs or networks may be defined so that they are suitable for processing by LLC/CC in DL/I. In this case, the low-level code represents the relative distance between a particular node in a network and one or more end nodes. Nodes in a network are linked by edges. The relative distance is expressed by the number of edges on the longest direct path from any one node to any one end node.

LLC/CC in DL/I is link-edited to a user-written application program. The application program either initially generates or updates low-level codes in a manufacturing data base organized by Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS). LLC/CC in DL/I interfaces with the DL/I language (see Figure 1 on page 2). It is a called subroutine which is link-edited with the application program. It supports application programs written in Assembler language, COBOL, or PL/I. Different entry points make function selection and language adjustments possible.

LLC/CC in DL/I is compatible with IBM System/370 Chained File - DL/I Bridge, Program Product 5748-XX3. Both LLC/CC in DL/I and Chained File - DL/I Bridge support the same data base structure; thus, an installation may use both programs concurrently.

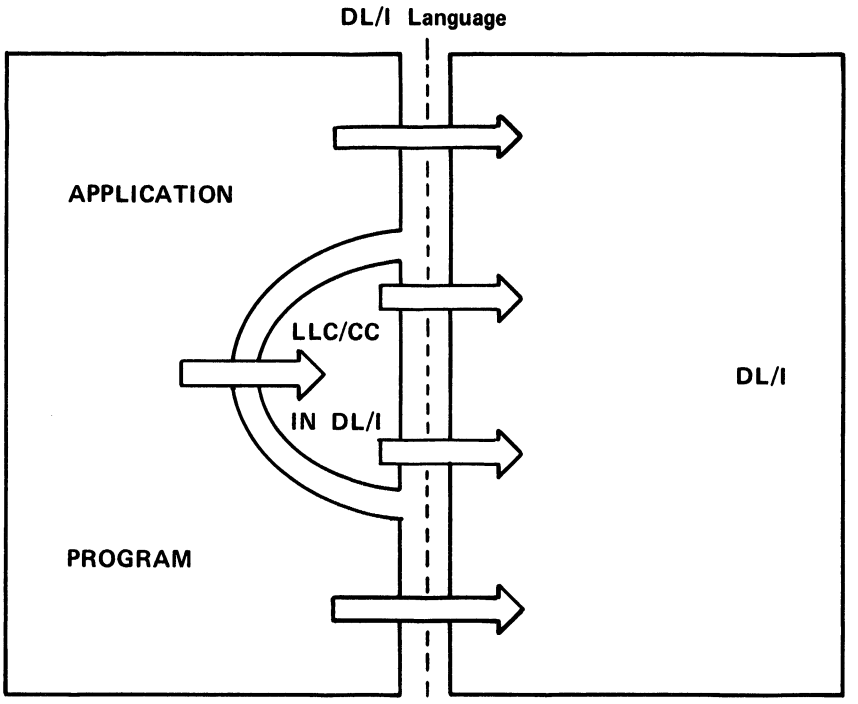


Figure 1. General System Overview

Chapter 2. General Description

Purposes and Objectives

LLC/CC in DL/I is designed to increase the general usability of the data base facilities of Data Language/I DOS/VS (DL/I DOS/VS). It offers all functions required to initially generate and to update low-level codes for manufacturing product structures. Low-level codes are used in production planning mainly for material requirements planning and for retrieval of bill of material and "where-used" data.

A prerequisite for this type of planning is reliable checking of assembly-to-subassembly continuity. Improper sequence may cause production planning programs to loop, for instance, when a part is contained - either directly or indirectly - in itself. It is impossible to assign a unique low-level code to parts within such a loop. If continuity checking detects an error, the request to assign low-level codes to a particular part and its components is rejected, and the previous state is reestablished. The calling application program is notified of the loop and of the part which caused the loop.

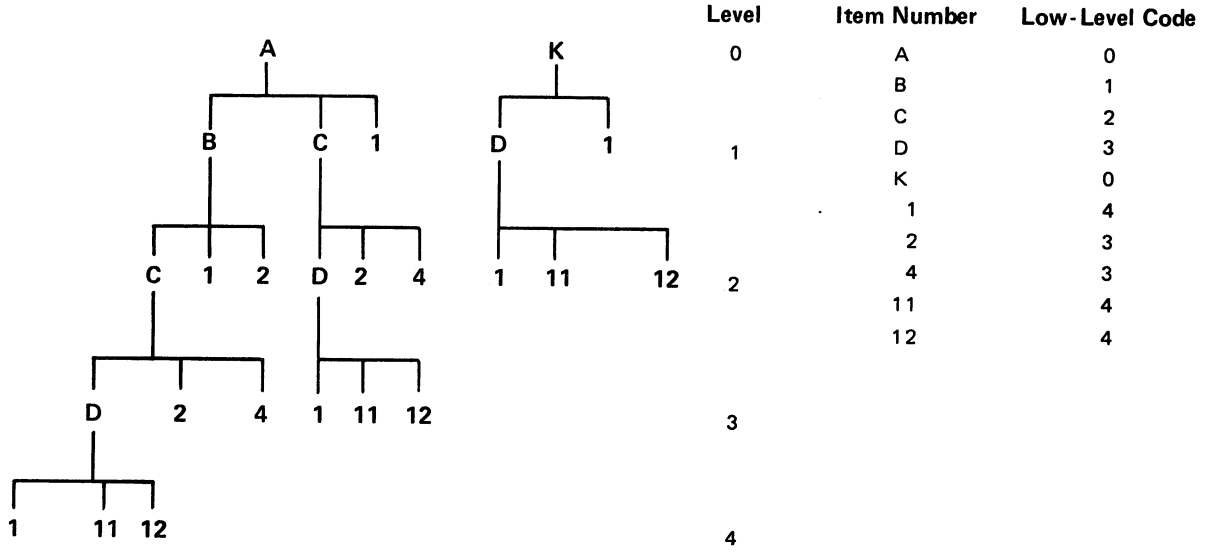
LLC/CC in DL/I is a callable subroutine, that is, it becomes a subroutine of a user-written application program. The application program must identify the product structure to be processed. During the initial generation, it is assumed that all product structure relationships that are present in the data base must be processed. During an update, it is assumed that a new product structure, that is, a line of a new bill of material, is added after LLC/CC in DL/I has updated low-level codes of affected components.

However, LLC/CC in DL/I is also an extremely valuable tool for many other applications. For instance, it may be used to resolve certain network problems if the network reflects the structure of a unidirectional graph.

Extent of Coverage

A manufacturing product structure may be considered as a tree structure (see Figure 2 on page 4). At the top of the diagram, there is the end item. Within such a manufacturing structure, no further processing takes place on the end item. End items may be exploded into components, into assemblies, subassemblies, parts, and finally into raw materials and purchased parts. An end item is defined as an item or part which is not a component of any other item or part, for instance, items A and K. A raw material or purchased part is defined as an item without any components, for instance, items 1, 2, 4, 11, and 12. An item may be an end item and a raw material, or a purchased part, simultaneously. An end item has a low-level code of zero. All component parts have a non-zero low-level code. If an item is only a component of an end item, it has a low-level code of 1, for instance, item B. The components of item B have a low-level code of 2, etc.

If an item is a component of more than one item, that is, of more than one parent item, the lowest level is assigned. A parent item may also be a component item of another parent item. For instance, item C is a component of parent item A with low-level code 0, and of parent item B with low-level code 1. As a consequence, item C has low-level code 2.



Level	Item Number	Low-Level Code
0	A	0
	B	1
	C	2
1	D	3
	K	0
	1	4
	2	3
	4	3
2	11	4
	12	4
3		
4		

Figure 2. Product Structure and Low-Level Code

If a new product structure is added, a new relationship between a parent part and a component part is established. While the actual low-level code of the parent part remains unchanged, the component part may be affected. According to the rules of low-level coding, its low-level code must be numerically higher than that of the parent part. Therefore, the component part is checked to determine whether it meets this condition. If so, no further updating of low-level codes is required. Otherwise, the low-level code of the component part is updated, and a recursive process is set up to check the component parts of the actual component. This process is discontinued either if a component part already has a low-level code which needs no updating, or if a part has no further components, that is, if the part is a purchased part or a raw material.

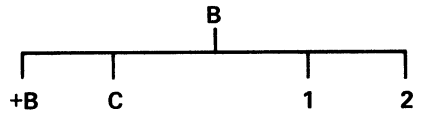
LLC/CC in DL/I does not support the updating of low-level codes in case of deletions of product structure relationships. The use of low-level codes is not affected if they are lower in the scale, that is, greater in numeric value, than the actual usages in the product structure trees. Processing of deletions would require extensive processing of where-used relationships which would be likely to decrease performance.

Continuity checking is a prerequisite for the effective maintenance of manufacturing product structures. Proper assignment of low-level codes becomes impossible if an item is contained in itself either directly or indirectly (see Figure 3 on page 5). Hence LLC/CC includes a continuity checking facility.

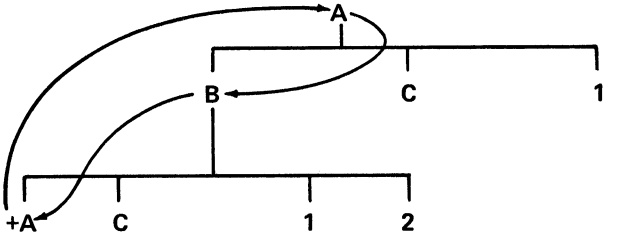
If a part is contained in itself directly, as part B of Figure 3 on page 5 , the application programmer may easily detect and intercept the violation. However, if a parent part is contained in itself indirectly, as part A of Figure 3 on page 5, the violation is no longer easily detectable. In both cases, the continuity checking facility of LLC/CC issues a return code to the invoking application program so that the erroneous product structure relationship may not be added to, or removed from, the data base.

Continuity checking also detects whether a component is contained in itself. This situation may be encountered during initial generation of low-level codes for an existing product structure data base. Parts may be processed sequentially in

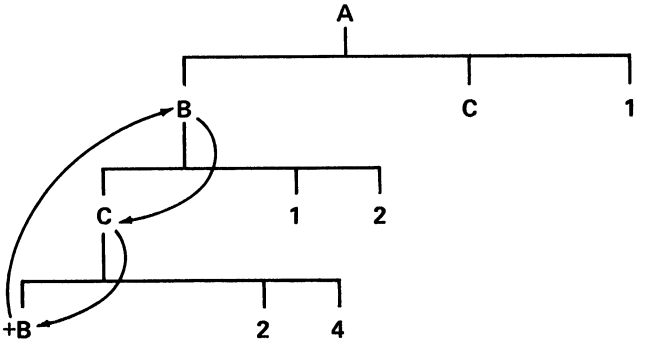
ascending order. In Figure 3, part A becomes the first part to be exploded; thus, part B has still to be checked. To avoid an endless loop caused by component part B, LLC/CC checks for continuity of all parts within each path of an explosion. This facility is valuable during the updating of low-level codes if the initial generation has not been executed properly. LLC/CC issues a return code and identifies the part which is in error so that the data base can be corrected.



Parent part B is contained in itself directly



Parent part A is contained in itself indirectly



Component part B is contained in itself indirectly

Figure 3. Continuity Checking

In a Gozinto graph (derived from "the part that goes into") in Figure 4 on page 6, the product structure is represented by a network which can be described as a directed graph. The nodes in this network are items, the edges are the intersection data between two adjacent nodes, that is, the product structure information. The graph may have multiple start nodes, (that is, raw materials or purchased parts) and multiple end nodes (that is, end items). A graph may only consist of a single node. The low-level code for a particular node is defined by the largest number of edges in a direct path from this node to an end node. End nodes, therefore, have low-level code zero.

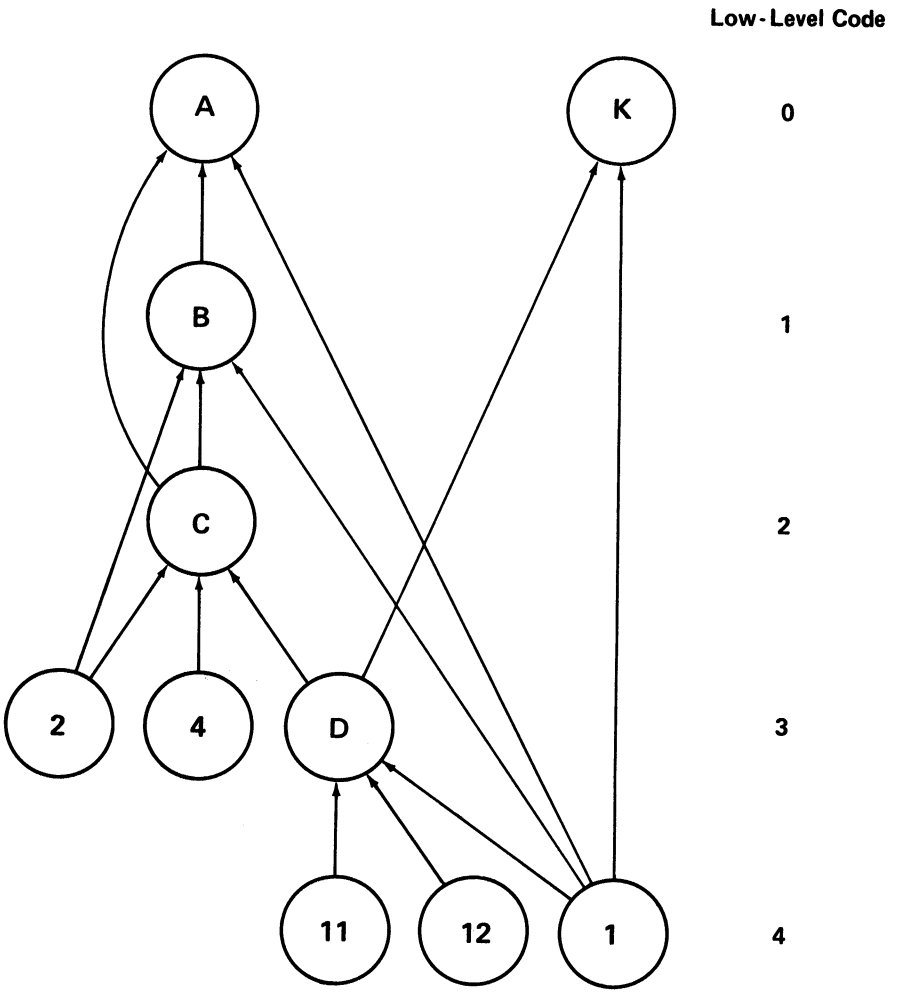


Figure 4. Gozinto Graph

The Gozinto graph concept allows the general application of LLC/CC to network analysis problems, for instance, project control or scheduling.

Product structures are normally created by a list of components for a particular item. This list is a single-level bill of material. Each line of the bill of material represents a parent-to-component relationship, that is, a product structure relationship. Low-level codes may be assigned either once after all product structures have been loaded into the data base, or consecutively each time a new line of a new bill of material, that is, a new product structure, is added to the data base. LLC/CC supports both processing modes. In the initial-generation mode, existing product structures are processed. For a particular part, all components and subcomponents are checked, and low-level codes are assigned. In the updating mode, a single parent-to-component relationship is processed. Only the particular component and its subcomponents are checked for the assignment of proper low-level codes.

LLC/CC does not perform the insertion of segments to realize a product structure relationship within a data base. The user must provide an application program to call LLC/CC. The application program identifies the keys of the items for

processing, interprets return codes from LLC/CC, and adds new bills of material in the updating mode if LLC/CC has signaled the proper completion of low-level coding.

Processing Description

LLC/CC is executed in two different modes, namely, initial-generation mode and updating mode.

The initial-generation mode is used to insert low-level codes into a parts data base where product structure relationships have already been established. It is assumed that the application program processes all root segments, that is, all parts, in an arbitrarily selected sequence. LLC/CC is invoked by a CALL statement indicating the particular part key. No processing is required if either:

- The part has a low-level code higher than zero, that is, if this part has already been processed as the component of another part.
- The part has no component parts.

If processing is required, the part is exploded into its components. If the low-level code of the component is higher than that of the parent, no low-level code updating is required. This also excludes a loop condition, and no continuity check is required. Otherwise, the numeric value of the low-level code of the parent part is incremented by one and stored in the component. If the component part has subcomponents, this process is repeated recursively. The original status of all low-level codes which have been changed is saved in the control data base, thus allowing the restoration of all low-level codes in their unchanged state if a part is detected by continuity checking to be contained in itself.

Upon completion of processing, control is returned to the calling application program. A return code is passed back, indicating either successful completion or an abnormal condition. An abnormal condition exists, if a part is missing, if a part is contained in itself, or if an unexpected DL/I condition has developed. If the operation is completed successfully, all dependent low-level codes are updated, and the user program identifies the next part for low-level code assignment. It should also be noted that LLC/CC has used the application program's PCB and, in doing so, has changed the position in the data base. If the original position is required for further processing, it must be re-established by another DL/I call.

Updating of low-level codes is performed in basically the same way. In the updating mode, however, the application program provides two part keys, namely, the parent key and the component key. A component part is exploded in the same way as in initial-generation mode. However, if the component already has a low-level code which is higher than the low-level code of the parent part, no further processing is required. To further minimize redundant processing, it is assumed that the product structures being updated in the update mode are continuity check error free.

LLC/CC employs a special technique of exploding to extend continuity checking to component parts. Explosion follows hierarchical paths. A hierarchical path is defined by the path from the parent part to a particular purchased part, or raw material, within a product structure. In Figure 2 on page 4, 13 different hierarchical paths are available for parent part A, for example:

A/B/C/D/1	A/C/D/1
A/B/C/D/11	A/C/D/11
A/B/C/D/12	A/C/D/12
A/B/C/2	A/C/2
A/B/C/4	A/C/4
A/B/1	A/1
A/B/2	

Following a hierarchical path is discontinued if a component needs no further explosion because its low-level code is already correct.

The relative position of a component in a hierarchical path also indicates the numeric value of the low-level code relative to the parent part.

The explosion sequence in a product structure tree is from top to bottom and from left to right. When a component is exploded, all its components which require low-level code updating are entered into a list. Each entry in this list is represented by an occurrence of the segment PARTBEXP of the control data base. The segment contains only an identifier which is composed of the low-level code and the part key of the component. If the explosion of a particular part is completed, all components are processed, and the low-level code is incremented by 1. Subsequently, the first occurrence of a part key headed by the new low-level code is read from PARTBEXP.

When passing through the various levels of a product structure from top to bottom, all parts making up a hierarchical path are entered into segment PARTBEXP of the control data base.

In a particular path, no part must occur twice. For instance, if DL/I rejects an insertion request for a part, a loop is detected. To differentiate between this type of entry into segment PARTBEXP and other entries, the part key is headed by a 2-byte field containing hexadecimal zeros.

Processing of a hierarchical path is terminated if either:

- a component has no more components
- the components of the component have already low-level codes which need no updating, that is, the numeric value is higher than the low-level code of the component previously exploded.

When processing of one path is terminated, the next parallel component is selected for processing. The previously exploded component is removed from both the continuity check information and the explosion control information in segment type PARTBEXP. The next explosion control entry is read for the same level. For instance, according to the structure depicted in Figure 2 on page 4, the path A/B/C/D/1 is replaced by the path A/B/C/D/11. If a particular level is completely exhausted, the next higher level is processed, that is, the numeric value is decremented by 1. For instance, in Figure 2 on page 4 the path A/B/C/D/12 is replaced by the path A/B/C/2.

The operation is terminated as soon as all hierarchical paths originating from the highest part in the tree structure, for example, from A, have been processed.

Since the initial value of all low-level codes modified during execution of LLC/CC is saved in segment type UPDMASTR of the control data base, the original status is restored immediately after a loop has been detected.

On return to the calling application program, the entries in the control data base are definitely deleted by deletion of the root segment LLCTL in the control data base.

Figure 5 gives an overview of how part A of Figure 2 on page 4 is exploded. This explosion sequence reflects the initial generation of low-level codes for A. In updating mode, the same sequence applies if A is identified as a component part (parm4) of a new product structure.

Actual			Remarks	Changes in PARTBEXP		Actual Hierarchial Path
Parent Part (LLC/Key)	Low-Level Code	Component Part (LLC/Key)		Explosion Ctrl. Entries (LLC/Key)	Continuity Check Entries (Key only)	
	0			+ 0/A		
0/A	1	1/B		+ 1/B	+ A	A
0/A	1	1/C		+ 1/C		
0/A	1	1/1		+ 1/1		
	1		Change to next level			
1/B	2	2/C		+ 2/C	+ B	A/B
1/B	2	2/1		+ 2/1		
1/B	2	2/2		+ 2/2		
	2		Change to next level			
2/C	3	3/D		+ 3/D	+ C	A/B/C
2/C	3	3/2		+ 3/2		
2/C	3	3/4		+ 3/4		
	3		Change to next level			
3/D	4	4/1		+ 4/1	+ D	A/B/C/D
3/D	4	4/11		+ 4/11		
3/D	4	4/12		+ 4/12		
	4		Change to next level			
4/1	5	–	No components		+ 1	A/B/C/D/1
	4		Next on same level	– 4/1	– 1	A/B/C/D
4/11	5	–	No components		+ 11	A/B/C/D/11
	4		Next on same level	– 4/11	– 11	A/B/C/D
4/12	5	–	No components		+ 12	A/B/C/D/12
–	4		Next on same level	– 4/12	– 12	A/B/C/D
–	3		Same level exhausted	– 3/D	– D	A/B/C
3/2	4	–	No components		+ 2	A/B/C/2
	3		Next on same level	– 3/2	– 2	A/B/C
3/4	4	–	No components		+ 4	A/B/C/4
	3		Next on same level	– 3/4	– 4	A/B/C
–	2		Same level exhausted	– 2/C	– C	A/B
2/1	3	–	No components		+ 1	A/B/1
	2		Next on same level	– 2/1	– 1	A/B
2/2	3	–	No components		+ 2	A/B/2
	2		Next on same level	– 2/2	– 2	A/B
–	1		Same level exhausted	– 1/B	– B	A
1/C	2	2/D	Comp. LLC not low	–	+ C	A/C
1/C	2	2/2	Comp. LLC not low	–		
1/C	2	2/4	Comp. LLC not low	–		
	3		Change to next level			
–	2	–	Same level exhausted			
	1		Next on same level	– 1/C	– C	A
1/1	2	–	No components		+ 1	A/1
	1		Next on same level	– 1/1	– 1	A
–	0	–	Same level exhausted	– 0/A	– A	
Terminate	0		Initial level reached			

Figure 5. Explosion Sequence for Part A

Advantages

LLC/CC incorporates a key function of the bill processor systems, for instance, IBM System/360 Data Base Organization and Maintenance Processor (DBOMP), Program Product 5736-XX4. This provides many advantages for the users of DL/I DOS/VS:

- Low-level codes facilitate summarized explosions and implosions of parts in a manufacturing data base and also in material requirements planning. Low-level codes may range from 0 to 999.
- Continuity checking detects violations of continuity rules for manufacturing product structures. It signals loops consisting of parts which are contained in themselves either directly or indirectly.
- The user need not code routines to assign low-level codes.
- Performance of net change operations is improved if low-level codes are available.
- Changing from DBOMP to DL/I becomes easier.
- The low-level code approach also provides valuable tools to resolve network problems in many other applications.
- LLC/CC is a called subroutine which may be invoked by user-written application programs of any structure.
- LLC/CC does not interfere with external input/output operations of a calling application program.
- LLC/CC supports application programs written in Assembler, COBOL, and PL/I.

Utilization of low-level codes may not be essential for applications in the manufacturing industry. However, experience has shown this method to be an extremely useful technique to resolve efficiently production planning problems.

Relationships Between LLC/CC in DL/I and Chained File - DL/I Bridge

LLC/CC in DL/I supplements the facilities of the IBM System/370 Chained File - DL/I Bridge, Program Product 5748-XX3. In conjunction with DL/I Bridge, LLC/CC in DL/I ensures a smooth change-over to DL/I DOS/VS for current users of the IBM System/360 Data Base Organization and Maintenance Processor (DBOMP), Program Product 5736-XX4. While users may continue to execute some DBOMP programs against a DL/I data base via the DL/I Bridge, they may develop more and more DL/I programs.

LLC/CC in DL/I is fully compatible with DL/I Bridge:

- Low-level codes have the same format, and the same rules for continuity checking apply.
- Both LLC/CC in DL/I and DL/I Bridge assume the same basic structure of the parts data base.
- The control data base is compatible.

Hence, for the requirements of an application programmer, both programs are compatible. During the transition period from DBOMP to DL/I DOS/VS, both programs may be executed concurrently without any impact on the data bases. For details about DL/I Bridge, see *IBM System/370 Chained File - DL/I Bridge (DOS/VS and OS/VS), General Information Manual*, GH12-5116.

Chapter 3. Data Base Description

The user of LLC/CC must provide two different data bases. The parts data base contains all manufacturing information which is related to parts and to product structures. The control data base is an empty data base which is only used by LLC/CC during execution; it does not contain any permanent information.

Parts Data Base

The parts data base is of key importance within a manufacturing environment. It is the vehicle to maintain the majority of item-related data, for instance, identification, description, cost data, forecast and demand data, inventory data, information about planned and open orders. Furthermore, certain internal and external relationships reflect the flow of production through the manufacturing environment.

Each individual part in a product structure - whether it occurs once or more than once -, or each node in a Gozinto graph, is reflected by the single occurrence of a logical data base record in DL/I. Each logical data base record consists of a root segment and a variable number of dependent segments. To establish the product structure relationships between a parent part and its component parts, or to reflect the edges connecting nodes within a Gozinto graph, the concept of logical relationships within DL/I is employed. Thereby, the root segments of all component parts directly or indirectly become logical dependent segments of the root segment of the parent part. The logical relationship is created using a pointer segment. It provides bidirectional pointers to establish both the bill-of-material relationships and the where-used relationships of a particular part. Using additional pointer segments, a part may be linked with related segments of other data bases, for instance, work centers. Figure 6 shows an example of a logical parts data base. A vertical arrow in a segment box indicates that this particular segment is linked to its parent by a pointer segment.

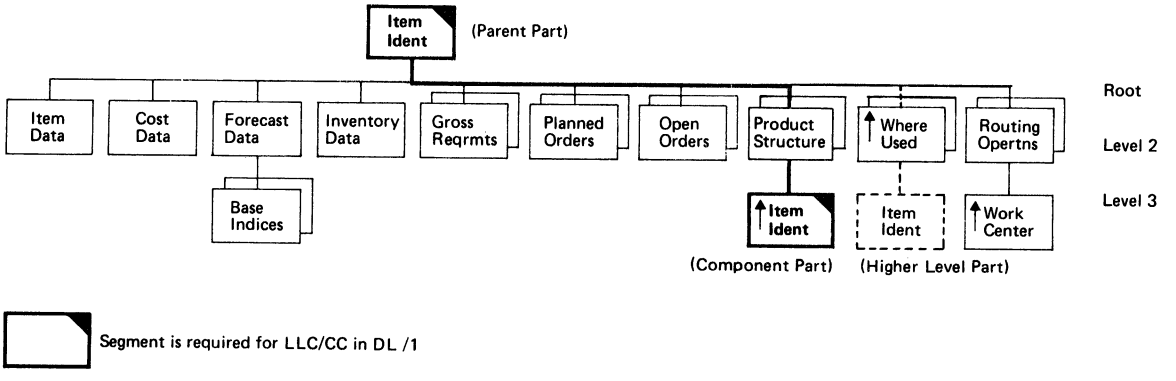


Figure 6. Sample Parts Data Base

If intersection data is to be stored, an additional segment should be introduced rather than storing this data in the data portion of a pointer segment. Intersection data is all the information which is unique to a particular parent part-to-component part relationship, or to a particular edge connecting two nodes in a Gozinto graph. For instance, intersection data is the quantity of a component part which is required to make a parent part. This type of data should be stored in a separate product structure segment which should be the physical parent segment of the

pointer segment. If where-used relationships are implemented, the virtual where-used segment of the component part should be paired with the pointer segment of the parent part.

If a user wishes to store the intersection data in the pointer segment, he must consider the following restrictions:

- Data without pertinent pointers is nonexistent for the system. This implies that new product structure data can only be added after all pertinent component part root segments have been successfully inserted. Also, when replacing one component part by another, the intersection data must be saved immediately.

- A pointer segment only provides a single logical relationship. If the product structure information is stored in the pointer segment, no other linkages to related segments can be established (for instance, logical relationships with routing data, tool data, or work center data).

- The data base compatibility with the IBM System/370 Chained File - DL/I Bridge, Program Product 5748-XX3, is lost. DL/I Bridge assumes a pointer segment without data stored in it.

LLC/CC does not require a fixed structure of the parts data base, that is, the user is free to arrange data and segments according to need. However, certain rules apply for the location of low-level code fields and for establishing logical relationships:

- The parts data base must have a minimum structure as depicted in Figure 7 on page 14, consisting of an item root segment and a dependent pointer segment, which must be a logical child of the root segment of the component part.

- The pointer segment may be located on any hierarchical level below the root segment of the parent part.

- Intersection data should not be stored in a pointer segment unless the restrictions governing its use, described above, are acceptable to the user. The pointer segment should be a physical child of the segment which contains the intersection data.

- The low-level code field is located in the root segment and is a 2-byte packed decimal number. Prior to initial generation of low-level codes, the field must contain packed zeros.

- The parts data base is organized according to the hierarchical indexed direct access method (HIDAM) or the hierarchical direct access method (HDAM).

When discussing the parts data base, it is necessary to differentiate carefully between physical data base and logical data base. Throughout this manual, the following segment names are used:

Segment	Level of Dependency	Physical Parts DB	Logical Parts DB
Root	**1**	**DLZLLSPN**	**DLZLLSPL**
Pointer	**2-15**	**DLZLLSLK**	**-**
Pointer + Component Root	**2-15**	**-**	**DLZLLLDD**

The segment names of the logical parts data base are also used as default names when customizing the source code of LLC/CC. (Refer to "Generation of the

Execution Program (Macro DLZNN)" on page 28 for further information.) However, the user may change the names because of special requirements. If the user-written application program calling the services of LLC/CC is sensitive to additional segment types (as defined by the SENSEG statements of the PSB), LLC/CC is not affected.

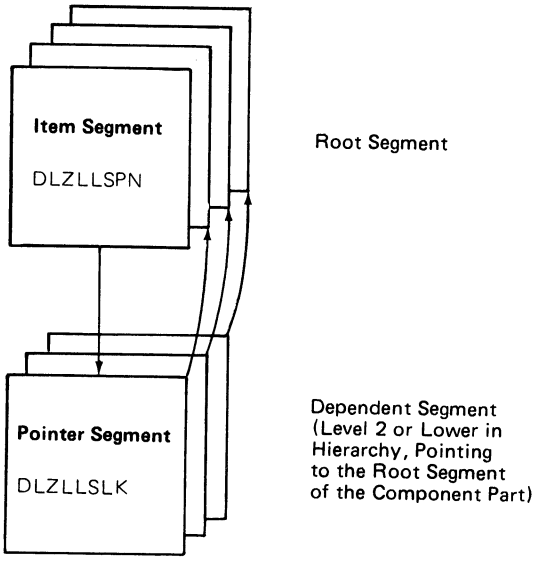


Figure 7. Minimum Parts Data Base (Physical Structure)

Control Data Base

The control data base is an empty data base which contains data only during the execution of LLC/CC. Between two invocations of LLC/CC, the control data base consists of a dummy root segment with key C'A'. Each invocation of LLC/CC implies the insertion into the control data base of a root segment with key C'X' and of a series of level-2 dependent segments. Prior to completion, root segment X and all dependent segments are deleted. One control data base may cover several parts data bases, provided that the keys of the root segments have the same length.

The structure of the control data base is depicted in Figure 8 on page 15. The root segment LLCTL is used as an anchor point for the dependent segments. Segment PARTBEXP stores data for explosion control. It acts as a guide through all relevant hierarchical paths. The segment UPDMASTR saves the old status of the low-level code of all parts which have been changed. The old status of the low-level codes is restored if continuity checking detects an error. Both dependent segments have multiple occurrences.

The three segment types have the following layout:

- Segment LLCTL, length 1 byte, key length 1 byte.
- Segment PARTBEXP, length (2 + length of part key), key length (2 + length of part key), i.e., the segment only consists of a key and does not contain data. The segment key is composed of a 2-byte prefix (either hexadecimal zeros or current LLC), and the part key.

- Segment UPDMASTR, length (length of part key + 2), key length equals length of part key. The key begins at displacement 0 = start position 1. The data portion has a length of 2 bytes. It contains the old LLC, and begins at displacement (length of part key) = start position (length of part key + 1).

The user must not change the segment layout or segment names of the control data base. Failure to correct definition of DBDs results in errors during execution.

The control data base is organized according to the hierarchical indexed direct access method (HIDAM) or the hierarchical direct access method (HDAM).

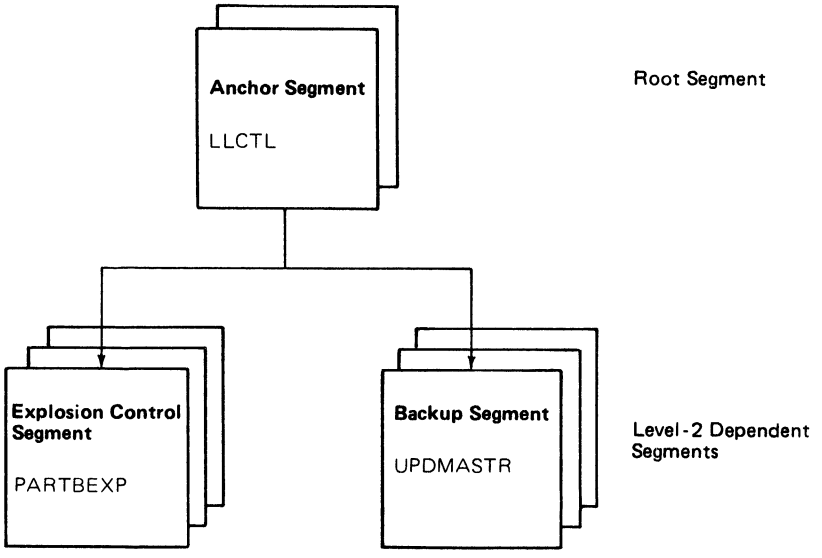


Figure 8. Control Data Base

Definition of DBDs, PSBs, and ACBs

The definition of DBDs, PSBs, and ACBs follows the rules described in *DL/I DOS/VS Resource Definition and Utilities*.

The following figures present a set of sample definitions. Figure 9 on page 16 shows the sample DBDs for a parts data base. The definition assumes a 3-level structure with a separate product structure segment, a data portion in the pointer segment, and no where-used relationships. All names may be changed. Figure 10 on page 17 shows the sample DBDs for a control data base. Names of segments and fields must not be changed. However, if the control data base of the Chained File - DL/I Bridge is used, no separate DBD is required. Figure 11 on page 17 shows a sample PSB used to generate the control data base. Figure 12 on page 17 shows a sample PSB used to execute an Assembler application program invoking LLC/CC.

The names of DBDs and PSBs must not exceed seven characters. For the names selected, the rules for naming phases in the VSE core image library apply. Before selecting a name, check a DSERV listing of the core image library for conflicting names.

Care must be taken when defining the operand for the RULES parameter in the SEGM statement of the generation of the physical DBD for the parts data base. For

both the root segment and the logical child pointer segment, RULES=xxV must be specified, where x can be one of the characters P, L, or V. If the user fails to define it this way, low-level codes may remain unchanged without any notice.

The BYTES parameter in the SEGM statement for the logical child pointer segment indicates the length of this segment. The length is composed of the concatenated key of the physical parent segment, that is, the root segment of the component part, and of the data portion of the logical child segment. If no data is stored in the logical child pointer segment, the BYTES parameter must have the same value as the BYTES parameter in the appropriate FIELD statement of the root segment. However, in the data base only the data portion of the logical child pointer segment is stored; the key portion is handled by DL/I.

```
         DBD     NAME=DLZNNPI,ACCESS=INDEX      INDEX-DB-DEFINITION
         DATASET DD1=DLZLDS1,DEVICE=3330    // DLBL DLZLDS1,'..PNI.CL'
         SEGM    NAME=DLZLLSPI,BYTES=2          KEYLENGTH OF PNM-ROOT
         LCHILD  NAME=(DLZLLSPN,DLZNNPN),INDEX=PNKEY  HIDAM-ROOT
         FIELD   NAME=(PIKEY,SEQ,U),BYTES=2,START=1
         DBDGEN
         FINISH
         END

DBD FOR INDEX DATA BASE OF PARTS DATA BASE


         DBD     NAME=DLZNNPN,ACCESS=HIDAM       (=HIDAM,VSAM)
         DATASET DD1=DLZLDSP,DEVICE=3330    // DLBL DLZLDSP,'..PNP.CL'
         SEGM    NAME=DLZLLSPN,PARENT=0,BYTES=50,                        C
               RULES=VVV          IF NO PTR,PTR=T (DOS/DL1-DEFAULT)
         LCHILD  NAME=(DLZLLSPI,DLZNNPI),PTR=INDX    =INDEX-DB
         LCHILD  NAME=(DLZLLSLK,DLZNNPN)    DOS/DL1-DEFAULT:PTR=SNGL
         FIELD   NAME=(PNKEY,SEQ,U),BYTES=2,START=5,TYPE=C
         FIELD   NAME=PNLLC,BYTES=2,START=10,TYPE=P   LLC-FIELD
         SEGM    NAME=DLZLLSST,PARENT=((DLZLLSPN,SNGL)),                 C
               BYTES=20,PTR=T,RULES=VVL
         FIELD   NAME=(STKEY,SEQ,M),BYTES=4,START=1,TYPE=C
         FIELD   NAME=COMPDATA,BYTES=5,START=5       COMP.QUANT.
         SEGM    NAME=DLZLLSLK,                      LCHILD-SEGMENT      C
               PARENT=((DLZLLSST,SNGL),(DLZLLSPN,VIRTUAL,DLZNNPN)),      C
               BYTES=10,PTR=(T,LP),RULES=VVV         DEFAULT:PTR=(T,..)
         DBDGEN
         FINISH
         END

DBD FOR PHYSICAL PARTS DATA BASE


         DBD     NAME=DLZNNPL,ACCESS=LOGICAL
         DATASET LOGICAL
         SEGM    NAME=DLZLLSPL,PARENT=0,                                 C
               SOURCE=((DLZLLSPN,DATA,DLZNNPN))
         SEGM    NAME=DLZLLSSL,PARENT=DLZLLSPL,                          C
               SOURCE=((DLZLLSST,DATA,DLZNNPN))
         SEGM    NAME=DLZLLLDD,PARENT=DLZLLSSL,                          C
               SOURCE=((DLZLLSLK,DATA,DLZNNPN),(DLZLLSPN,DATA,DLZNNPN))
         DBDGEN
         FINISH
         END

DBD FOR LOGICAL PARTS DATA BASE
```

Figure 9. Sample DBDs for a Parts Data Base

For improved program performance, you should define the largest VSAM control intervals possible (up to 4096) for the control data base. This is done by specifying the BLOCK keyword in the DATASET statement at DBD generation.

```
         DBD     NAME=DLZNNCI,ACCESS=INDEX       INDEX-DB-DEFINITION
         DATASET DD1=DLZLDS2,DEVICE=2314     // DLBL DLZLDS2,'..CTI.CL'
         SEGM    NAME=LLCTLI,PARENT=0,BYTES=1
         LCHILD  NAME=(LLCTL,DLZNNCT),INDEX=CTLKEY
         FIELD   NAME=(CIKEY,SEQ,U),BYTES=1,START=1
         DBDGEN
         FINISH
         END

DBD FOR INDEX DATA BASE OF CONTROL DATA BASE


         DBD     NAME=DLZNNCT,ACCESS=HIDAM       (=HIDAM,VSAM)
         DATASET DD1=DLZLDSC,DEVICE=2314,  // DLBL DLZLDSC,'..CTP.CL'  C
               BLOCK=4096
         SEGM    NAME=LLCTL,PARENT=0,BYTES=1
         LCHILD  NAME=(LLCTLI,DLZNNCI),PTR=INDX   PTR TO INDEX-DB
         FIELD   NAME=(CTLKEY,SEQ,U),BYTES=1,START=1
         SEGM    NAME=PARTBEXP,PARENT=((LLCTL,DBLE)),                  C
               BYTES=4,PTR=TB
         FIELD   NAME=(EXPKEY,SEQ,U),BYTES=4,START=1,TYPE=C
         SEGM    NAME=UPDMASTR,PARENT=((LLCTL,DBLE)),                  C
               BYTES=4,PTR=TB
         FIELD   NAME=(UPDKEY,SEQ,U),BYTES=2,START=1,TYPE=C
         DBDGEN
         FINISH
         END

DBD FOR PHYSICAL DATA BASE OF CONTROL DATA BASE
```

Figure 10. Sample DBDs for a Control Data Base

```
PCB     TYPE=DB,DBDNAME=DLZNNCT,PROCOPT=LS,KEYLEN=5
SENSEG NAME=LLCTL,PARENT=0
PSBGEN LANG=ASSEM,PSBNAME=DLZNNP2
END
```

Figure 11. Sample PSB to Generate the Control Data Base

```
         PCB     TYPE=DB,DBDNAME=DLZNNPL,PROCOPT=AE,KEYLEN=12
         SENSEG NAME=DLZLLSPL,PARENT=0
         SENSEG NAME=DLZLLSSL,PARENT=DLZLLSPL
         SENSEG NAME=DLZLLLDD,PARENT=DLZLLSSL
*
         PCB     TYPE=DB,DBDNAME=DLZNNCT,PROCOPT=AE,KEYLEN=5
         SENSEG NAME=LLCTL,PARENT=0
         SENSEG NAME=PARTBEXP,PARENT=LLCTL
         SENSEG NAME=UPDMASTR,PARENT=LLCTL
*
         PSBGEN LANG=ASSEM,PSBNAME=DLZNNP4
         END
```

Figure 12. Sample PSB to Execute an Assembler Application Program Using LLC/C

Chapter 4. Invocation of LLC/CC in DL/I

This chapter describes how to invoke the services of LLC/CC in DL/I. It shows the relationships between DL/I DOS/VS, a user-written application program executing under control of DL/I DOS/VS, and the LLC/CC subroutine. The information is addressed to the application programmer.

Since LLC/CC is a called subroutine of a user-written application program, the application programmer must be aware of the requirements LLC/CC places on the application program.

- The section "Relationships Between LLC/CC in DL/I and Application Programs" describes the operations that the application program is expected to perform.
- The sections "Application Programs Written in Assembler Language" on page 20 and"Application Programs Written in High-Level Languages" on page 22 explain how to invoke the services of LLC/CC and how to transfer the parameter information.
- The section "Error Recovery" on page 25 presents some aspects of efficient error recovery.

It is assumed that the application programmer is familiar with coding for DL/I DOS/VS, including logical relationships. Refer to *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces.*

Relationships Between LLC/CC in DL/I and Application Programs

An application program using LLC/CC in DL/I as a subroutine may operate in either initial-generation mode or in updating mode.

Initial-generation mode is selected if an existing parts data base representing a manufacturing product-structure network is to be upgraded by introducing low-level codes for each part. The updating mode is used to add a new product-structure relationship to an existing parts data base that already contains low-level codes. An addition usually requires updating of low-level codes, that is, existing low-level codes are checked to determine whether they must be incremented. After updating the new product-structure relationship is ready to be added physically to the data base. If any part in a product structure is contained in itself, either directly or indirectly, the request to generate initially or to update low-level codes is rejected by LLC/CC. The application program receives information on the reason for the rejection and the part causing the loop.

The application program, thus, fully controls the usage of LLC/CC. It identifies the requested services, starts the execution of LLC/CC, and sets up error recovery if required. The application programmer must therefore provide particular functions in his program if he wants to make efficient use of LLC/CC.

Prerequisites for Initial-Generation Mode

The prerequisites for the initial-generation mode are:

- All low-level code fields must contain packed zeros.

- The application program must identify the keys of the parts to be exploded.
- The application program may either scan sequentially the parts data base, or it may read external input, for instance, from tape, to process end items first and raw materials or purchased parts last.
- If the root segment of a particular part already contains a low-level code greater than zero, this part should be bypassed because it no longer needs exploding. However, if LLC/CC is called, no data is changed.
- Since initial generation of low-level codes is a time-consuming function, the job may be split into steps. It is the responsibility of the application programmer to define the range of each step and to ensure that updating mode processing is not used until initial generation is completed.
- The application program must test the return code. If the continuity checking function of LLC/CC detects a loop, the key of the part causing the loop is returned to the application program. The key may identify a component of the part which is actually exploded. When the data base has been corrected, the part which was exploded must be reprocessed in initial-generation mode.
- The application program should provide audit lists indicating the successful assignment of low-level codes or continuity checking information.

Prerequisites for Updating Mode

In updating mode, the prerequisites differ slightly:

- All low-level code fields must contain non-negative packed decimal numbers.
- The application program must identify the keys of the parent part and of the component part of a new product structure. This means that a new bill of material is processed line by line.
- The application program must test the return code. A blank return code indicates that:
  - Low-level codes for the component part and all components of the component part have been properly assigned.
  - The application program may add the new product structure.
- The application program must insert the DL/I segments which establish the new product structure in the parts data base.
- If the status code of LLC/CC is not blank, the new product structure violates the rules for assembly-to-subassembly continuity and must not be added to the parts data base. The application program receives the key of the part causing the loop. After error correction has taken place, the product structure may be resubmitted.
- The application program should provide audit lists indicating successful assignment of low-level codes or continuity checking information.

Application programs invoke the services of LLC/CC by standard CALL statements. The basic information passed to the LLC/CC subroutine is:

- The key of the part to be exploded, for initial-generation mode.
- The keys of parent part and of component part, for updating mode.

The application programmer should be careful with processing the parts data base by the transaction codes GN, GNP, GU, GHU, GHN, or GHNP if this operation is followed by an invocation of LLC/CC because the status of the respective pointers is undefined and position within the data base has been lost.

Apart from the above requirements, the general organization of the application program may be designed according to the needs of the particular environment.

Application Programs Written in Assembler Language

Application programs written in Assembler language use normal CALL conventions to invoke the services of LLC/CC. The user must set up a parameter list, load the address of a register save area into register 13, load the address of the parameter list into register 1, and issue the CALL macro instruction indicating the proper entry point. Upon return to the application program after execution of LLC/CC, the return code should be tested.

The following sequence of instructions applies to the request for initial generation of low-level codes:

```
LA      13,savearea     (load address of save area)
LA      1,parmlist      (load address of parameter list)
CALL    DLZNNGA         (invoke LLC/CC in DL/I)
```

The following sequence of instructions applies to the request for updating of low-level codes:

```
LA      13,savearea     (load address of save area)
LA      1,parmlist      (load address of parameter list)
CALL    DLZNNCA         (invoke LLC/CC in DL/I)
```

The entry point name DLZNNCA in the CALL statement can be modified by the user. For further information refer to the section "Generation of the Execution Program (Macro DLZNN)" on page 28.

The parameter list consists of

- 5 contiguous fullwords in initial-generation mode (parm1, parm2, parm3, parm5, parm6),
- 6 contiguous fullwords in updating mode (parm1 through parm6).

Parameter parm4 must not be present in initial-generation mode.

Each entry in the parameter list contains an address constant. Note that the PCB pointers must be moved directly into the first and the second fullwords of the parameter list when the application program receives control from DL/I.

parm1 Pointer to the PCB of the logical parts data base

parm2 Pointer to the PCB of the control data base

parm3 Pointer to a field containing the key of the part (for initial generation), or the key of the parent part (for updating)

parm4 Pointer to a field containing the key of the component part (for updating only); this parameter must not be present for the initial-generation mode

parm5 Pointer to a 2-byte character field which contains the return code after execution of LLC/CC

parm6 Pointer to a field containing the key of the part causing a loop which has been detected by the continuity checking function of LLC/CC

The length of the fields identified by parm3, parm4, and parm6 equals the length of the part key. The fields identified by parm5 and parm6 need no initialization. The contents of parm1 through parm4 remain unchanged during execution of LLC/CC.

Figure 13 shows parts of an Assembler program that performs both initial generation and updating of low-level codes. It is assumed that during PSB generation the PCB for the parts data base was defined first, followed by the PCB for the control data base. Both pointers thus can be moved with a single instruction.

```
SAMPLE   CSECT
         USING *,R8
           .
           .
           .
         MVC   APCBIN,0(R1)       LOAD PCB'S FOR INITIAL GEN
         MVC   APCBUP,0(R1)       LOAD PCB'S FOR UPDATE
           .
           .
           .
INITGEN  DS    0H                 INITIAL GENERATION
         MVC   PARM3,PARKEY       SPECIFY PARENT PART
         LA    R13,SAVEAREA       ESTABLISH SAVE AREA
         LA    R1,APARMIN         LOAD PARMLIST FOR INIT GEN
*
         CALL  DLZNNGA            CALL LLC/CC IN DL/I FOR INIT GEN
*
         CLC   PARM5,=X'4040'     TEST RETURN CODE FOR BLANKS
           .
           .
           .
UPDATE   DS    0H                 UPDATE
         MVC   PARM3,PARKEY       SPECIFY PARENT PART
         MVC   PARM4,COMPKEY      SPECIFY COMPONENT PART
         LA    R13,SAVEAREA       ESTABLISH SAVE AREA
         LA    R1,APARMUP         LOAD PARMLIST FOR UPDATE
*
         CALL  DLZNNCA            CALL LLC/CC IN DL/I FOR UPDATE
*
         CLC   PARM5,=X'4040'     TEST RETURN CODE FOR BLANKS
           .
           .
           .
```

Figure 13 (Part 1 of 2). Sample Assembler Program

```
*
*        DEFINITIONS OF FIELDS AND CONSTANTS
*
PARKEY   DS    CL2               PARENT KEY INPUT AREA
COMPKEY  DS    CL2               COMPONENT KEY INPUT AREA
*
SAVEAREA DS    18F               REGISTER SAVE AREA
*
PARM3    DS    CL2               PARENT PART
PARM4    DS    CL2               COMPONENT PART
PARM5    DS    CL2               RETURN CODE
PARM6    DS    CL2               LOOPING PART
           .
           .
           .
*
*        PARAMETER LIST FOR INITIAL GENERATION
*
APARMIN  DS    OF                PARMLIST START ADDRESS
*
APCBIN   DS    OCL8              PCB ADDRESSES
AINPARM1 DC    A(0)              - PARTS DATA BASE
AINPARM2 DC    A(0)              - CONTROL DATA BASE
AINPARM3 DC    A(PARM3)          PARENT PART
AINPARM5 DC    A(PARM5)          RETURN CODE
AINPARM6 DC    A(PARM6)          LOOPING PART
*
*        PARAMETER LIST FOR UPDATE
*
APARMUP  DS    OF                PARMLIST START ADDRESS
*
APCBUP   DS    OCL8              PCB ADDRESSES
AUPPARM1 DC    A(0)              - PARTS DATA BASE
AUPPARM2 DC    A(0)              - CONTROL DATA BASE
AUPPARM3 DC    A(PARM3)          PARENT PART
AUPPARM4 DC    A(PARM4)          COMPONENT PART
AUPPARM5 DC    A(PARM5)          RETURN CODE
AUPPARM6 DC    A(PARM6)          LOOPING PART
           .
           .
           .
         END
```

Figure 13 (Part 2 of 2). Sample Assembler Program

Application Programs Written in High-Level Languages

LLC/CC supports application programs written in COBOL and PL/I.

If the application program is written in COBOL, the following CALL statements may be used:

- For initial generation of low-level codes:

```
CALL 'DLZNNGC' USING    parm1,
                        parm2,
                        parm3,
                        parm5,
                        parm6.
```

- For updating of low-level codes:

```
CALL 'DLZNNCC' USING    parm1,
                        parm2,
                        parm3,
                        parm4,
                        parm5,
                        parm6.
```

If the application program is written in PL/I, the following CALL statements may be used:

- For initial generation of low-level codes:

```
CALL DLZNNGP          (parm1,
                       parm2,
                       parm3,
                       parm5,
                       parm6);
```

- For updating of low-level codes:

```
CALL DLZNNCP          (parm1,
                       parm2,
                       parm3,
                       parm4,
                       parm5,
                       parm6);
```

The entry point name DLZNNCC in the CALL statement can be modified by the user. For further information refer to the section "Generation of the Execution Program (Macro DLZNN)" on page 28.

The parameters identify the following fields in the user-written application program:

parm1 PCB of the logical parts data base

parm2 PCB of the control data base

parm3 Field containing the key of the part (for initial generation), or the key of the parent part (for updating)

parm4 Field containing the key of the component part (for updating only); this parameter must not be present for the initial-generation mode

parm5 2-byte character field which contains the return code after execution of LLC/CC

parm6 Field containing the key of the part causing the loop which has been detected by the continuity checking function of LLC/CC

The length of the fields identified by parm3, parm4, and parm6 equals the length of the part key. The fields identified by parm5 and parm6 need no initialization. The contents of parm1 through parm4 remain unchanged during the execution of LLC/CC. In PL/I, parm3 through parm6 must be defined in character format.

Note: The DLZNN module expects the parameter passed in a call from PL/I to point to the locators of the parameters. Do not pass a parameter list with some parameters pointed to directly.

Figure 14 on page 24 shows parts of a COBOL program and Figure 15 on page 25 shows parts of a PL/I program. Both samples show how to invoke LLC/CC for initial generation and for updating of low-level codes.

```
IDENTIFICATION DIVISION.
         •
         •
         •
DATA DIVISION.
         •
         •
         •
   FILE SECTION.
         05  KEY1                              PIC XX.
         05  KEY2                              PIC XX.
         •
         •
         •
   WORKING-STORAGE SECTION.
         •
         •
         •
         05  WC-LLC-RETURN-CODE                PIC XX.
         05  WC-LOOP-KEY                       PIC XX.
         •
         •
         •
         05  WE-PNKEY                          PIC XX.
         05  WE-COMPKEY                        PIC XX.
         •
         •
         •
LINKAGE SECTION.
   01  PN-PCB.
         •
         •
         •
   01  CTL-PCB.
         05  CTL-PCB-DBDNAME                   PIC X(8).
         05  CTL-PCB-LEVEL                     PIC S9(4).
         05  CTL-PCB-STATUS                    PIC XX.

PROCEDURE DIVISION.
      ENTRY 'DLITCBL' USING PN-PCB, CTL-PCB.
         •
         •
         •
300-PROCESS-INI-GENRTN.

MOVE KEY1 TO WE-PNKEY.

CALL 'DLZNNGC' USING PN-PCB, CTL-PCB,
                     WE-PNKEY,
                     WC-LLC-RETURN-CODE, WC-LOOP-KEY.

400-PROCESS-UPDATE.
      MOVE KEY1 TO WE-PMKEY.
      MOVE KEY2 TO WE-COMPKEY.

      CALL 'DLZNNCC' USING PN-PCB, CTL-PCB,
                           WE-PNKEY, WE-COMPKEY,
                           WC-LLC-RETURN-CODE, WC-LOOP-KEY.
         •
         •
      GOBACK.

      STOP RUN.
```

Figure 14. Sample COBOL Program

```
DLITPLI:  PROC (PCB1, PCB2) OPTIONS (MAIN);
          DCL   PLITDLI ENTRY,
                DLZNNCP ENTRY,
                DLZNNGP ENTRY,
                PCB1    POINTER,
                PCB2    POINTER;

          DCL   PARTS_PCB BASED (PCB1),
                   .
                   .
                   .

          DCL   CNTRL_PCB BASED (PCB2),
                   .
                   .
                   .

          DCL   PART CHAR (2),
                PAR_PART CHAR (2),
                COMP_PART CHAR (2);

          DCL   RET_CODE CHAR (2) INIT ('  '),
                LOOP_PART CHAR (2) INIT ('  ');
                   .
                   .
                   .
/* ROUTINE FOR INITIAL GENERATION OF LOW-LEVEL CODES */
                   .
                   .
                   .
                CALL DLZNNGP (PARTS_PCB,
                              CNTRL_PCB,
                              PART,
                              RET_CODE,
                              LOOP_PART);
                IF RET_CODE = '  '
                   THEN;
                      DO;   . . .    ; END;
                   ELSE;
                      DO;   . . .    ; END;
                   .
                   .
                   .
/* ROUTINE FOR MAINTENANCE OF LOW-LEVEL CODES */
                   .
                   .
                   .
                CALL DLZNNCP (PARTS_PCB,
                              CNTRL_PCB,
                              PAR_PART,
                              COMP_PART,
                              RET_CODE,
                              LOOP_PART);
                IF RET_CODE = '  '
                   THEN;
                      DO;   . . .    ; END;
                   ELSE;
                      DO;   . . .    ; END;
                   .
                   .
```

Figure 15. Sample PL/I Program

Error Recovery

If an error occurs during the execution of an application program calling the services of LLC/CC the user must investigate carefully the error conditions. The error may be caused by various reasons:

- The application program calling for the services of LLC/CC is not properly cleared of errors.
- DL/I control blocks are incorrectly specified.

- The data bases are incomplete, damaged, or destroyed.
- The contents of the data bases are incomplete, damaged, or destroyed.
- The organization of the user's data processing system does not ensure data base integrity.
- A system component does not execute accurately.

System errors may be caused by incorrect operation procedures, for instance:

- The PSB does not ensure exclusive use of the data bases during execution.
- Not all updates of the parts data base are performed when LLC/CC is used.

The user of LLC/CC is entirely responsible for a correct application of this function.

To facilitate error recovery, a return code is always returned to the application program. Abnormal conditions are indicated by a non-blank return code (refer to "Return Codes" on page 39 for further details). The application programmer is responsible for the setting up of appropriate procedures for error recovery. In general, their implementation depends on the particular environment; however, some guidelines may help the programmer.

Return codes beginning with the characters D, E, and F signal that the data bases are damaged. In this case, the application program should display all available information including the contents of the PCBs and immediately terminate processing. In addition, a VSAM VERIFY run is recommended. (For information on the VERIFY command, see *Using VSE/VSAM Commands and Macros*, SC24-5144.)

All other non-blank return codes allow processing to continue. However, it is the application programmer's responsibility to specify under which conditions continuation is appropriate.

In initial-generation mode, continuity checking may detect multiple violations. Under these circumstances it is recommended to continue processing and to produce a complete list of all loops. If LLC/CC rejects the request for assignment of low-level codes, the application program must print an exception notice indicating the exploded part and the part causing the loop. This information is used by the data base administrator to correct the data base, that is, to delete erroneous product structures and to add correct product structures. After correction, all affected parts must be reprocessed in initial-generation mode. It is recommended therefore, that the application program punch a card for each exploded part for which low-level code assignment is rejected. These cards are resubmitted in a second run in initial-generation mode. However, if corrections of the data base affect more parts than originally detected, it is advisable to reset all low-level code fields to zero and to reprocess all parts in initial-generation mode.

In updating mode, a continuity check indicates that a particular product structure must not be added. Normally, it results from an error when preparing a new bill of material, that is, when a wrong key is entered for the particular component. However, in some cases existing data in the parts data base may be wrong. The application program may then either reject the new bill of material altogether or only defer insertion of the erroneous line in the bill of material.

After the bill of material and the parts data base have been corrected, updating may be retried.

If an existing data base has been processed in initial-generation mode by LLC/CC, and if the updating mode detects continuity errors in existing data, an initial-generation run is advisable. This allows improper structures to be cleaned up, and reduces the amount of rework during future updating runs.

Because error recovery involves many functions outside the data processing department, it is up to the judgment of the application programmer to decide how to proceed if an error occurs.

Chapter 5. Operational Procedures

The following sections describe the procedures required to operate LLC/CC and to execute user-written application programs calling the facilities of LLC/CC. Samples of job input streams are included. The reader should modify the samples (for instance, disk addresses, names for data sets, PSBs, and phases) to meet the requirements for a particular environment. If job control becomes lengthy, the user should consider storing a general set of job control statements in the procedure library.

Distribution of LLC/CC in DL/I

LLC/CC in DL/I is a component of Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS), Program Product 5746-XX1. The machine-readable material of DL/I DOS/VS contains two macro definitions which provide the facilities of LLC/CC in DL/I:

- The macro DLZNN generates the logic to execute LLC/CC in DL/I.
- The macro DLZNNICT generates the logic to initialize the control data base, which is a prerequisite for executing LLC/CC in DL/I.

The macros are distributed both in edited and unedited format. LLC/CC in DL/I is stored on the DL/I DOS/VS distribution tape.

- The macros in edited format are included in file 1 of the distribution tape. They are cataloged as E-books in the source statement library.
- File 3 of the distribution tape contains the macro instructions in unedited format. They may be cataloged as A-books in the source statement library.

For full details on cataloging and maintenance, refer to *DL/I DOS/VS Resource Definition and Utilities*.

Generation of the Execution Program (Macro DLZNN)

To generate executable code for LLC/CC in DL/I, the macro definition DLZNN is assembled and stored in the relocatable library. A set of macro parameters is used to tailor the program to meet the needs of the user's individual environment.

The macro instruction has the following format:

```
DLZNN     [COMPNAM=DLZLLLDD,
                   name     ]
          [COMPLEN=KEYLEN,
                   length]
          [DLZNNCA=DLZNNCA,
                   name    ]
          [DLZNNCC=DLZNNCC,
                   name    ]
          [DLZNNCP=DLZNNCP,
                   name    ]
          [DLZNNEC=DLZNNEC,
                   name    ]
          [DLZNNGA=DLZNNGA,
                   name    ]
          [DLZNNGC=DLZNNGC,
                   name    ]
          [DLZNNGP=DLZNNGP,
                   name    ]
          [INGEN=YES,
                 NO ]
          KEYDIS=displacement,
          KEYLEN=length,
          LLCDIS=displacement,
          [MODNAM=DLZNN,
                  name ]
          [PNKEY=PNKEY,
                 name ]
          [ROOTNAM=DLZLLSPL,
                   name     ]
          [BRIDGE=NO,
                  YES]
          ROOTLEN=length
```

For all optional parameters, which are indicated by brackets, a default operand (indicated by underscoring) is provided. When entering the input statements, the macro syntax of the Assembler language applies. In particular, the rules that apply to continuation lines must be observed carefully.

```
[COMPNAM=DLZLLLDD,
         name     ]
```

Name of the component segment in the logical parts data base. This segment consists of the pointer segment of the parent part and of the root segment of the component part. The default name for this segment is DLZLLLDD.

```
[COMPLEN=KEYLEN,
         length]
```

Length of the pointer segment which establishes the relationship to the root segment of the component part as defined in the generation of the physical DBD for the parts data base. The length is composed of the concatenated key of the logical parent segment, that is, of the key of the component part, and of the length of data stored in the pointer segment. If no data is stored, COMPLEN is equal to KEYLEN. If COMPLEN is omitted, no data is assumed, and KEYLEN is taken as default.

```
[DLZNNCA=DLZNNCA,
         name    ]
```

Name of the entry point for Assembler application programs to update low-level codes. The name must not exceed seven characters. The default name is DLZNNCA.

```
[DLZNNCC=DLZNNCC,
         name   ]
```

Name of the entry point for COBOL application programs to update low-level codes. The name must not exceed seven characters. The default name is DLZNNCC.

```
[DLZNNCP=DLZNNCP,
         name   ]
```

Name of the entry point for PL/I application programs to update low-level codes. The name must not exceed seven characters. The default name is DLZNNCP.

```
[DLZNNEC=DLZNNEC,
         name   ]
```

External name for the LLC/CC Execution Control Block (LECB). The name must not exceed seven characters. The default name is DLZNNEC.

```
[DLZNNGA=DLZNNGA,
         name   ]
```

Name of the entry point for Assembler application programs to initially generate low-level codes. The name must not exceed seven characters. The default name is DLZNNGA.

```
[DLZNNGC=DLZNNGC,
         name   ]
```

Name of the entry point for COBOL application programs to initially generate low-level codes. The name must not exceed seven characters. The default name is DLZNNGC.

```
[DLZNNGP=DLZNNGP,
         name   ]
```

Name of the entry point for PL/I application programs to initially generate low-level codes. The name must not exceed seven characters. The default name is DLZNNGP.

```
[INGEN=YES,
       NO ]
```

Inclusion of the routine to initially generate low-level codes. The default is inclusion.

```
KEYDIS=displacement,
```

Position of the part key in the root segment of the parts data base relative to the beginning of the data area of the root segment.

Note: Because LLC/CC is written in assembler language, the displacement value specified must be relative to zero. (The first character position in the data area is displacement 0.)

```
KEYLEN=length,
```

Length of the part key.

```
LLCDIS=displacement,
```

Position of the low-level code field in the root segment of the parts data base relative to the beginning of the data area of the root segment.

Note: Because LLC/CC is written in assembler language, the displacement value specified must be relative to zero. (The first character position in the data area is displacement 0.)

```
[MODNAM=DLZNN,
        name ]
```

Definition of the name of the CSECT and of the module which will be stored in the relocatable library. The default name is DLZNN. The macro instruction automatically produces an appropriate CATALR statement preceding the object code. The module name defined by parameter MODNAM must be entered into the INCLUDE statement when the user-written application program is link-edited to LLC/CC. In conjunction with the definition of different entry point names, this allows you to call multiple LLC/CC modules within a single application program to service several data bases concurrently. (Refer to the section "Link-Editing of Application Programs" on page 35 for further information.)

```
[PNKEY=PNKEY,
       name ]
```

Name of the key field in the root segment. This field will be defined in the DBD generation for the physical parts data base. The default name is PNKEY.

```
[ROOTNAM=DLZLLSPL,
         name     ]
```

Name of the root segment in the logical parts data base. The default name is DLZLLSPL.

```
[BRIDGE=NO,
        YES]
```

Specification of BRIDGE=YES develops code which assumes utilization of the control data base as created by the Chained File - DL/I Bridge, Program Product 5748-XX3. In this case, no separate control data base need be generated for executing a phase containing DLZNN.

```
ROOTLEN=length
```

Length of the data area of the root segment in the parts data base.

Incorrect specification of operands for mandatory parameters results in an abnormal termination of code generation. MNOTE statements are used to display error messages (refer to "Error Messages" on page 38 for further information). All segment names refer to the names as defined during DBD generation for the logical parts data base. When changing the name of the CSECT or the names of the entry points, the user should carefully check the cross reference table of an assembly list for duplicate names.

A sample job input stream to generate and to catalog object code is given in Figure 16. The job control statements assume private libraries. The object code generated by the Assembler is stored in the private relocatable library and is ready to be link-edited to user-written application programs. The macro parameters of the sample apply to the sample DBD described in the section "Definition of DBDs, PSBs, and ACBs" on page 15.

```
// JOB GOELLASS (0795,4),T=6
// OPTION DECK
// ASSGN SYS020,DISK,VOL=DS2PR2,SHR
// ASSGN SYSSLB,SYS020
// DLBL IJSYSSL,'DOS.DLZ.LLC.MACRO'
// EXTENT SYSSLB,DS2PR2
// ASSGN SYSRLB,SYS020
// DLBL IJSYSRL,'DOS.DLZ.LLC.OBJECT'
// EXTENT SYSRLB,DS2PR2
// DLBL IJSYSCL,'DOS.DLZ.LLC.CILIB'
// EXTENT SYSCLB,DS2PR2
   ASSGN SYSCLB,SYS020
   ASSGN SYSPCH,UA
   ASSGN SYSPCH,X'361'
// EXEC PGM=ASSEMBLY
         DLZNN KEYDIS=4,KEYLEN=2,LLCDIS=9,ROOTLEN=50,COMPLEN=10
         END
/*
   CLOSE SYSPCH,X'00D'
   ASSGN SYSIPT,UA
   ASSGN SYSIPT,X'361'
// EXEC PGM=MAINT
   CLOSE SYSIPT,X'00C'
// EXEC PGM=DSERV
   DSPLYS ALL
/*
/&
   ASSGN SYSCLB,UA
```

Figure 16. Sample Job Input Stream for DLZNN

The generated object code applies to all application programs requiring low-level code generation or updating, that is, a single copy of the object code covers all functions and all source code languages. Regeneration of the object code is required only if modifications to the layout of the data base affect fields and segments which are used by LLC/CC.

Generation of the Initialization Program for the Control Data Base (MACRO DLZNNICT)

The macro instruction DLZNNICT needs no customization. It is assembled, link-edited, and stored in the core image library.

A sample job input stream to generate and to catalog executable code is given in Figure 17 on page 33. The job control statements assume private libraries. The executable code is stored in the private core image library and is ready for execution by DL/I. The name of the phase is DLZNNICT.

```
// JOB GOEASSIC (0795,4),T=5
// OPTION CATAL
// ASSGN SYS020,DISK,VOL=DS2PR2,SHR
// ASSGN SYSSLB,SYS020
// DLBL IJSYSSL,'DOS.DLZ.LLC.MACRO'
// EXTENT SYSSLB,DS2PR2
// ASSGN SYSRLB,SYS020
// DLBL IJSYSRL,'DOS.DLZ.LLC.OBJECT'
// EXTENT SYSRLB,DS2PR2
// DLBL IJSYSCL,'DOS.DLZ.LLC.CILIB'
// EXTENT SYSCLB,DS2PR2
   ASSGN SYSCLB,SYS020
// EXEC PGM=ASSEMBLY
         DLZNNICT
         END
/*
// EXEC PGM=LNKEDT
// EXEC PGM=DSERV
   DSPLYS ALL
/*
/&
   ASSGN SYSCLB,UA
```

Figure 17. Sample Job Input Stream for DLZNNICT

For a successful link-editing run, the DL/I module DLZLI000 and the device-independent I/O module IJJFCBZD must be present in the relocatable library.

Generation of the DL/I Control Blocks

The Data Base Descriptions (DBDs), the Program Specification Blocks (PSBs), and the Application Control Blocks (ACBs) are generated in exactly the same way as for any other application program running under DL/I. (Refer to *DL/I DOS/VS Resource Definition and Utilities*.) For detailed information on the specification of the various parameters, refer to the section "Definition of DBDs, PSBs, and ACBs" on page 15.

Note that the names of the DBDs and PSBs must not exceed seven characters. For the names selected, the rules for naming phases in the core image library apply. Moreover, a DSERV listing of the core image library should be checked for conflicting names.

Preparation of the VSAM Master Catalog

The VSAM master catalog contains all information to describe fully a data set that is organized by VSAM. The Access Method Services are used to create and to maintain the VSAM master catalog (refer to the publication *Using VSE/VSAM Commands and Macros*, SC24-5144).

Both the parts data base and the control data base are DL/I data bases organized in HIDAM format. This implies two virtual storage access method (VSAM) clusters for each data base.

A key-sequenced file (KSDS) cluster represents the index portion, and an entry-sequenced file (ESDS) cluster represents the data portion of a HIDAM data base. A KSDS cluster consists internally of two separate data sets.

To prepare the VSAM master catalog, the options DELETE, DEFINE, and LISTCAT of the Access Method Services are used. It is recommended that a DELETE run be executed before any DEFINE run. This automatically deletes all information about obsolete VSAM data sets and facilitates recovery if a data set has become inoperable, for example, during testing. Figure 18 on page 34 shows a sample job

input stream to define the entries in the VSAM master catalog describing the control data base. Notice that the VSAM control interval size has been set to the maximum, 4096 or 4K bytes. This is necessary to assure optimum performance for LLC/CC.

A VERIFY run may be required if a VSAM data set was not correctly closed. The VERIFY option resets certain indicators in the VSAM master catalog, according to the actual status of the data sets.

To avoid mismatches of entries in the DL/I control blocks and in the VSAM master catalog, the DBD generation of DL/I produces a checklist. This facilitates definition of parameters for the DEFINE option of the Access Method Services.

```
// JOB WRNALLOC (0795,4),T=06
// ASSGN SYS020,DISK,VOL=DS2PR2,SHR
// DLBL KSDINDX,,0,VSAM                    KSDS-INDEX
// EXTENT SYS020,DS2PR2,1,0,640,20
// DLBL KSDDATA,,0,VSAM                    KSDS-DATA (ROOTKEYS)
// EXTENT SYS020,DS2PR2,1,0,660,20
// DLBL ESDS,,0,VSAM                        SEGMENT-AREA
// EXTENT SYS020,DS2PR2,1,0,680,20
// EXEC IDCAMS,SIZE=100K
   DELETE DOS.DLZLLCTI.CL          /*DEL DB-INDEX FROM MSTR.CTLG*/
      FILE(KSDINDX)
      NOERASE;
   DELETE DOS.DLZLLCTP.CL          /* DEL DB-CONTENTS FROM MSTR.CTLG*/
      FILE(ESDS)
      NOERASE;
   DEFINE CLUSTER (NAME(DOS.DLZLLCTI.CL)
          /* KSDS OF CTL - DATABASE (=INDEX-DB) */
      INDEXED
      UNIQUE)
    DATA (NAME(DOS.DLZLLCTI.INDX.DATA)
          /* DATA PORTION OF KSDS */
      FILE(KSDDATA)
      VOL(DS2PR2)
      TRACKS(5)
      RECORDSIZE(12,20) /*ROOTKEY,PREFIX(6),DL1-INT(5) PLUS 1*/
      KEYS(1,10)
      RECOVERY
      NOWRITECHECK)
   INDEX(NAME(DOS.DLZLLCTI.INDX.INDX)
          /* INDEX PORTION OF KSDS */
      FILE(KSDINDX)
      TRACKS(5)
      VOL(DS2PR2))
   CATALOG(AMASTCAT);
   DEFINE CLUSTER(NAME(DOS.DLZLLCTP.CL)
          /* ESDS (CLUSTER) OF CTL-DATABASE (NAME=DLZLLCTP)*/
      NOINDEXED
      UNIQUE)
   DATA(NAME(DOS.DLZLLCTP.DATA)  /* ESDS (DATA) OF CTL-DB*/
      FILE(ESDS)
      VOL(DS2PR2)
      TRACKS(10)
      RECORDSIZE(4086,4086)  /* 4096 MINUS 10(VSAM CI)*/
      CONTROLINTERVALSIZE(4096)
      RECOVERY
      NOWRITECHECK)
   CATALOG(AMASTCAT);
   LISTCAT ALL;
/*
/&
```

Figure 18. Sample Job Input Stream for Definition of the Control Data Base to VSAM

Initialization of the Control Data Base

The control data base is initialized by inserting a single root segment in load mode. Its key is the character A. This root segment remains permanently in the control data base. Therefore, temporary insertions of control and backup information into the control data base during execution of LLC/CC are performed in updating mode.

The initialization is a prerequisite for the execution of LLC/CC. It requires a particular PSB with PROCOPT=LS (refer to "Definition of DBDs, PSBs, and ACBs" on page 15 for further information). If, for any reason, the control data base becomes inoperable, the entries in the VSAM master catalog for the two clusters defining the index portion and the data portion of the control data base must be deleted and redefined (refer to "Preparation of the VSAM Master Catalog" on page 33 for further information). After reestablishing the entries in the master catalog, the control data base may be reinitialized. This procedure also applies if the initialization fails.

The initialization program is stored in the core image library under the phase name DLZNNICT. The user generates it by means of the macro instruction DLZNNICT. After generation, a list output is printed, reporting either successful execution or failure. If initialization fails, the status code returned by DL/I is displayed. Additional information may be produced by DL/I on the system console.

Figure 19 shows a sample job input stream which corresponds to the sample generation and to the definitions for DBDs and the PSB given in Figure 10 on page 17 and Figure 11 on page 17. Since initialization does not change any data, no logging is required. It is important to allocate as many DL/I buffers as possible to the control data base. This is done by specifying only the DBD name of the control data base in the HDBFR parameter of the DL/I control statement.

```
// JOB GOECTLLD (0795,4),T=8
// UPSI 00000010
// ASSGN SYS020,DISK,VOL=DS2PR2,SHR
// ASSGN SYSSLB,SYS020
// DLBL IJSYSSL,'DOS.DLZ.LLC.MACRO'
// EXTENT SYSSLB,DS2PR2
// ASSGN SYSRLB,SYS020
// DLBL IJSYSRL,'DOS.DLZ.LLC.OBJECT'
// EXTENT SYSRLB,DS2PR2
// DLBL IJSYSCL,'DOS.DLZ.LLC.CILIB'
// EXTENT SYSCLB,DS2PR2
   ASSGN SYSCLB,SYS020
// DLBL IJSYSCT,'AMASTCAT'
// EXTENT SYSCAT,DS2PR0
// ASSGN SYS021,SYS020
// DLBL DLZLDS2,'DOS.DLZLLCTI.CL',,VSAM
// EXTENT SYS021,DS2PR2
// ASSGN SYS022,SYS020
// DLBL DLZLDSC,'DOS.DLZLLCTP.CL',,VSAM
// EXTENT SYS022,DS2PR2
// EXEC PGM=DLZRRC00,SIZE=280K
DLI,DLZNNICT,DLZNNP2,2,HDBFR=(32,DLZNNCT)
/*
/&
   ASSGN SYSCLB,UA
```

Figure 19. Sample Job Input Stream for Initialization of the Control Data Base

Link-Editing of Application Programs

LLC/CC in DL/I is a called subroutine for user-written application programs. The user must link-edit an application program to LLC/CC. The default name entry in the directory of the relocatable library is DLZNN. Since the various entry point names of LLC/CC do not appear explicitly in the directory, the user must provide an INCLUDE statement preceding the EXEC statement:

```
INCLUDE       DLZNN
              user-defined name
// EXEC       PGM=LNKEDT
```

If the user has selected a module name other than DLZNN during customization of the LLC/CC in DL/I execution program, the operand in the INCLUDE statement must be replaced by the user-defined name. (Refer to the description of the parameter MODNAM in "Generation of the Execution Program (Macro DLZNN)" on page 28.)

The output of the linkage editor should be checked carefully to determine whether the correct entry points have been selected by the application program.

For successful link-editing, the DL/I module DLZLI000 must be present in the relocatable library.

For application programs written in high-level languages, certain additional modules must be INCLUDEd. Refer to *DL/I DOS/VS Application Programming: CALL and RQDLI Interfaces*, under "DL/I Application Programming."

Execution of LLC/CC in DL/I

Application programs using the services of LLC/CC are executed as normal application programs running under DL/I. They are stored fully link-edited in the core image library. Execution is initialized via the DL/I phase DLZRRC00. A parameter statement indicates that normal DL/I support is requested, and identifies the names of the application program and of the related PSB, both of which are stored in the core image library.

Care must be taken to select the correct PSB. It is particularly recommended to define PROCOPT=E or PROCOPT=A for the parts data base. This ensures integrity of the parts data base during all low-level code operations, thus preventing the data base from being tampered with.

Figure 20 on page 37 shows a sample job input stream for the execution of an application program calling LLC/CC. Normally, the control statements do not differ for initial generation or updating of low-level codes. XXX is the phase name of the application program, YYY is the name of the PSB. The DBD name of the control data base (DLZNNCT) should be specified in the HDBFR parameter during execution in the same manner as it was during initialization. In all cases of abnormal conditions, the output of the system console should be checked for additional information produced by DL/I.

```
// JOB GOEEXEC (0795,4),T=25
// UPSI 0
// ASSGN SYS020,DISK,VOL=DS2PR2,SHR
// ASSGN SYSSLB,SYS020
// DLBL IJSYSSL,'DOS.DLZ.LLC.MACRO'
// EXTENT SYSSLB,DS2PR2
// ASSGN SYSRLB,SYS020
// DLBL IJSYSRL,'DOS.DLZ.LLC.OBJECT'
// EXTENT SYSRLB,DS2PR2
// DLBL IJSYSCL,'DOS.DLZ.LLC.CILIB'
// EXTENT SYSCLB,DS2PR2
   ASSGN SYSCLB,SYS020
// DLBL IJSYSCT,'AMASTCAT'
// EXTENT SYSCAT,DS2PR0
// ASSGN SYS021,SYS020
// DLBL DLZLDS2,'DOS.DLZLLCTI.CL',,VSAM     INDEX FOR CONTROL DB
// EXTENT SYS021,DS2PR2
// ASSGN SYS022,SYS020
// DLBL DLZLDSC,'DOS.DLZLLCTP.CL',,VSAM     DATA FOR CONTROL DB
// EXTENT SYS022,DS2PR2
// ASSGN SYS015,SYS020
// DLBL DLZLDSI,'DOS.DLZLLPNI.CL',,VSAM     INDEX FOR PARTS DB
// EXTENT SYS015,DS2PR2
// ASSGN SYS016,SYS020
// DLBL DLZLDSP,'DOS.DLZLLPNP.CL',,VSAM     DATA FOR PARTS DB
// EXTENT SYS016,DS2PR2
// ASSGN SYS011,X'380'
// TLBL LOGOUT,'DOS.DLZ.LLC.BACKUP'         BACKUP TAPE
// EXEC PGM=DLZRRC00,SIZE=280K
DLI,XXX,YYY,2,HDBFR=(32,DLZNNCT)
          .
          .
          .
/*
/&
   ASSGN SYSCLB,UA
```

Figure 20. Sample Job Input Stream for Execution of LLC/CC Operations

Chapter 6. Error Messages and Return Codes

This section describes the error messages and return codes which may be encountered during the generation and execution of LLC/CC facilities. In many cases, additional information is produced on the system console by DL/I DOS/VS, by VSAM, and by the system (refer to *DL/I DOS/VS Messages and Codes*, SC24-5146, and *VSE/VSAM Messages and Codes*, SC24-5146, and *VSE/Advanced Functions Messages*, SC33-6098).

Error Messages

LLC/CC produces error messages during the generation of the execution program from the macro instruction DLZNN, and during the initialization of the control data base. Error messages are displayed on the printer.

Generation of the execution program (macro instruction DLZNN):

LLC001I PARAMETER parm MISSING, GENERATION ABORTED

Cause: A mandatory parameter parm was omitted.

Action: Complete macro parameter definition.
Resubmit.

LLC002I PARAMETER KEYDIS ILLEGAL, GENERATION ABORTED

Cause: The displacement for the key points beyond the end of the root segment.

Action: Check parameters KEYDIS and ROOTLEN. Correct the erroneous parameter.

LLC003I PARAMETER KEYDIS AND/OR KEYLEN ILLEGAL, GENERATION ABORTED

Cause: The key overlaps the end of the root segment.

Action: Check parameters KEYDIS, KEYLEN, and ROOTLEN. Correct the erroneous parameter(s).

LLC004I PARAMETER LLCDIS ILLEGAL, GENERATION ABORTED

Cause: Displacement and length of the low-level code field do not fit into the root segment.

Action: Check parameters LLCDIS and ROOTLEN. Correct the erroneous parameter.

LLC005I PARAMETER COMPLEN ILLEGAL, GENERATION ABORTED

Cause: The specified length of the pointer segment is less than the length of the key in the parameter KEYLEN.

Action Check parameters COMPLEN and KEYLEN. Correct the erroneous parameter.

Initialization of the control data base:

LLC100I CONTROL DATA BASE HAS BEEN INITIALIZED

Cause: Successful completion.

Action: Continue.

LLC101I CONTROL DATA BASE INITIALIZATION FAILED DL/I STATUS CODE = xx

Cause: This message is issued if DL/I returns a non-blank status code when the program attempts to load a root segment. This message may occur under many conditions, for instance:

- Conflicting definitions in DBD and VSAM master catalog,
- PSB does not indicate PROCOPT=LS.
- VSAM master catalog was not reset after an unsuccessful attempt to initialize the control data base.

Action The DL/I status code which is returned after loading has been attempted is represented by xx.

1. Check the DL/I status code.
2. Check the console output for additional information produced by DL/I. DL/I messages begin with the characters DLZ.
3. Correct errors.
4. Reset entries in the VSAM master catalog by using the DELETE and DEFINE options of the Access Method Services.
5. Resubmit.

Return Codes

After completion of operation, LLC/CC posts a return code into the 2-byte field identified by parameter parm5 of the calling application program. Refer to Chapter 4, "Invocation of LLC/CC in DL/I" on page 18 for further information on the specification of parameters. The return code consists of two alphameric characters which indicate one of the following three conditions:

- Successful completion - ␣␣ (two blank characters).
- Rejection of the request - return codes CC, CP, EC, EP, FC, FP, IP, NC, NP.

  The request by the application program was not satisfied because of conflicting structures of the parts data base. Data in the parts data base is not changed or it is restored to its previous state. The user should correct the contents of the data base and/or the input data of the application program.

- Data Base Error - return codes DC, DP, EC, EP, FC, FP.

  LLC/CC did not satisfy the request because an unexpected DL/I status code was encountered during execution. The leftmost 240 bytes of the related PCB are saved in a save area. The address of the save area is stored in a fullword which may be addressed by a pointer which is stored at DLZNNEC+12. DLZNNEC is the name of an entry point in LLC/CC. The fullword starting at DLZNNEC+8 points to the last parameter list submitted to DL/I. The actual status of the parts data base is not predictable. However, if the DL/I status signals an open error, the data bases are not likely to be affected.

Return codes starting with the letters E and F indicate both an invalid request and a data base error.

CC A part other than the part identified by parameter parm3 (that is, a component part of any level) is contained in itself either directly or indirectly. The key of the looping part is posted into the field identified by parameter parm6.

Action: The application program may continue. Correct input and/or parts data base; resubmit.

CP The part identified by parameter parm3 (that is, the parent part) is contained in itself either directly or indirectly. The key is posted into the field identified by parameter parm6.

Action: The application program may continue. Correct input and/or parts data base; resubmit.

DC Unexpected DL/I status code encountered when accessing the control data base.

Action: Display all available information, discontinue processing of the application program. Display contents of the parameter fields parm1 through parm6. Test whether input has already been processed. Obtain the output of the system console, check messages originating from DL/I and/or from VSAM. Reset the VSAM master catalog by means of a VERIFY run.

If the control data base is damaged, reinitialize. (Refer to "Initialization of the Control Data Base" on page 34 for further information.)

DP Unexpected DL/I status code encountered when accessing the parts data base.

Action: Display all available information, discontinue processing of the application program. Display contents of the parameter fields parm1 through parm6. Test whether input has already been processed. Obtain the output of the system console, check messages originating from DL/I and/or from VSAM. Reset the VSAM master catalog by a VERIFY run.

If the parts data base is damaged, use the standard DL/I reconstruction procedures. Resubmit application program only after the error has been fixed.

EC Both error conditions CC and DP have occurred.

Action: See code DP. Thereafter, correct input and/or parts data base; resubmit.

EP Both error conditions CP and DP have occurred.

Action: See code DP. Thereafter, correct input and/or parts data base; resubmit.

FC Both error conditions CC and DC have occurred.

Action: See code DC. Thereafter, correct input and/or parts data base; resubmit.

FP Both error conditions CP and DC have occurred.

Action: See code DC. Thereafter, correct input and/or parts data base; resubmit.

IP Erroneous input parameters, for instance:

- Same part specified by parm3 and parm4 of the input parameter list (in updating mode only).
- A parameter received from the application program is not likely to be a valid address.

Action: The application program may continue. Correct input; resubmit.

NC The component part identified by parameter parm4 cannot be found (in updating mode only).

Action: The application program may continue. Correct input and/or parts data base; resubmit.

NP The parent part identified by parameter parm3 cannot be found.

Action: The application program may continue. Correct input and/or parts data base; resubmit.

b b Successful completion, all affected low-level codes are properly assigned.

Action: Insert the new product-structure relationship between the parts identified by the parameters parm3 and parm4 (in updating mode only). Continue processing.

Chapter 7. Installation Requirements

To use LLC/CC in DL/I, a user must perform the following activities:

- Tailor LLC/CC, using the macro facilities of the Assembler language and the appropriate library of object modules.
- Write and test appropriate application programs.
- Generate an appropriate parts data base. The low-level code field of 2 bytes is assumed to be located in the root segment. Establish logical relationship with the root segments of the component parts, using a pointer segment (admitted at any hierarchical level). The pointer segments should not contain data.
- Generate a control data base, using the utility program DLZNNICT. Users of the IBM System/370 Chained File - DL/I Bridge, Program Product 5748-XX3, may use the control data base which has been generated by this program.

If initial generation of low-level codes is planned to be executed by LLC/CC, all low-level code fields must contain packed zeros. In this case, updating must not be performed until the total data base has been completely processed by initial generation.

If only updating of low-level codes is planned, all parts must contain the correct low-level code reflecting the actual position of each part within the product structure. The low-level code is a 2-byte packed decimal number.

Both data bases must be generated by DL/I DOS/VS, Version 1.1 or later.

Chapter 8. Performance Considerations

It is not possible to give quantitative performance estimates. Performance figures vary within a wide range. However, certain particular aspects should be taken into account:

- Complexity of product structures involved. The number of components to be updated influence execution time to a great extent. Also, the number of short paths on a product structure tree has an impact on program performance; the more short paths there are, the faster the execution.

- The size of the data base and its structure, in conjunction with the buffer size in main storage, influences number and duration of physical input and output operations on direct access storage devices.

- A well-defined numbering scheme for part keys allows closely related parts to be stored in the same area. Furthermore, performance is likely to improve if end items are processed first and parts on low levels last when performing the initial generation of low-level codes or when adding stacks of new product structures. This implies that identifiers for end items should have a low position in the sorting sequence, while raw materials and purchased parts should have a high position.

Proper continuity checking requires extensive input/output activities. Therefore, a user should not attempt to process large amounts of input data in a single step.

Chapter 9. Control, Audit, and Reconstruction Procedures

All reliability, availability, and serviceability (RAS) facilities of DL/I apply without restrictions to user-written application programs utilizing LLC/CC.

LLC/CC maintains a separate control data base which contains an entry for all parts whose low-level code has been altered. If a loop is detected during continuity checking, the control data base is used to restore the previous state of the low-level codes. The PSB of the control data base is defined for exclusive control so that single-thread processing is ensured.

Initial generation of low-level codes is likely to become a time-consuming function. Therefore, this function may be divided into several small jobs, thus reducing the impact of a system breakdown.

If a system breakdown occurs during the execution of an application program using LLC/CC, the log tape should be used to reestablish the original state of the parts data base. However, the parts data base is not affected by reprocessing of requests for low-level code operations which have already been completed successfully. The control data base needs only to be regenerated.

Refer to "Error Recovery" on page 25 for further information about the actual recovery from abnormal conditions.

Chapter 10. System Configuration

LLC/CC operates within a minimum configuration as defined for DL/I DOS/VS, Version 1.1 or later. A virtual address space of approximately 4K bytes is required to execute LLC/CC in DL/I.

The control data base requires space on direct access storage devices. The amount of space to be allocated depends on the length of the part number and on the maximum number of subordinate parts which may be encountered when exploding an end item into its components.

Chapter 11. Programming Systems

LLC/CC is written in the Assembler language. It is distributed as a part of DL/I. The object code is generated after the definition of macro parameters and after assembly.

The object code of LLC/CC may be used as a subroutine of application programs written in Assembler language, in COBOL, or in PL/I. If the user plans to write application programs in high-level languages, he requires one or more of the following programs:

- DOS/VS COBOL Compiler and Library, 5746-CB1 or Library only, 5746-LM4
- Full ANS COBOL V3 Compiler, 5736-CB2, and Full ANS COBOL Library, 5736-LM2
- ANS Subset COBOL, 5736-CB1
- ANS COBOL, 360N-CB-482 or 370-CB-482
- PL/I Optimizing Compiler and Libraries, 5736-PL3
- PL/I Optimizing Compiler, 5736-PL1
- PL/I Resident Library, 5736-LM4
- PL/I Transient Library, 5736-LM5

Chapter 12. Bibliography

IBM Corporation. *System/360 Data Base Organization and Maintenance Processor General Information Manual.* GH20-0771.

IBM Corporation. *System/370 Chained File - DL/I Bridge: General Information Manual.* GH12-5116.

IBM Corporation. *Manufacturing Data Base Guide: Bill Processor Systems to DL/I.* GH20-1593.

IBM Corporation. *DL/I Data Base Techniques for Manufacturing Applications.* GE20-0480.

Index

LLC/CC in DL/I DOS/VS
Program Reference and Operations Manual
Order No. SH20-9046-3

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

LLC/CC in DL/I DOS/VS P. R. O. M. (File No. S370/4300-50)

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

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

Fold | Fold

If you would like a reply, *please print:*

Your Name _____

Company Name _____ *Department* _____

Street Address _____

City _____

State _____ *Zip Code* _____

IBM Branch Office serving you _____

IBM®

SH20-9046-3

LLC/CC in DL/I DOS/VS
Program Reference and Operations Manual
Order No. SH20-9046-3

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

LLC/CC in DL/I DOS/VS P. R. O. M. (File No. S370/4300-50)

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

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

Fold | Fold

If you would like a reply, *please print:*

Your Name _____

Company Name _____ *Department* _____

Street Address _____

City _____

State _____ *Zip Code* _____

IBM Branch Office serving you _____

IBM®

SH20-9046-3

LLC/CC in DL/I DOS/VS
Program Reference and Operations Manual
Order No. SH20-9046-3

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

LLC/CC in DL/I DOS/VS P. R. O. M. (File No. S370/4300-50)

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

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

Fold | Fold

If you would like a reply, *please print:*

Your Name _____

Company Name _____ *Department* _____

Street Address _____

City _____

State _____ *Zip Code* _____

IBM Branch Office serving you _____

IBM®

SH20-9046-3

SH20-9046-3

LLC/CC in DL/I DOS/VS P. R. O. M. (File No. S370/4300-50)

SH20-9046-3

IBM®

SH20-9046-03