Program Product

Information Management System/360 for the IBM System/360 Program Description

Program Number 5736-CX3

The Information Management System/360 is an Operating System/360 processing program designed to facilitate the implementation of medium to large common data bases in a multiapplication environment. This environment is created to accommodate both online message processing and conventional batch processing, either separately or concurrently. The system permits the evolutionary expansion of data processing applications from a batch-only to a teleprocessing environment.

This manual contains a description of the functions concerned with IMS/360 in a data processing environment and of the system and its facilities provided; a discussion of data base and application program structure and details; and systems and terminal operations interfaces. An appendix includes application program examples in both COBOL and PL/I. An index is included to facilitate the use of the manual.

IBM

Inside Front Cover of SH20-0634-1
Revised March 19, 1971
By TNL SN20-2339

Second Edition (July 1970)

This edition applies to Version 1, Modification Level 2, of the Program Product Information Management System/360 for the IBM System/360, 5736-CX3.

This is a major revision obsoleting H20-0634-0. Besides correcting errors, this edition contains additions and changes supporting Release 18 of the Operating System.

Here is a summary of the major new and changed items in this revision.

- PL/I message program structure example is clarified.
- PL/I and COBOL batch program structure examples are clarified.
- Sensitive segments should not be listed in PCB working storage mask.

Other changes to the text and small changes to illustrations are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol ● to the left of the caption.

This edition applies to Release 18 of IBM System/360 Operating System and to all subsequent releases until otherwise indicated in new additions or Technical Newsletters.

Changes are continually made to the specifications herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (GN20-0360), for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form for readers' comments is provided at the back of this publication. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 1133 Westchester Avenue, White Plains, New York 10604.

The Program Description Manual is one of a set of manuals prepared to define the various functions and personnel relationships involved in the implementation and system operation of Information Management System/360 (IMS/360). The other manuals in the set are:

IMS/360 Application Description Manual (GH20-0524)

IMS/360 Operations Manual, Volume I - Systems Operation (SH20-0635)

IMS/360 Operations Manual, Volume II - Machine Operations (SH20-0636)

IMS/360 System Manual, Volume I - Program Logic (LY20-0431)

IMS-360 System Manual, Volume II - Flowcharts (LY20-0432)

This introductory chapter restates some of the same information found in the introductory chapter of the Systems Operation and Machine Operations Manuals.

The necessity for these manuals became apparent during the design phase of the IMS/360. The usual mix of data processing personnel normally provides for application programming, system programming, and machine operations functions. With the introduction of IMS/360, however, the need for a fourth function, a coordinating force in implementing, administering, and maintaining the system, became apparent. The function is the "heart" of the IMS/360 system and has been designated the "Systems Operation function"; it is so referred to herein. The application programming interface with the Systems Operation function is delineated in this manual (see Figure 1).

An understanding of the following is a prerequisite for a thorough comprehension of this manual:

IMS/360 Application Description Manual

OS/360 COBOL or PL/I Language (GC28-6516 or GC28-8201)

This manual gives the application programmer a view of all the functions and facilities provided by IMS/360 for application development and serves as both a general information manual and a reference manual. It is so structured that the reader may obtain a basic understanding of all he needs to know about IMS/360 to design and write application programs using the system.

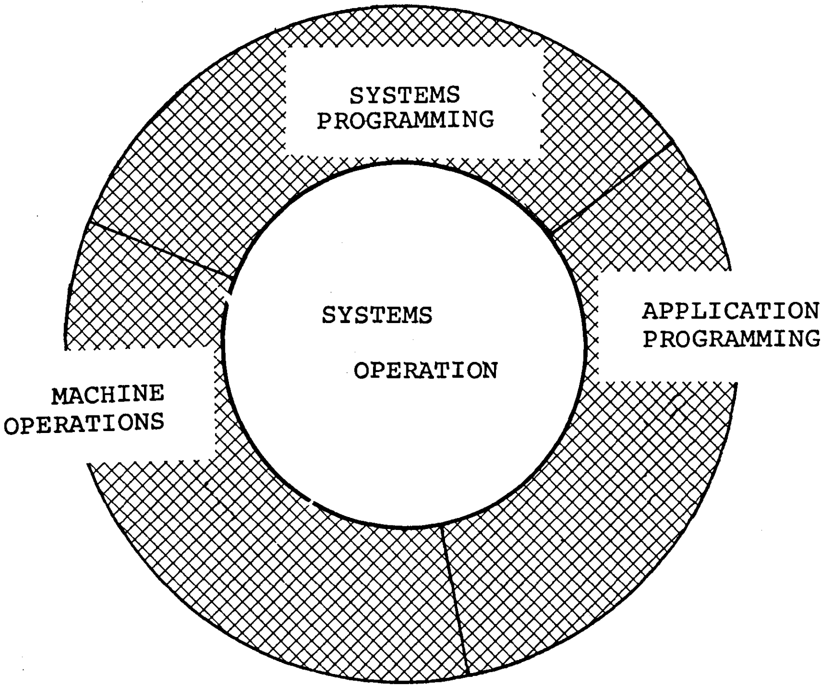


Figure 1. IMS/360 functional relationships

APPLICATION PROGRAMMING FUNCTION

The Systems Operation function provides for applications planning, implementation, and audit. The application programming function must consider the following in its analysis of a proposed application:

- Configuration and storage device requirements for anticipated applications
- Data base structuring, storage device cost/performance tradeoffs, and sharing of mutual data with existing data bases
- Program structuring, core limits, duration of execution, overlay structure, and program chaining
- Message formats and length, transaction types, priorities, passwords, and logical terminal names
- Schedule of data base checkpoints and checkpoint cost versus reconstruction cost
- Schedule of data base dumps and reorganization

SYSTEMS OPERATION FUNCTION

The functions of Systems Operation encompass the following:

- Configuration planning, for all purposes, of new applications so that communication lines, consoles, and software are available to support approved applications

- Responsibility for control over and approval of all new data base designs and descriptive control blocks

- Maintenance of the data bases under Data Language/I, including all control, allocation, and data base generation

- Maintenance of a catalog of programs "certified" to operate as message processing programs under IMS/360, including related documentation, processing priorities, transaction codes, control blocks, etc.

- Responsibility to provide the capability for reconstruction and recovery of IMS/360 and its associated data bases when routine procedures known and understood by the Machine Operations function are insufficient for such recovery and reconstruction. The Systems Operation function also has the responsibility to be available to participate in such extraordinary operations whenever they are required.

- Responsibility for the utility programs that process the IMS/360 system log tapes and for causing these programs to process the log tapes and to yield accounting information, machine operations statistics, usage and data base statistics, and certain management reports on utilization and errors incurred. The function shall also have the responsibility for auditing these reports for quality and for assigning certain reports to other functions for analysis, as appropriate.

- Responsibility for IMS/360 system generation and modification

- Maintenance of all IMS/360 documentation

SYSTEMS PROGRAMMING FUNCTION

The functions of Systems Programming encompass the following:

- Assistance and participation in the hardware installation, test, and initial operations of any new equipment or changed configurations

- Consultation with IMS/360 Application Programmers in conjunction with the Systems Operation function to assist in the integration of applications with IMS/360

- Software maintenance and improvement of IMS/360 utility programs and modifications to Operating System/360

MACHINE OPERATIONS FUNCTION

In addition to the usual operational assignments, the Machine Operations function shall be responsible for:

- All master terminal capabilities in accordance with established procedures, with especially prepared instructions to cover extraordinary happenings

- Assisting terminal operators at remote terminals in the initial diagnoses of apparent problems, be they concerned with the remote terminal, the connecting communication line, the central hardware, the central software, or message processing application programs. After the initial diagnoses, the Machine Operations function should have accumulated sufficient information to determine whose assistance is required and to intelligently describe the problem, and will be able to assist in determining the degree of emergency sustained.

CHAPTER 2. DESCRIPTION OF IMS/360

The Information Management System/360 (IMS/360) is a set of control program modules designed to operate under the control of and within the framework of Operating System/360. The intent is to give the user of Operating System/360 the ability to construct large data bases and to interface with the data in an efficient teleprocessing manner. To gain maximum utilization of the resources of IMS/360, a multiprogramming environment is required and is obtained through the facilities of Operating System/360 with Multiprogramming with a Fixed Number of Tasks (MFT) or Multiprogramming with a Variable Number of Tasks (MVT).

At system IPL time (see Figure 2), the Operating System/360 nucleus is brought into core storage to become the foundation of this multiprogramming environment. The highest priority region of Operating System/360 is used for the IMS/360 resident control program. The remainder of the available core storage is divided into message regions and batch regions, depending upon user requirements.

OS/360 NUCLEUS	REGION 0	REGION 1	REGION 2	REGION 3
	CONTROL PROGRAM	PROCESSING	PROCESSING	PROCESSING

Figure 2. Operating System/360-IMS/360 multiprogramming environment

The Operating System/360 nucleus and its resident extensions provide the nucleus resident service modules, resident access methods, SVC's, storage protect control, etc. that are required when running in an IMS/360 environment.

The IMS/360 control program region includes all the resident control modules and facilities available to the application program. These facilities are:

1. Communications Control, which provides terminal polling, message receiving, message validity checking, input message enqueuing, output message dequeuing, message sending, and other user and terminal control functions.

2. Application Scheduler, which considers input messages for processing after being signaled by communications control of their availability. The application scheduler checks for the availability of resources (message processing regions, message processing program, data base, and data base buffers) in the IMS/360 region. If all required resources are available, messages are scheduled for processing on a priority basis. The application scheduler also provides for the orderly termination of application programs for normal or abnormal reasons.

3. Data Language/I, which provides the application programmer access to data bases in a manner that allows him not only a high degree

of device independence but also data management software independence. (See "Definition of Data Language/I" below.)

4. Checkpoint and Restart, which provides for recovery of system data bases and message queues in the event of system failure, and for normal restart of IMS/360. Checkpoint also condenses system message queues and assists in dumping and restoring data bases for reconstruction or audit. The checkpoint facility is an integral part of restart. Both normal startup and restart after system failure are accomplished using the last or a previous checkpoint from the IMS/360 system log.

DEFINITION OF DATA LANGUAGE/I

The traditional limitation of every data processing application has been the organization of the data to be manipulated. The structure of each data record and its manner and medium of storage have affected application design and programming, and a great deal of effort has been expended to free the data organization from the physical restriction of the data storage medium.

It is the purpose of Data Language/I to allow the application program to gain a high degree of independence from the input/output software systems and storage devices that are required for storage and manipulation of the data. As seen in Figure 3, Data Language/I provides a "wall" or separation between the application program and the data bases. An application program has two distinct interfaces with Data Language/I: (1) a data base description, a mapping or transformation relating the logical data structure and physical storage structure of the data base given as a definition external to the application program; and (2) a common source program linkage (referred to later in this manual as the application program language interface), which allows data input/output requests during the execution of the application program.

A second, and possibly more important, purpose of Data Language/I is to provide a medium through which a programmer can have access to large data files not specifically built and organized for his use. This should lead to the ability to combine common data into a single data base rather than maintain redundant data. Data Language/I relieves the application program of the necessity of knowing the physical location of its data in the data bases. The application program requests input/output data base operations of Data Language/I, using the logical data relationships of the application. Data Language/I translates or maps this logical data relationship to the physical storage of the data. In this manner, the physical storage of data may be changed, and, if the logical data relationships are retained, the application programs need not be modified.

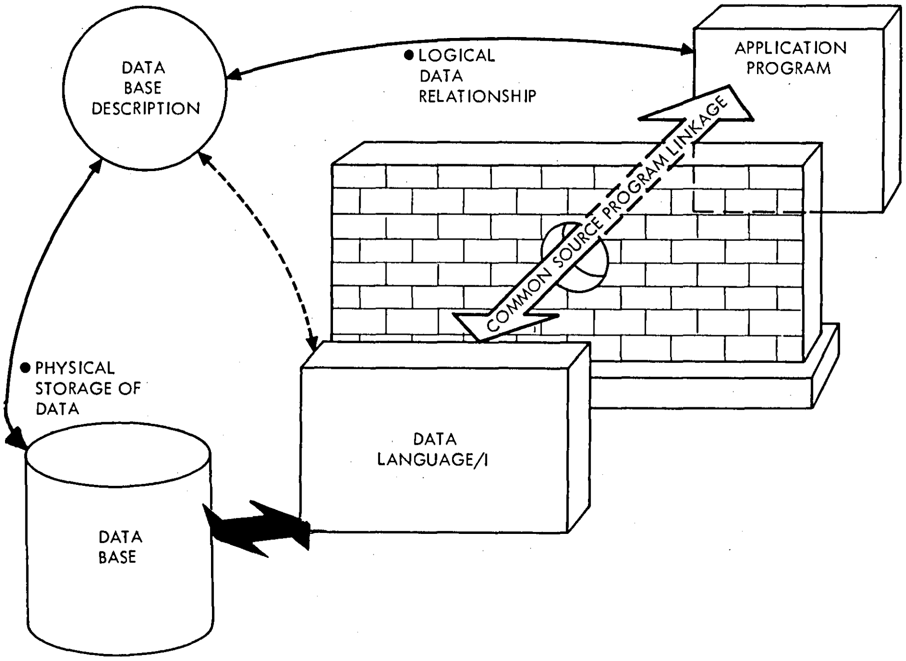


Figure 3. Data Language/I relationship between application program and data base

The availability of noncustomized data brings into full meaning the concept of a data base. In this context, the ability to create and access large data files having multiple uses and eliminating redundant information takes on real meaning.

The data base processing capabilities of IMS/360 are represented by Data Language/I and form an important part of IMS/360. Note, however, that Data Language/I can operate independently of the IMS/360 teleprocessing facilities used exclusively for batch processing. It is also used in the batch processing environment to create (load) all batch and message processing data bases. Data bases cannot be created by a message processing program.

The following are the significant capabilities available to the Data Language/I user:

1. A common source program interface is provided between the application program and the data base.

2. A data base description provides a mapping from the logical data relationships to the physical storage of the data. This description is maintained external to the application program.

3. The ability for a program to define the portions of a data base to which it wishes to be "sensitive" (that is, to have access) without considering the total data base structure permits the organization of nonsensitive data to be changed or added to without affecting application programs.

4. Data Language/I uses Operating System/360 fixed-length ISAM with an improved capability for data insertion and overflow control.

5. In the teleprocessing environment, data base security is assisted through a password technique.

6. In a multiprogramming environment, controlled access is provided during update operations to maintain integrity of a data base.

7. A hierarchical data element relationship is introduced between the various portions of a data base. This permits the handling of variable-length data structures in a fixed-length manner. Simplification of data handling should be experienced in COBOL or PL/I application programs.

8. A utility program is provided for use in describing and storing a definition of the data base structure. (See "Data Base Description".)

9. A utility program is provided for use in describing the application program's data base "sensitivity and usage". (See "Program Specification Block".)

DATA LANGUAGE/I VS OPERATING SYSTEM/360 DATA MANAGEMENT

This portion of the manual shows the relationship of Data Language/I to Operating System/360 Data Management, lists the difference between the two, and defines the terminology associated with each.

Data Language/I - Data Base

The data base concept is introduced by Data Language/I in IMS/360. In order to define the term data base, its relationship to the Operating System/360 Data Management data set should first be defined. The SRL publication, *IBM System/360 Operating System Supervisor and Data Management Services*, says, "Any information that is a named, organized collection of logically related records can be classified as a data set.... A data set may be...a file of data records used by a processing program." The data set is the major unit of data storage and retrieval in Operating System/360. Figure 4 shows the Operating System/360 Data Management data set structure to be made up of physical records, which are further broken down into logical records. The only relationship between the physical and logical data structure provided by Operating System/360 is one or more logical records within a physical record.

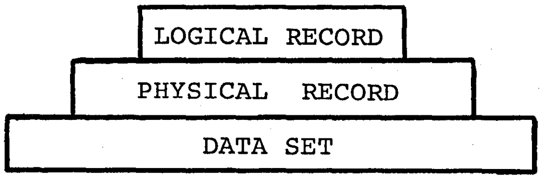


Figure 4. OS/360 data management data set structure

Data Language/I, in order to accommodate variable-length application records (data base records), provides the capability of a logical record within a physical record or a logical record spanning one or more physical records.

A data base may be considered similar to a data set because it is an organized collection of data entered and maintained in some logical sequence to facilitate later inquiry and processing.

In the application logical data sense, the data base is composed of data base records (Figure 5). The data base record is the logical record of the application. A data base record is a collection (of variable number) of fixed-length data elements, called "segments", hierarchically related to a single occurrence of a root segment. A segment is a portion of a data base record containing one or more logically related data fields. A root segment is the highest hierarchical segment in the data base record. Each data base record must have only one root segment. The root segment comprises data which applies to all users in the processing of the data base record. A dependent segment is a segment that relies on at least the root segment for its full hierarchical meaning. It is therefore always at a lower hierarchical level than the root segment. A dependent segment may also be dependent on other dependent segments for its full meaning. In order to process the segments in a data base, it is only necessary for the user to be aware of those segments which comprise the data base record and the relationship of these segments to each other (that is, the logical data base record structure of segments). There can be 255 segment types within a data base and 15 levels of segment hierarchy within a data base record.

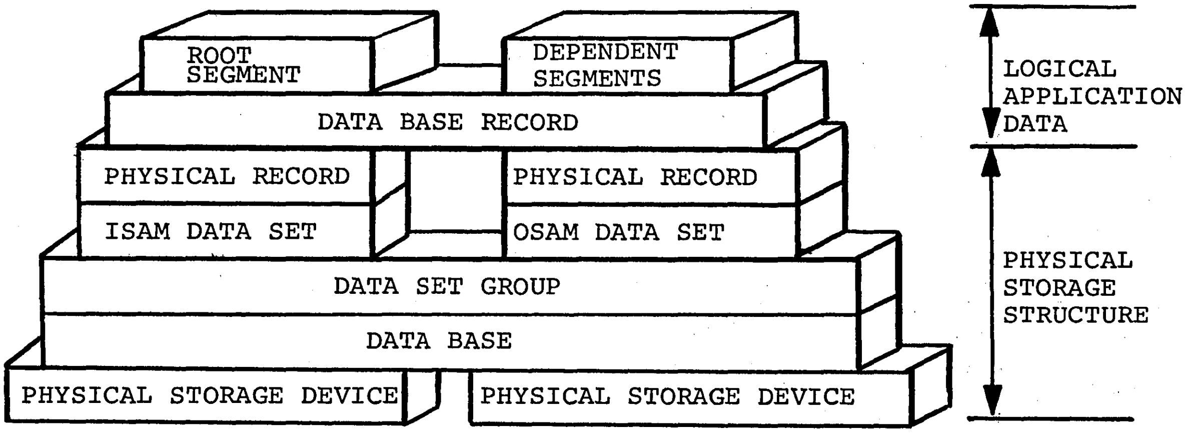


Figure 5. IMS/360 Data Language/I data base structure

Referring to Figure 5, note that each data base, in the physical sense, is composed of at least one data set group. Each data set group consisting of one or more data sets is dependent upon the organization of the data base for exact definition. The data base/data set group concept represents the Data Language/I expansion of the Operating System/360 data set concept in the storage of data. The data set group concept allows Data Language/I to accommodate variable-length logical records even with the constraints of available storage devices. The user of a data base (that is, application program) is insensitive to the number of data set groups which comprise the data base (the physical structure of the data base) and which may change periodically on the basis of the processing and storage requirements of the data base. The user should view the data base as a collection of data base records, not of data set groups.

The descriptive information of logical data base record-segment relationships and the physical device and data set group description used by Data Language/I are stored apart from the data base and application program in a Data Base Description (DBD). The DBD is built using the Data Base Description generation utility program and must be completed before data base creation or application program execution.

APPLICATION DEVELOPMENT AND STRUCTURING OF IMS/360

When the user of the IMS/360 system initiates the definition of an application program to operate with IMS/360, the following must be performed:

1. The definition of each Data Language/I data base in terms of its hierarchical structure and storage, and the creation or load of data into each data base in the batch processing environment, using the capabilities of Data Language/I

2. The definition and construction of all message and batch processing programs and the control blocks that define how a program intends to use a data base

3. The definition of various message types and their associated processing programs, scheduling priorities, and security aspects

4. The definition of the logical and physical communications terminal and line network utilized by the application

The user must also structure IMS/360 by the creation of a control block for each communication line, terminal, message type, message processing program, and data base. The construction and integration of these control blocks into the resident IMS/360 control program are facilitated by the use of the utility programs. Restructuring of the control blocks will be necessary periodically.

A detailed description of the events that must occur before execution of the IMS/360 control program with the user's application program and data bases follows (see Figure 6). Figures 7 through 12 describe the component functions that are the user's responsibility. All block numbers refer to Figure 6.

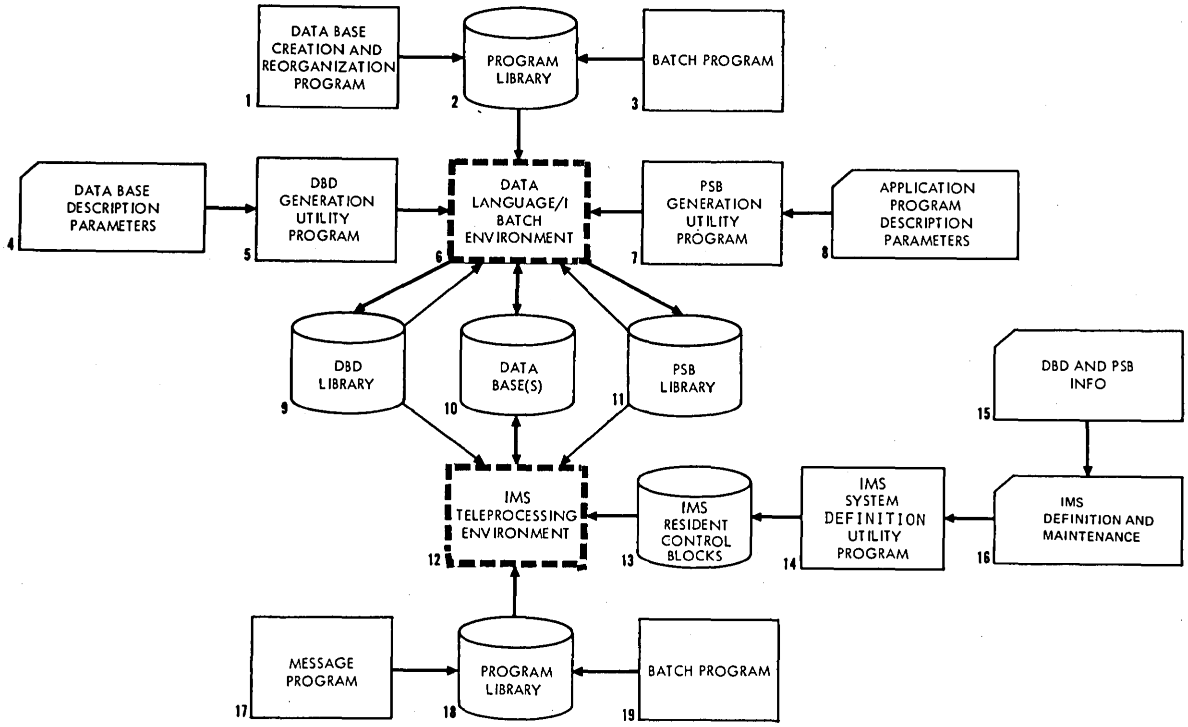

DATA BASE CREATION AND REORGANIZATION PROGRAM
1
PROGRAM LIBRARY
2
BATCH PROGRAM
3
DATA BASE DESCRIPTION PARAMETERS
4
DBD GENERATION UTILITY PROGRAM
5
DATA LANGUAGE/I BATCH ENVIRONMENT
6
PSB GENERATION UTILITY PROGRAM
7
APPLICATION PROGRAM DESCRIPTION PARAMETERS
8
DBD LIBRARY
9
DATA BASE(S)
10
PSB LIBRARY
11
DBD AND PSB INFO
15
IMS TELEPROCESSING ENVIRONMENT
12
IMS RESIDENT CONTROL BLOCKS
13
IMS SYSTEM DEFINITION UTILITY PROGRAM
14
IMS DEFINITION AND MAINTENANCE
16
MESSAGE PROGRAM
17
PROGRAM LIBRARY
18
BATCH PROGRAM
19

Figure 6. Events for IMS/360 use

1. The user must create a Data Base Description (DBD, Block 4) for each data base associated with an application program. He then uses the Data Base Description of the data base as input to the DBD generation utility program (Block 5) in the Data Language/I batch environment (Block 6) (or OS/360 batch environment). The resultant DBD is stored as a member of an OS/360 partitioned data set called the DBD library (Block 9). See Figure 7.

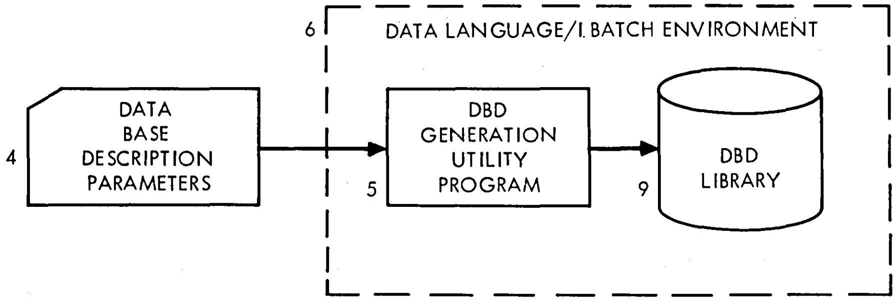

6
DATA LANGUAGE/I BATCH ENVIRONMENT
4
DATA BASE DESCRIPTION PARAMETERS
5
DBD GENERATION UTILITY PROGRAM
9
DBD LIBRARY

Figure 7. Data base description

2. Next, the user must create three items. First, he must create a data base creation (load) program (Block 1) for each data base. Second, he must create the description of each load program's data base requirements (Application Program Description, Block 8) according to the parameters of the Program Specification Block (PSB) generation utility program (Block 7). Third, he must use the data base creation program description (PSB, Block 8) as input to the PSB generation utility program (Block 7) in the Data Language/I batch environment (Block 6) (or OS/360 batch environment). The resultant PSB is stored as a member of an OS/360 partitioned data set called the PSB library (Block 11). See Figure 8.

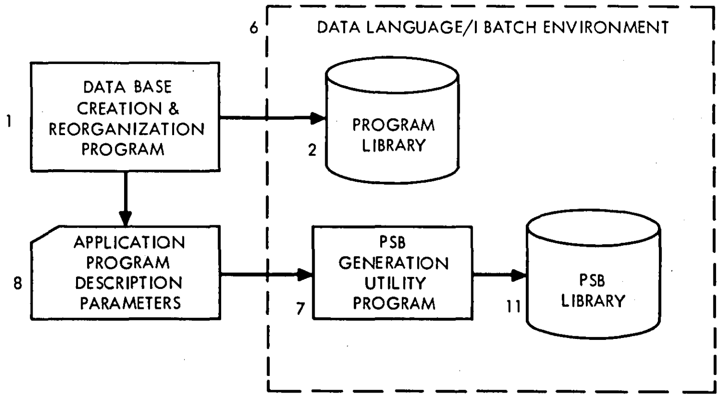

6
DATA LANGUAGE/I BATCH ENVIRONMENT
1
DATA BASE CREATION & REORGANIZATION PROGRAM
2
PROGRAM LIBRARY
8
APPLICATION PROGRAM DESCRIPTION PARAMETERS
7
PSB GENERATION UTILITY PROGRAM
11
PSB LIBRARY

Figure 8. Data base creation

3. The user's data base load program may now be executed either for loading or for reorganizing the data base. The program requires the Data Base Description (DBD) for that particular data base, and the Program Specification Block (PSB) associated with the data base load program. See Figure 9.

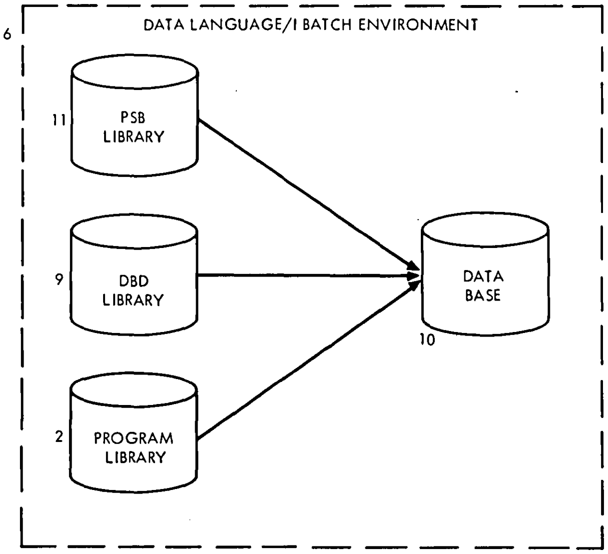


Figure 9. Creation or reorganization into batch environment

4. Once all data bases have been created, the user must create a PSB (Blocks 8 and 7) for each message processing program. The user must place all programs that use Data Language/I into a user program library. The name of each PSB is identical to the name of the program with which it is associated. See Figure 10.

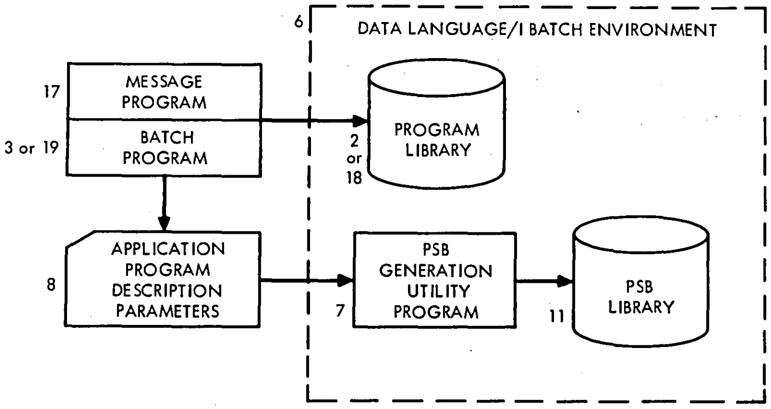

6
DATA LANGUAGE/I BATCH ENVIRONMENT
17
MESSAGE PROGRAM
3 or 19
BATCH PROGRAM
2 or 18
PROGRAM LIBRARY
8
APPLICATION PROGRAM DESCRIPTION PARAMETERS
7
PSB GENERATION UTILITY PROGRAM
11
PSB LIBRARY

Figure 10. Storing in library

5. The user must supply information about each DBD and PSB to the Systems Operation function. This information is needed for IMS/360 control program definition and maintenance (Block 15 input to Block 16). See Figure 11.

6. Using the information in Step 5, above, and the IMS/360 System Definition Utility program (Block 14), the user creates (or updates) the IMS/360 control program with the resident information (Block 13) necessary for the execution of his application program and for incorporation of new (or modified) data bases and application programs. See Figure 11.

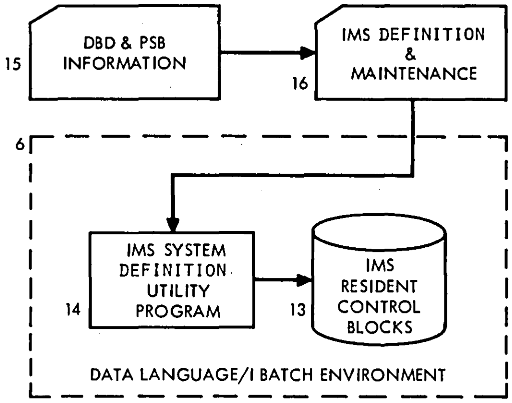

15
DBD & PSB INFORMATION
16
IMS DEFINITION & MAINTENANCE
6
14
IMS SYSTEM DEFINITION UTILITY PROGRAM
13
IMS RESIDENT CONTROL BLOCKS
DATA LANGUAGE/I BATCH ENVIRONMENT

Figure 11. System definition and maintenance

7. After the above steps have been completed, execution of the application programs with the applicable data bases, either in an IMS/360 teleprocessing environment (Block 12) or a Data Language/I batch environment (Block 6), occurs. See Figure 12.

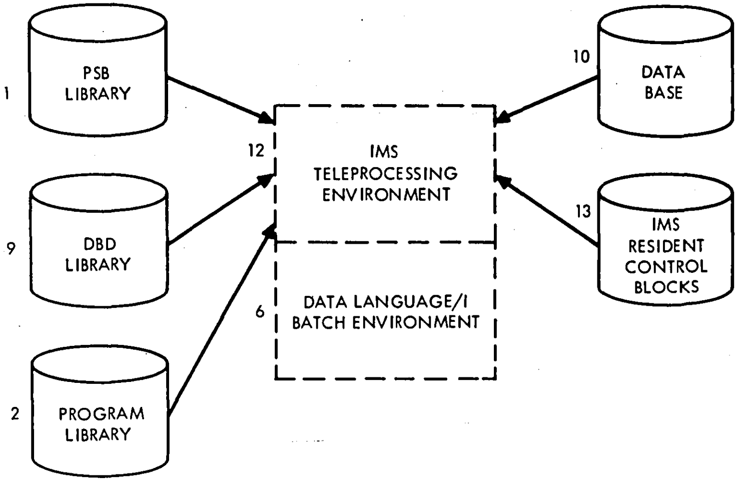

11
PSB LIBRARY
9
DBD LIBRARY
2
PROGRAM LIBRARY
12
IMS TELEPROCESSING ENVIRONMENT
6
DATA LANGUAGE/I BATCH ENVIRONMENT
10
DATA BASE
13
IMS RESIDENT CONTROL BLOCKS

Figure 12. Execution begins

INTERFACE WITH SYSTEMS OPERATION

A Systems Operation function is intended to provide for applications planning, implementation, and audit. The application programmer must provide enough of the following information for Systems Operation to assist in the analysis of the proposed application:

1. Configuration and storage device requirements for anticipated applications

2. Data base structuring, storage device cost/performance tradeoffs, and commonality of data with existing data bases

3. Program structuring, core limits, duration of execution, overlay structure, and program chaining

4. Message formats and length, transaction types, priorities, passwords, and logical terminal names

5. Schedule of data base checkpoints and checkpoint cost versus reconstruction cost

6. Schedule of data base dumps and reorganization

The application programmer must interface with Systems Operation, which provides the following for creation and maintenance of libraries and logs:

1. Naming conventions for Data Base Descriptions (DBD's), Program Specification Blocks (PSB's), and application programs

2. Allocation and maintenance of libraries for DBD's, PSB's, application programs, and IMS/360 PSB and Data Management Block (DMB) directories

3. Data Base Description generation and Program Specification Block generation

4. Certification and final incorporation of programs into application program library

5. Logs of logical terminal names, transaction codes, priorities, and passwords

6. Schedules of data base checkpoints and data base reorganization

7. Master terminal operations

8. IMS/360 system status

9. Trouble logs for data base, program, system, lines, terminal operators, and documentation

Systems Operation must provide to the application programmer:

1. Assignment of disk packs, physical arrangement of data bases, and audit of volume and overflow activity

2. Procedures for checking out new applications in the IMS/360 production environment

3. Published guidelines for application programmers, stating standards and procedures, and enumerating steps in implementing an application

Systems Operation must provide the application programmer with failure diagnosis and recovery procedures for the following:

1. Types of failure and operator reaction

2. Terminal diagnostic program

3. Master console control of data base checkpoint

4. Master console restart procedures, including recovery of in-process messages and reconstruction of data bases

5. Master console control of IMS/360 stand-alone batch programs

6. Control of non-IMS/360 batch programs running background in the IMS/360 environment

7. IBM Field Engineering interface

8. System restart

As a part of the Systems Operation function, accounting and billing for IMS/360 batch and message programs and a background batch program in the IMS/360 environment are provided. Statistics from the system log tape processing that reflect activity by system, transaction type, terminal, line, etc. are also distributed.

Systems Operation also makes periodic reports to management and the other functions on data base activity, size, device allocation, terminal activity, line activity, transaction activity, etc. At all times, Systems Operation is ready to assist each function in achieving operational efficiency.

COMMUNICATIONS CONTROL

Communications control is a set of modules within IMS/360 that provide the service to or interface for the terminal user. Two major divisions or facilities are provided within the framework of communications control:

- Command language processing
- Message processing, including receiving, analyzing, queuing, handling, and sending

Within the above divisions are many subfunctions important to a message-oriented system.

Under the general description of command language processing are the dual functions of master terminal commands and user terminal commands. The master terminal may be considered the nerve center of IMS/360. All system conditions and many error conditions are reported to this terminal. Through the master terminal come all the decisions and commands that affect the general status of IMS/360, and through it the status of lines and terminals is controlled. The master terminal controls checkpoint and restart, which terminals are to be polled, what transaction codes are usable, whether a program is usable, the priority of specific transaction types, and the relationship between a physical and a logical terminal.

For the terminal user, communications control becomes the primary entity with which to communicate. If a terminal is available, it is polled or enabled until the user indicates a need for service. The data is then accepted and validated, stamped with time and date, logged for restart or audit, and enqueued for scheduling by IMS/360 and processing by the application program in a message region. If IMS/360 determines that the data is incorrect because of format or security, the error condition is immediately communicated to the user terminal. When the data base has been processed by the application program and a reply formulated, the reply message is enqueued by Data Language/I for communications control to transmit back to the using terminal.

Message queues are maintained in core storage as long as possible, but the primary copy is always kept on a direct access storage device. If the message is still in core storage when Data Language/I retrieves it, no disk access is required.

APPLICATION SCHEDULER

The application scheduler may be called the "resource manager" of IMS/360. The decisions to be made concern whether the necessary resources are available to process a specific message type. Two major events must occur within IMS/360 before the availability of resources can be considered:

- A complete message must be received, validated, and enqueued by communications control. The application scheduler is then notified.
- It must be ascertained that a message processing region is ready and waiting to be scheduled. The application scheduler is then notified.

When these two events have occurred, the application scheduler gains control and attempts to initiate a message processing program on a priority basis.

When the application scheduler has control, and a complete message and a message region are available, scheduling takes place. The highest priority message in the system is selected for processing. The application scheduler checks to see whether the application program is available for use by this message type. Only one copy of a message processing program will be in core storage at any one time regardless of how many message types it services. If the program is available, the scheduler determines whether all the data bases required by the message program are available. If program and data bases are available, I/O buffers for the data bases are requested. When all the resources are available, they are reserved for this message, and the IMS/360 control module located in the message region is notified to load the proper program and initiate it.

If one of the resources required is not available, the position of the message in the priority queue is maintained, and the next message type at the same priority is tested for selection.

Within IMS/360 there are 15 levels of scheduling priority (0 through 14). Level 14 is the highest level, and zero or "null" is the lowest. The null priority is a holding priority and is never scheduled. Every transaction code or message type within IMS/360 carries three numbers related to priority. First is the current priority, which indicates where the message actually is at any given instant in time; this is the number used whenever a message type is to be enqueued. The second priority number is the normal priority and is the normal source for the current priority. The third priority number is the limit priority and has associated with it a control number called "limit count". The current priority for any given transaction within the system is equal to the normal priority whenever the current number of messages of that transaction type in the input queue is less than the limit count. However, if the current queue of messages of a given transaction type equals or exceeds the limit count, the current priority is changed from the normal priority to the limit priority. When all messages of the given transaction type have been processed, thus reducing the queue length for that transaction type to zero, the current priority is restored to the normal priority.

The scheduling algorithm is based on these three priority numbers. An example of the scheduling process is as follows:

- Transaction Code = MTI
- Normal Priority = 4 (level 4)
- Limit Priority = 11 (level 11)
- Limit Count = 20

  Assume that this application requires a maximum of one hour turnaround on all messages. The minimum message rate is 25 per hour. During normal working hours message type MTI may be scheduled every 15 minutes, or more often, and most of the messages are processed each time. During peak periods when there is high activity on messages at levels 9 through 14, for example, messages at the lower levels may receive service only every two or three hours or perhaps not until the peak is over. During these peak periods, message MTI will stay at level 4 without service until the 20th message is enqueued. When the 20th message arrives, the priority of MTI is automatically boosted to level 11 by making the current priority equal to the limit priority (that is, 11). MTI will now contend for service at a priority of 11. MTI will remain at priority 11 until the enqueued message count returns to zero. MTI is then automatically restored to priority 4.

It is important to note that the master terminal also has the ability to modify the current priority. If this occurs, the message type remains at the modified priority until the enqueued message count goes to zero; it is then restored to the normal priority.

A message type held at the null normal priority will be scheduled only if the enqueued count reaches the limit count and the limit priority is not null.

Using the null normal priority, a message type can be batched and run only when a specific number of transaction types (equal to the limit count) are available.

The application scheduler is also responsible for the orderly termination of a message processing program. When the scheduler is notified that a message program has finished, all the pending data buffers are written out, and the data bases, the data base buffers, and the program are released for reuse by another message type. The application scheduler also ensures the orderly release of resources used by a message program that ABENDs. The message type and its associated program that were running at ABEND time are flagged as unscheduled. The master terminal operator must then take positive action to allow scheduling of the message type after correction of the program.

DATA LANGUAGE/I

IMS/360-Data Language/I utilizes the facilities of two Operating System/360 Data Management access methods. Basic Sequential Access Method (BSAM) has been adopted to offer the basic ability for processing data bases which have been stored sequentially on 2301 drum, 2302 disk file, 2311 disk packs, 2314 disk facility, 2321 data cell, or 2400 magnetic tapes. In addition, Indexed Sequential Access Method (ISAM) has been adopted to provide the capability of holding indexed sequential data bases on 2302 disk file, 2311 disk packs, 2314 disk facility, or 2321 data cell.

To complement the facilities of ISAM, a new access method, called Overflow Sequential Access Method (OSAM), has been implemented. OSAM was developed to facilitate the sequential addition of fixed-length physical blocks on a multivolume direct access data set, and concurrently to provide the capability to directly access and update existing blocks on the data set. The capability is used for queuing IMS/360 messages received or transmitted to communications terminals and for handling the overflow data from ISAM records under Data Language/I.

Data Language/I Major Features

IMS/360-Data Language/I has many outstanding features:

1. It provides for a common source program interface between the application programs and the data bases they reference. This interface takes the form of a CALL statement. CALL statements are provided for both PL/I and COBOL. The format of the CALL statement and the features provided are almost identical, regardless of which compiler language and data base are used. This results in a significant reduction in programmer training. It also removes from the application program any Operating System/360 data management definition.

2. IMS/360-Data Language/I also provides for the data descriptions associated with the data base to be retained as members of an Operating System/360 partitioned data set independent of application programs that use the associated data bases. This partitioned data set is known as the Data Base Description Library. By holding a data base description separate from the program, an application program is relatively independent of the organization of the data base. Thus, moderate changes in the data organization are possible without affecting the programs produced and maintained by an application programmer.

3. Prior to the advent of improved data base management, several programmers could use portions of a combined data base only if there was positive and continuous coordination between the several users. Furthermore, a change that affected any one of the users more than likely affected them all. Data Language/I offers a unique capability that allows a programmer to state which portions of a combined data base he wishes to be "sensitive" to. Within the constraint of a single logical sequence (sort order), the physical organization of a data base may be changed, or data may be added simply by modifying the programs that are sensitive to the changed elements. Of course, if changes are required for data elements common to every program, they must all be modified. However, if changes are made to those elements unique to one (or perhaps a few) of the programs, only that fraction of the programs affected need be modified.

4. At the present time, Operating System/360 Data Management allows the programmer to describe data only if the total number of characters in a logical data record is less than one physical track of the direct access storage device selected. Thus the programmer must ultimately be aware of the characteristics of the device he is using for information storage. Data Language/I allows logical data base records to span one or more physical tracks, if necessary. This provides a functional capability through Data Language/I otherwise available only through custom programming.

5. The general trend is toward combining files that share common data elements. The common elements (including the sort key) are called the root segments; the remaining data elements pertaining to individual application programs are called dependent segments. By storing the root segment only once for any logical record, requirements are reduced. The data is logically represented as a hierarchy of segments, with the root segment at the highest level. Access to lower segments is accomplished by qualification from the higher levels in the hierarchy. To allow expeditious accessing of segments, sensitivity codes were introduced. Each application program indicates through these sensitivity codes those dependent segments which it is prepared to process. Thus, it is possible to have many different segments relating to a single root segment; yet, through the mechanism of the sensitivity code, to retain simple, uncluttered application programs that relate only to a single root-dependent segment combination.

6. Another feature is that of design compatibility. The facilities provided within IMS/360-Data Language/I support two logical data structures. The Systems Operation function first describes the root segment, which consists of a field containing the highest level sort key and may contain one or more data fields always associated with the sort key and common to the root. If no further structure is provided, the data base thus created

represents the simplest case of the simple data base with one segment type.

If a more complex organization is required, the Systems Operation function describes one or more dependent segment types which are logically dependent on the root segment. If a program is sensitive to a single root-dependent segment combination, it need not be complicated with code that concerns the other dependent segments. Although the applications program will not be logically affected by segments to which it is insensitive, its execution time may be increased, since all segments are logically stored following their related root. This inefficiency when using dependent segments can be alleviated or remedied through the use of multiple secondary data set groups.

Data Language/I Rules

The following rules govern operation under Data Language/I:

1. There shall be only one root segment per data base record. This implies that there shall be only one sort key and, hence, only one sort order per data base.

2. The total length of the root segment key field or identifier in bytes shall be equal to or less than 255 bytes.

3. Each segment type may be composed of one or more fields; however, there may be only one key field within a segment type. The key fields of the segments determine the sort order of the segments within the data base.

4. The total number of the dependent segment types under a root segment must not exceed 254.

5. Each segment, be it root or dependent, must be a single fixed length. The length may vary from segment type to segment type, irrespective of segment level, but a single named segment shall have a fixed length.

6. Up to 15 levels of dependency including the root segment may be described in any single data base record.

7. A data base may consist of a single or multiple data set groups. The primary data set group contains at least the root segment. The secondary data set groups must all start with second-level dependent segments. There can be only one primary data set group within a data base. A maximum of nine secondary data set groups can be defined in a data base.

MESSAGE REGION

This discussion is not designed to give the details of the internal structure of a message region under IMS/360. However, a general understanding of the relationship of the interface involved will be beneficial to a programmer.

The existence of a message region is established by entering the Job Control Language (JCL) statements for an Operating System/360 job representing an IMS/360 message region into the input job stream. The message region is considered to be an IMS/360 type 1 processing region. The Operating System job scheduler then establishes the size of the message region and loads into the message region a small IMS/360 module called the region controller. It is the responsibility of the region controller to keep the message region under the control of IMS/360. The region controller in effect establishes an "endless" Operating System/360 job, which maintains control of the message region for an indefinite period of time. The JCL statement for the region controller may be entered into the Operating System/360 job stream as many times as desired to establish multiple message regions.

The region controller performs the following operations critical to the function of IMS/360:

1. Establishes the existence of a message region

2. Initiates interregional communication to the IMS/360 control program region for message processing program scheduling and data base requests

3. Requests initial scheduling of a message processing program into this message region

4. Establishes a maximum "time slice", before relinquishing control to the message processing program, to provide against indefinite program execution

5. ATTACHes the message processing program into the message region

6. Initiates the execution of message processing program message and data base requests through communication to the IMS/360 control program region

7. Causes the data to be moved from the IMS/360 control program region to the message program region when a message or data base request involves placement of data into application program work areas

8. Records accounting information and requests scheduling of a new message processing program into the message region after completion of an old message processing program

9. Records, at message program termination for accounting purposes, total message processing program execution time, number and type of data base requests, and number of transactions processed by a message processing program

Figure 13 shows graphically the general control flow in a message region.

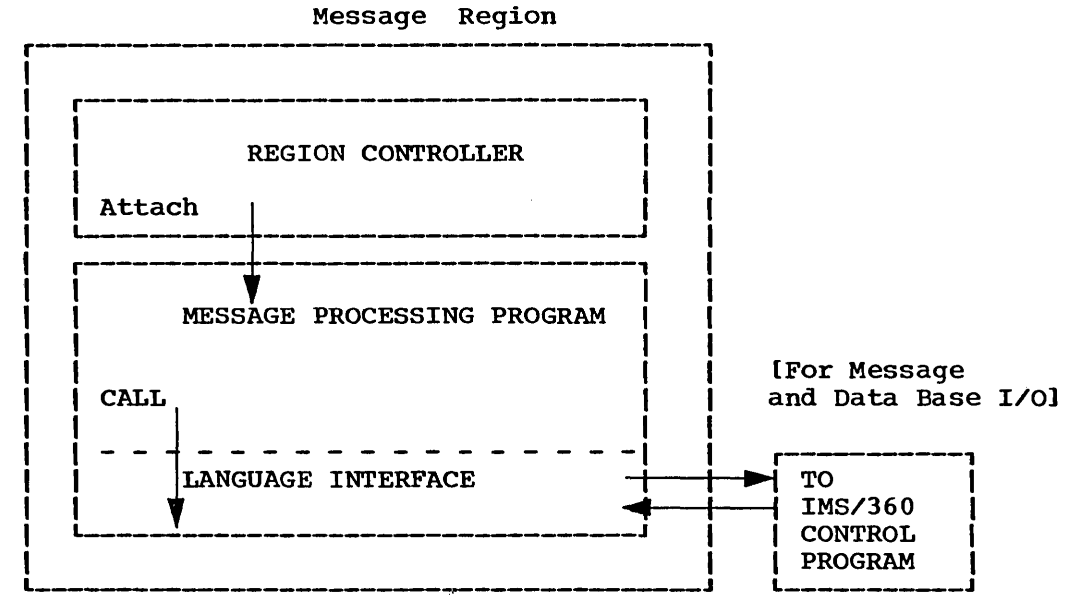


Figure 13. Organization and control flow in message region

The region controller becomes a resident module in the message region, leaving the remainder of the defined region space available for the message processing program and the language interface.

The language interface is an application program language-dependent module which is required for every programming language that may be used to program an application under IMS/360. The language interface provides the means for making every language appear the same to IMS/360. It is link-edited to the user's program and loaded as a standard part of every message processing program run under IMS/360. The language interface is discussed further in the section entitled "Program Structure".

The message processing program occupies the balance of a message region. It is this program and its logic for which the application programmer is responsible.

The region controller ATTACHes the message processing program. When a message or data base request is made by the message processing program, it is through a call to the language interface. The language interface in turn branches to the region controller to communicate with the IMS/360 control program region. All data requests made in a message region must be made through Data Language/I calls.

BATCH REGION

The concept of a batch region within the framework of IMS/360 allows some portion of a system that is not dedicated to online operations to be used for traditional batch data processing (see Figure 14). In the light of IMS/360, traditional data processing has two meanings:

1. A portion of core storage is available under the control of Operating System/360 MVT or MFT to be used for non-IMS/360 related data processing. Assuming that the batch region is sufficiently large, a compile, an assembly, or even a COBOL job may be executed. In the same manner that any job in a multiprogramming environment is not aware of another job, a non-IMS/360-Data Language/I batch job will not be aware of the existence of IMS/360.

2. In the same batch region indicated above, batch programs using IMS/360 or Data Language/I only may also be loaded. The division of this type of program into two groups is primarily due to data base handling.

   Batch programs that reference online data bases require the services of the IMS/360 control program for the purposes of resource management, security control, and data base access. These programs utilize an IMS/360 type 2 processing region.

   The other group of programs comprise those that neither require the services of IMS/360 nor reference online data bases. However, these programs use the power of Data Language/I for construction and maintenance of their own data bases. This type of batch facility is also used for the creation of online data bases. These programs utilize an IMS/360 type 3 processing region.

   In either type of batch region, standard Operating System/360 data sets may be used in addition to IMS/360 data bases.

All batch programs utilizing type 2 and 3 processing regions enter the system as Operating System/360 jobs and are scheduled and loaded by the Operating System job scheduler. Because the job is started by the Operating System, at completion the job returns control and its space to the Operating System for reuse and rescheduling by the Operating System job scheduler. As provided in the message region, all batch region execution using IMS/360 or Data Language/I is initiated through a region controller. The Operating System/360 job control language statements for the batch execution describe the particular type of region in which the batch program will be executed.

The general structure of the type 3 processing region is shown in Figure 14.

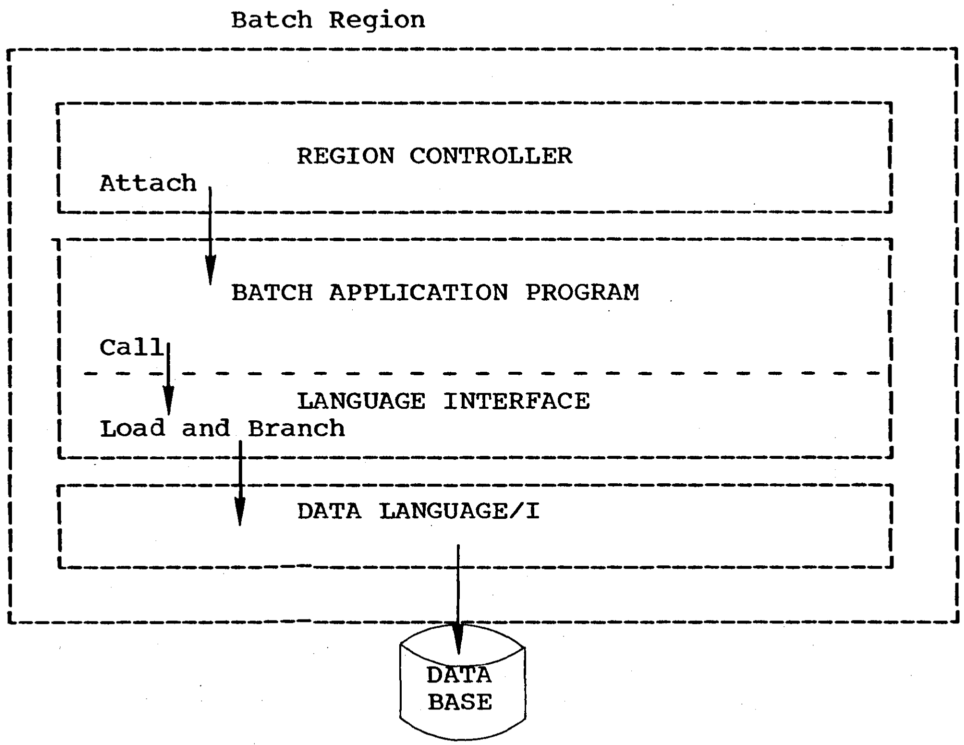


Figure 14. Organization and control flow in batch region

Except for accounting and scheduling, the region controller and the language interface provide the same function as described in the section on the message region.

In the batch environment, when the region controller determines, in conjunction with the Operating System/360 job control language statements for the job, that the program is type 3, it performs the following operations:

1. Causes Data Language/I function capability to be loaded into the region
2. Causes all necessary blocks, tables, and Data Language/I control modules to be loaded into the batch region
3. Returns control to the batch program to start processing

In addition to Data Language/I data base requests, data management input/output operations may be performed within either type of batch region.

MESSAGE PROCESSING AND MESSAGE SWITCHING

One of the earliest uses of terminal-oriented systems was for message switching. Large and complex hardware and software have been designed to handle the message switching problem alone. IMS/360 provides this capability with no additional requirement being placed upon the user of the message processing system.

The fundamental approach for any terminal user is to enter his message beginning with a transaction code; this code is the key value which gives entry to a specific message processing program. The communications control module uses the transaction code to place the message in the proper queue at the correct priority, and to ensure security. The message processing program replies by sending a message to a logical terminal name. Message switching could be accomplished in the same manner. However, the only reason for creating a message program to handle message switching would be for the instances when editing or examination was required.

IMS/360 eliminates the need to create a message switching application program. The terminal user may use the logical terminal name in place of the transaction code at the beginning of the message to be switched. Communications control recognizes logical terminal names and queues the message for immediate output, bypassing the input processing queues.

Normal Message Format

▶ (See note below)	TRANSACTION CODE	PASSWORD (Optional)	TEXT	■ (See note below)

Message Switching Format

▶	LOGICAL TERMINAL NAME	TEXT	■

Note: The input message for a 2260 is considered to be that data contained between the START MI symbol (▶) and the position of the CURSOR symbol (■) at the time the ENTER key is depressed. These two symbols are used only when the 2260 Display Station is used as the input device. The 1050 and 2740 terminals do not require these symbols. All other data displayed on the screen at this time is ignored and is not transmitted to the CPU. If no START MI symbol (▶) is displayed at the time the ENTER key is depressed, no data is sent to the CPU.

CHECKPOINT AND RESTART

The facility to checkpoint all or any portion of a large online system is vitally important. IMS/360 provides three types of checkpoints:

1. System-scheduled checkpoints, where the whole system is checkpointed on the basis of the number of messages received, or an explicit master terminal checkpoint command

2. Master terminal-requested checkpoint at normal shutdown time

3. Master terminal request to checkpoint or copy a selected data base. This procedure causes the message queues that affect a specific data base to be purged (input processed and output sent) and the data base dumped through a message processing program. A user-provided batch program is later used to copy the data base back into a direct access device if necessary.

Under controlled conditions, startup and restart are the same procedure. A normal startup is accomplished by performing a restart with the shutdown checkpoint tape of the previous period. Restart procedures are provided to handle:

- Restart after an ABEND of IMS/360 caused by a program or machine failure that did not disturb data sets, the log, or the message queues

- Restart procedure after a failure that did not allow proper data set closing

- Restart allowing for reconstruction of IMS/360 queues

- Restart after a data base is damaged or destroyed

SYSTEM RECORDING, LOGGING, AND MEASUREMENT

IMS/360 with Data Language/I is fundamentally a service system that provides high-level service to communications terminals and data bases. Since these service functions constitute the major purpose of the system, special consideration must be given to making available a record of the system activity. This record is important for purposes such as:

- Historical information

- Restart of the system

- Reconstruction of data bases

- System failure, repair, and recovery

- Audit trails

Significant numbers of persons are in direct contact with the system and dependent upon it for information necessary to the accurate and timely performance of their jobs. Quality performance is demanded of IMS/360.

Performance may be measured in terms of system availability, mean time to interruption, mean time to repair, system throughput, response time to user, simplicity of use, volume of work, and other factors. In order to determine the quality of performance of these systems, management must be informed of system activity.

For restart and statistical purposes, a module of the IMS/360 control program, the systems recorder, is provided to record all significant activity of the system.

Following is a list of the significant events that are recorded on the system log:

- Messages received from terminals
- Command messages from terminals
- Error messages received from terminals, and their causes
- Messages sent to a terminal from a message program by Data Language/I
- Error messages sent to terminals, and their causes
- Completion of transmission of a message to a terminal, with time and date stamping
- The termination of a message program and all available accounting information, including:

  Transaction code processed

  Number of messages processed

  Elapsed task time

  Environmental information
- The enqueuing and dequeuing of messages from receipt, through process, to output message generation and process termination
- Data base opens
- Data base closes
- Checkpoint records of the input and output message queues on direct access storage
- Modifications to any data base at the user's discretion

Since a different type of record is written on the log for each of the above uses, some technique must be used to identify the different records. The first byte of each logical record is called a log flag and may be used to identify that logical record. A user of the log can then look at the first byte of each logical record, process those records with which he is concerned, and bypass any records having a log flag with which he is not concerned. IMS/360-Data Language/I provides a system log utility program for analyzing system log messages.

System Measurement

System measurement information may be placed in two broad categories:

- Online statements
- Batched statistical reports

ONLINE STATEMENTS: Online statement information provides awareness of present system operations. This type of information is available through the master terminal of the system.

The following are available:

1. Current activity by terminal, displaying counts of valid messages transmitted, errors received and errors transmitted, messages queued for transmission, present operational status of the terminal, and alternate routing specification
2. Queue lengths for message types awaiting scheduling by transaction code
3. Status of message processing programs and online data bases

BATCHED STATISTICAL REPORTS: Batched statistical reports should be prepared periodically, for example, at the close of each day's operation, monthly, or as conditions justify. Batched statistics are initially processed and maintained on magnetic tape. This precludes inquiry from terminals relative to the previous day's activity and year-to-date activity. However, it is expected that the system user may use a data base so that all or some of these statistics may be maintained on direct access storage devices for online inquiry, with batch updates at completion of each day's operations.

The system log is the source of information for developing and maintaining batched statistics. The log also provides a source of input to IBM Field Engineering Systems Maintenance Management programs where Systems Maintenance Management Contracts are in effect.

To the terminal operator, the primary measure of performance is response time. The major factors that cause variation in response performance are system load and system errors. The statistical reports provide traffic volumes, error statistics, and response times; the reports provide systems management information for traffic analysis and system planning.

Priority inequities, system bottlenecks, varying transaction patterns, intermittent errors, and other conditions may be identified by analysis of these reports.

The following reports are available on a batched basis:

- By terminal and line - valid message counts by time of day
- By terminal and line - error message counts by time of day
- By transaction code - valid message counts by time of day
- By transaction code - response times for the following: shortest response time, median, 75th percentile, 95th percentile, and longest response time

TYPES OF DATA BASES

The organization of a data base is related directly to the hierarchical relationship of its segments. The segment of data is fundamental to Data Language/I and allows the structuring of any data base into either a simple or a complex hierarchical relationship. Neither the simplicity nor the complexity influences the type of access method that is used, although it may alter the desirability of one over the other. The following discussion is to assist the programmer to conceptualize his data base even though he need only consider those segments to which he is "sensitive".

The occurrence of dependent data on root information causes a dependency to exist in a data base. If the dependent information can be stratified, or if a dependent segment has segments dependent upon it, the file begins to assume a hierarchical relationship. The term "levels of information" is introduced to describe how far removed from the root segment a dependent segment is. The root segment is defined as containing the key and level-one information. The first dependent segment carries level-two information. A dependent segment which was in turn dependent upon the level-two dependent segment would be called a level-three segment, etc. Data Language/I allows 15 of these levels to be defined, along with many dependent segment types within each of those levels. However, a table has been set aside with 255 entries in it. The root segment takes up one entry position; therefore, there may not be more than 254 other segment types in the entire data base.

Simple Hierarchical Files

The simplest of all files consists of only a single segment type. In Data Language/I terminology, this file consists solely of root segments and no dependent segments. Each segment in the file has a fixed-length key field and one or more fixed-length data fields accompanying that key. All of the fields are always present. The file is stored and retained in an order based on the values of the keys when taken as simple binary values. Further more, the simple file has fixed definitons for each field.

The simple file is the most common file in existence and in fact requires no hierarchical consideration. Even though the simple data base lacks the true hierarchical structure, it may be handled by Data Language/I and as such becomes the simplest form. This file structure produces fixed-length root segments within logical data management records, and these logical data management records are collected to form physical records.

Complex Hierarchical Files

The first instance of data file complexity usually occurs when control totals are appended to a simple file. If the simple file described an inventory of parts, each with its part number (key), a fixed-length alphameric description, a fixed-length quantity-on-hand field, and a fixed-length unit price field, it might be necessary to insert some total records indicating the total dollar volume of a series of parts in a specific category and the total quantity on hand independent of part number. This would be done in one of two ways, depending on the type of storage media used to retain the files (see Figure 15).

PART NUMBER KEY	DESCRIPTION	QUANTITY-ON-HAND	UNIT PRICE

Figure 15. Simple physical file layout

If the file were retained on tape, it would have to be rewritten in its entirety whenever a single inventory item was used to fill an order. Given this situation, it is quite logical to embed the control totals at the appropriate point following the sequence of data they summarized. Thus, the detail information for a category would be read, and, after all the transactions to the category had been processed, the control totals for that category could also be processed, updated, and written out on the new tape drive. This is both traditional and practical, since a spare tape drive to hold the total separately is expensive, and since the entire tape must be rewritten during every processing cycle anyway. Therefore, it makes no sense to build a second simple file which would require a separate drive of its own just for the control totals.

Alternatively, a one-character field segment type could be added to each entry and appended to the least significant end of the key. All of the detail data pertaining to a single type of line item in the inventory would be awarded one segment-type code, say the numeric value 1. The control totals would assume the key of the highest line item they summarized and have a record-type code field equal to any number greater than 1. Thus, if the file is sorted or sequence-checked on the augmented key, it is in fact in numeric sequence on that key, even though the control totals may be embedded in the data. After a record has been read from the device and delivered to core storage, a simple test based on the type code field could allow the program to process detailed data or know that it was dealing with a summary segment (see Figure 16).

LAST PART NUMBER OF A PARTICULAR CLASS

Last Part Number Key	01	Description	Qty on Hand	Unit Price

Record Type Code

CLASS TOTALS RECORD

Last Part Number Key	02	$VOL Qty

Record Type Code

Figure 16. Complex physical file layout

The "total record" is a simple case of the occurrence of a segment-type code field. Another instance of the segment-type code field will occur should two similar files be merged to allow common information to be held once and information unique to two different purposes to be subservient to it. Frequently, files occur that have many fields in common. If these files also enjoy a common key and common sort order, they can be combined. The common information (root segment) is held once, and the unique information (dependent segments) can be held in a compact form, since the key and related fields need be specified only once for each pair of records.

When several files have been combined to gain improved storage utilization or ease of processing, a second-order effect occurs. Not every programmer requires access or is even authorized to access every different dependent segment type. To solve this problem and eliminate the superfluous information (in the eyes of a programmer) that he does not need to handle, sensitivity codes are introduced. Each segment type is assigned a unique name, and Data Language/I allows a programmer to state the names of the segments in the data base he wishes to "see", using an area within the Program Specification Block (PSB) called the Program Communication Block (PCB). At execution time, the Data Base Description (DBD) relates those segment names to the numeric segment type codes stored with the data. A block of data is then read into core storage by Data Language/I. While Data Language/I has control, the segment-type code fields embedded in the segments of data just read are compared against the type codes of the segments to which the PSB has declared it is sensitive. If a block of data is obtained that contains no segment whose type codes match the programmer's sensitivity list, another read is initiated.

Internally, at execution time, Data Language/I keeps an identification table of segment names, type codes, and levels. Data Language/I returns status codes, level numbers, segment names, and segment keys to the program through the PCB to indicate the relationship to the hierarchy of the segment just obtained. This relationship is obtained by entering into the identification table the name of the segment just retrieved and comparing its level and type with the level and type previously obtained.

Data bases can thus be constructed from combined complex files made up of several different segment types, all of which contain limited header and control information and a minimum amount of redundant data. It should be noted that Data Language/I reads and sometimes rewrites segments to which the programmer is not sensitive.

STRUCTURE OF DATA BASES

The application program's independence from the access methods, physical storage organization, and characteristics of the devices on which the data is stored is provided through a common source program linkage (consisting of a list of parameters that are addresses of PCB's, I/O functions, and segment identifiers) and a data base description. This common source program linkage and data base description allow the application program the ability to request Data Language/I to:

- Reference a unique segment (GET UNIQUE)
- Retrieve the next sequential segment (GET NEXT)
- Replace the data in an existing segment (REPLACE)
- Delete the data in an existing segment (DELETE)
- Insert a new segment (INSERT)

Note: "Segment" refers to a fixed-length data element containing one or more logically related data fields.

The above calls are described in a later section of this manual.

In the COBOL language, this common source program linkage uses the ENTER LINKAGE and the CALL verb to perform the input/output functions listed above. Application programs written in PL/I or Assembler Language use similar statements to reference Data Language/I. Because of this approach to data reference, input/output operations and associated control blocks are not compiled into the application program. This removes dependency upon the currently available access methods and physical storage organizations.

Each data base description is created from user-provided statements of the logical and physical structure of each data base. These statements are input to an offline utility program of IMS/360. The result of the utility program is the creation and storage of a Data Base Description in the user-defined Data Base Description library. This Data Base Description provides Data Language/I with a "mapping" from the logical structure of the data base used in the application program, to the physical organization of the data used by Operating System/360 data management. The logical data structure can be "remapped" into a different physical organization without the necessity for program modification. Integration of other application data can also be added to this data base and still not cause a change to the original application programs. The concept of a Data Base Description reduces application program maintenance caused by changes in the data requirements of the application.

Data Language/I provides for elimination of redundant data while providing integration or sharing of common data. The majority of the data utilized by any company has many interrelationships and hence many redundancies. For example, Manufacturing and Engineering have many pieces of data which would be useful to Quality Control; so do Purchasing and Accounting. If analysis of the number of types of segments shows that all the data cannot be placed in a single common data base, Data Language/I allows the user the additional capability of physically structuring the data over more than one data base. Before Data Language/I, personnel responsible for application programs frequently were not able, nor did they have the time, to integrate other data with their own to eliminate redundancies without the necessity of a major rewrite of the application programs involved.

Another capability of Data Language/I protects each application of a multiapplication data base through the concept of "sensitive" segments. When operating against a Data Language/I data base, only the data segments that are predefined as sensitive are available for use in this application. Each application using the data base can be sensitive to its unique subset of "sensitive" segments. Where an application program has defined "sensitivity" to a subset of segments within a data base record, modification and addition of nonsensitive segments do not affect the processing capability of the program. In addition, any application program can be restricted to "read only" operations against its sensitive segments.

Data Language/I - Data Base Organization

The data base structure is best described by providing an example. Figure 17 depicts the hierarchical relationship for a company data base made up of engineering data, inventory data, and purchasing data, which could be typical of any company. All this is based on part master (part number) data.

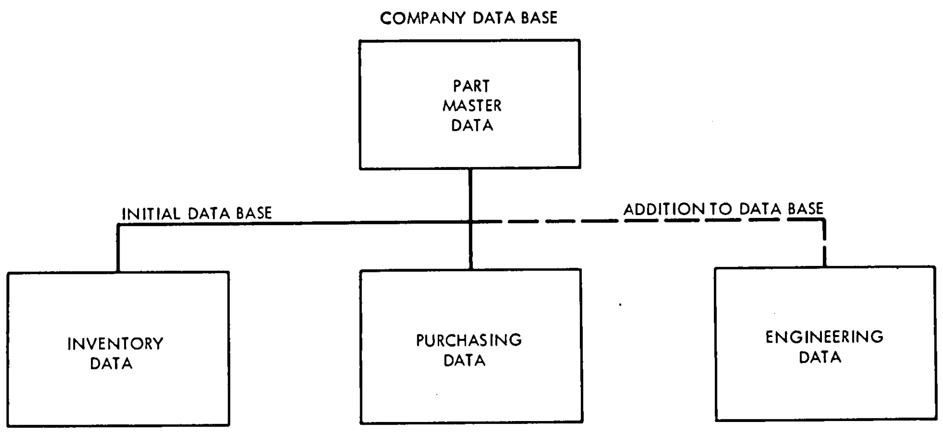


Figure 17. Company data base hierarchical data relationship

A data base is composed of data base records. A data base record is a collection (a variable number) of hierarchically related, fixed-length data elements, called "segments." A root segment is the highest hierarchical segment in the data base record. A dependent segment is a segment that relies on at least the root segment for its full hierarchical meaning. It is therefore always at a lower hierarchical level than the root segment. There can be 255 segment types within a data base and 15 levels of segment hierarchy within a data base record.

Details of the segment of this data base example for the inventory and purchasing data contained in the initial company data base segment structure are shown in Figures 18 and 19. This logical structure may be physically stored in either of Data Language/I's organizations:

- Hierarchical sequential: The Operating System/360 Basic Sequential Access Method (BSAM) is used to implement the hierarchical sequential organization. Storage medium may be tape or direct access storage. See Figure 20.

- Hierarchical indexed sequential: The Operating System/360 Indexed Sequential Access Method (ISAM) and a unique access method of IMS/360, called Overflow Sequential Access Method (OSAM), are used to enhance the capabilities of ISAM and to implement the hierarchical indexed sequential organization. The storage medium must be direct access storage. See Figure 21.

After the initial data base details shown in Figures 18 through 22, the addition of engineering data in Figures 23 through 27 may be accomplished. As illustrated, the data base segments may be organized or reorganized into the hierarchical indexed sequential organization or the hierarchical sequential organization. Note that, even with the addition of engineering data, the expansion of the data base may be accomplished without altering the existing processing programs that reference the data base.

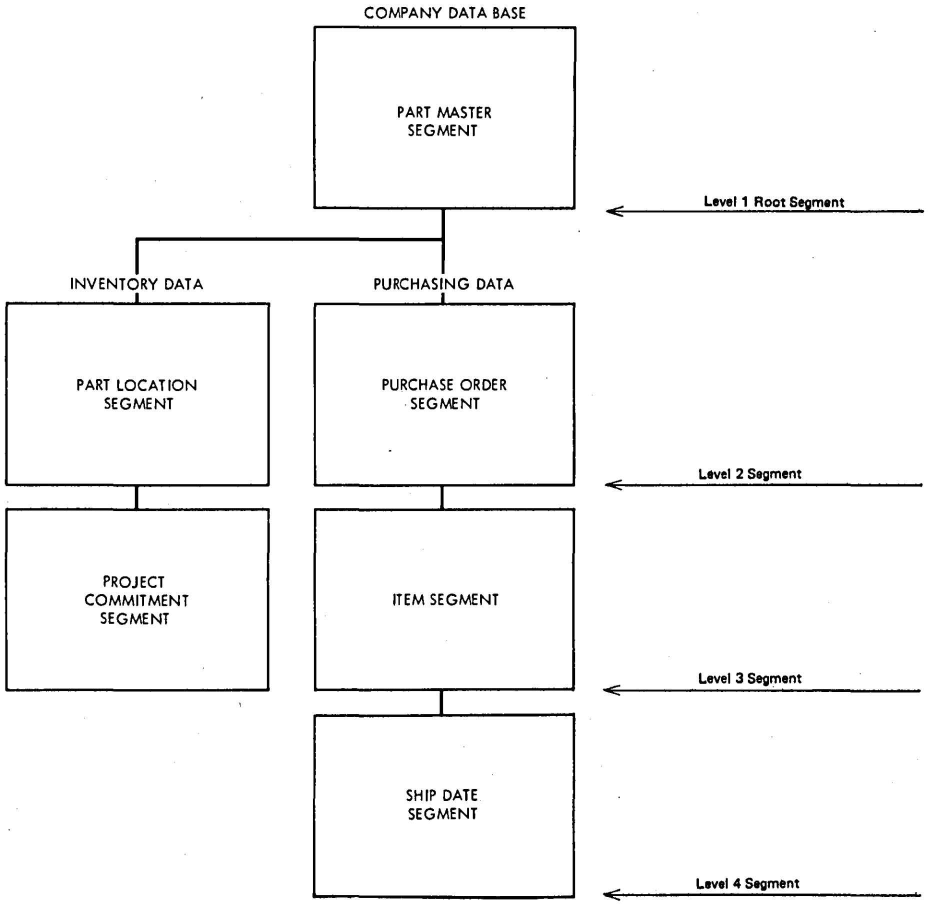


Figure 18. Company data base segment logical hierarchical relationship -- inventory and purchasing data

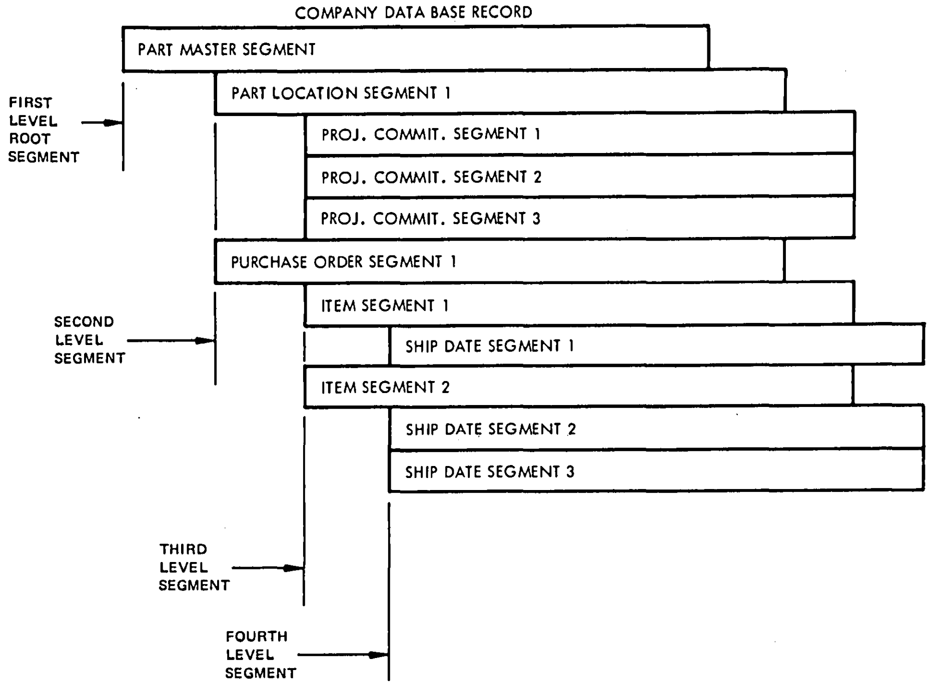


Figure 19. Company data base record segment level structure

The highest level (level one) segment or root segment is the part master segment. All segments immediately subordinate to the root segment are called second-level dependent segments: part location segment and purchase order segment. Third level dependent segments are related to the second-level dependent segments. In this example, project commitment segment 1 is related to part location segment 1, and item segment 1 is related to purchase order segment 1. Fourth-level dependent segments are related to the second-level dependents. All the segments in Figure 19 constitute a data base record.

If the hierarchical sequential organization is chosen for the data base, (Figure 20), each segment type is fixed-length within a data base record and is stored in physical sequence according to its hierarchical relationship.

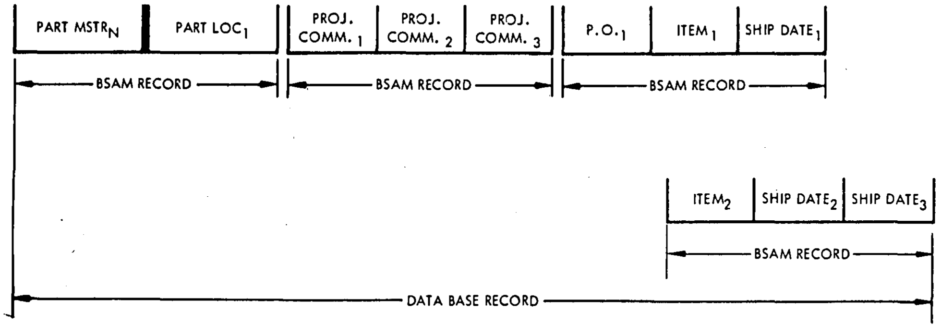


Figure 20. The Nth data base record stored in hierarchical sequential organization

Figure 20 represents the Nth data base record depicted by the data base in Figure 18. The part master root segment has one occurrence of the second-level dependent part location segment type. For inventory purposes, this part has only one storage location. The second-level part location segment type has three occurrences of the third level project commitment segment type. There are three projects in this company that have commitments against the inventory of this part.

There is one second-level purchase order segment type, the root segment of which is the Part Master. This second-level segment type has two occurrences of the third-level dependent segment type: item segment 1 and item segment 2. They in turn have subordinate or fourth-level dependent segment types: ship date segments. The item segments are the purchase components for this particular part (or assembly). Each has a particular shipping date.

All data base records are stored sequentially in sort sequence of the root segments. The only direct data reference provided with the hierarchical sequential organization is to the first root segment in the first data record of the data base. All subsequent reference is sequential.

If the hierarchical indexed sequential organization is chosen, direct reference is provided to each root segment (and therefore to each data base record) within a data base. When the data base is created or reorganized, the key of each root segment is an ISAM logical record key. As many segments (the root and its dependents) are stored as will fit within the ISAM logical record. If storage for additional segments within the data base record is required, a relative block pointer is placed in the ISAM logical record. This pointer relates the ISAM record to one or more OSAM records which contain the remaining segments of the data base record (Figure 21).

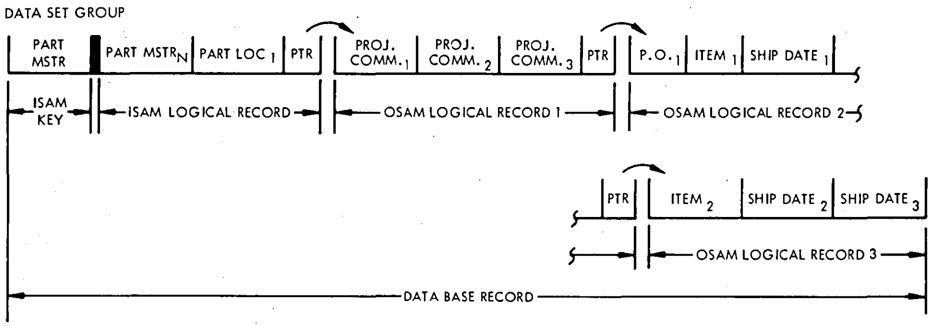


Figure 21. The Nth data base record in hierarchical indexed sequential organization

When the data base is created or reorganized, each data base record starts as an ISAM logical record and may overflow into one or more OSAM logical records. Note in Figure 21 that the two data sets, ISAM and OSAM, represent a data set group. Reference to segments within the data base record is sequential.

As shown in Figure 21, the ISAM logical record consists of two segments: the part master root segment and the second-level part location dependent segment. At the end of the ISAM logical record is a pointer to the first OSAM record. The first OSAM logical record consists of three second-level project commitment dependent segments 1, 2, and 3. The second and third OSAM logical records contain the remainder of the data base record.

An additional capability of the hierarchical indexed sequential organization is to provide direct access to all root segments and to all or some first-level dependent segment types. This capability is provided through the use of multiple ISAM and OSAM data sets (multiple data set groups). (Figure 22).

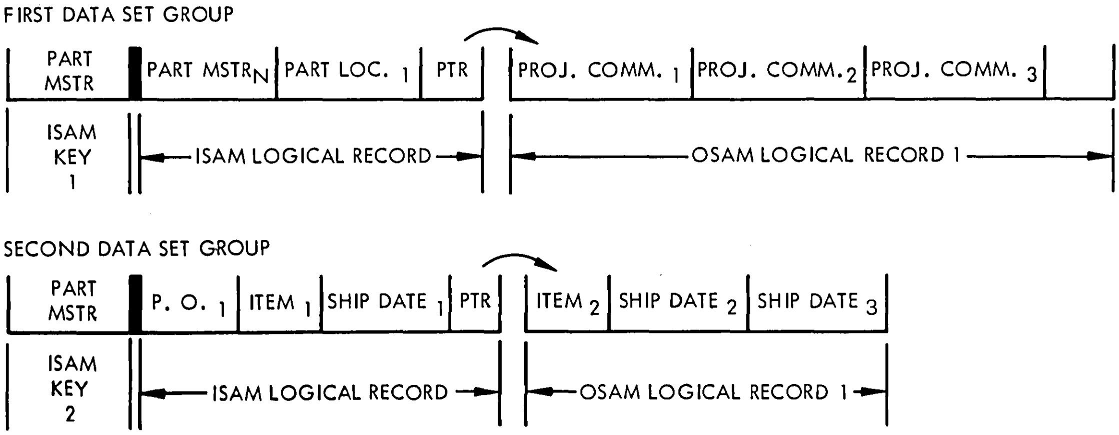


Figure 22. The Nth data base record in hierarchical indexed sequential organization -- multiple data set groups

The part master root segment, the part location dependent segment 1, and the project commitment dependent segments 1, 2, and 3 are contained within one data set group. The purchase order second-level dependent segment type and its remaining dependent segments are contained within a second data set group. This allows direct reference to the first purchase order segment type within each data base record as well as to the root segment, part master. A maximum of ten data set groups is permitted in the hierarchical indexed sequential organization. There can be only one primary data set group and a maximum of nine secondary data set groups for any one data base.

An example is now depicted which illustrates an addition to the company data base of engineering data. It is assumed that the inventory and purchasing applications are not changing. The addition to the company data base is the integration of the engineering application as shown in Figure 23.

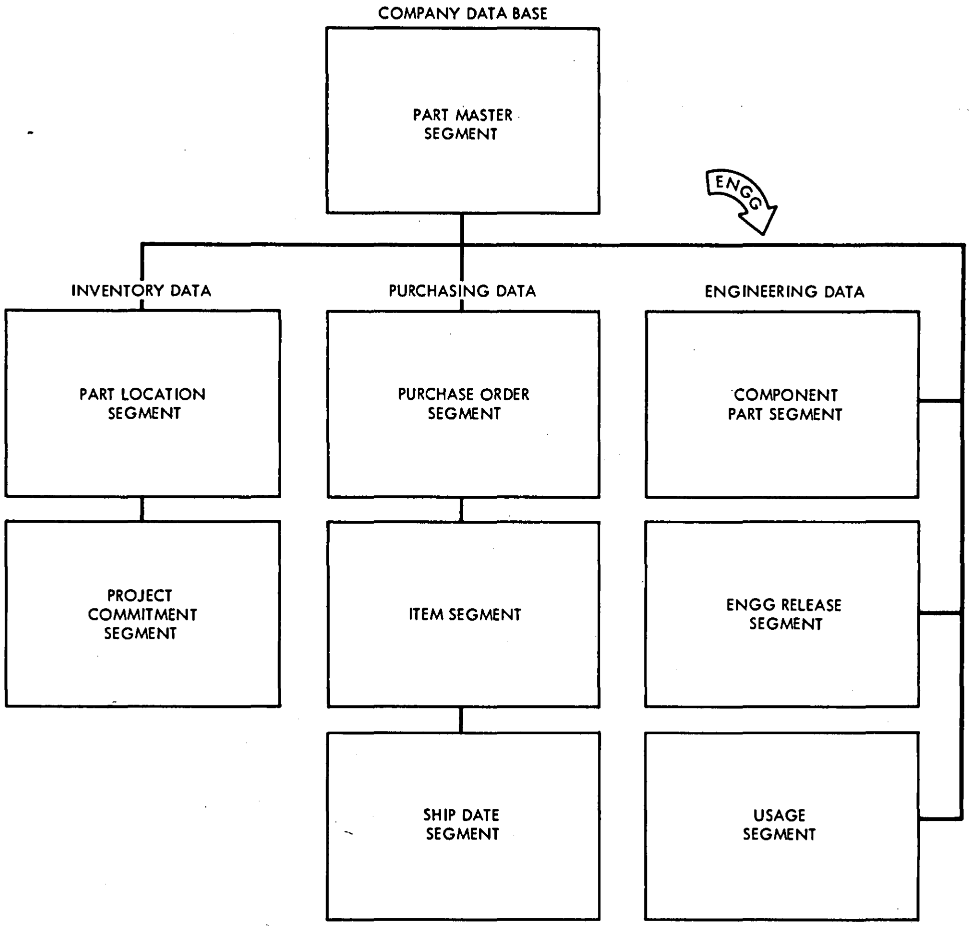


Figure 23. Company data base segment logical hierarchical relationship -- inventory and purchasing data -- engineering data added

The responsible user personnel may extend the company data base description and insert the engineering data segment structure. Then the existing company data base and the engineering data segments are merged to create the new company data base. Figure 24 depicts a segment-level picture of the data base record.

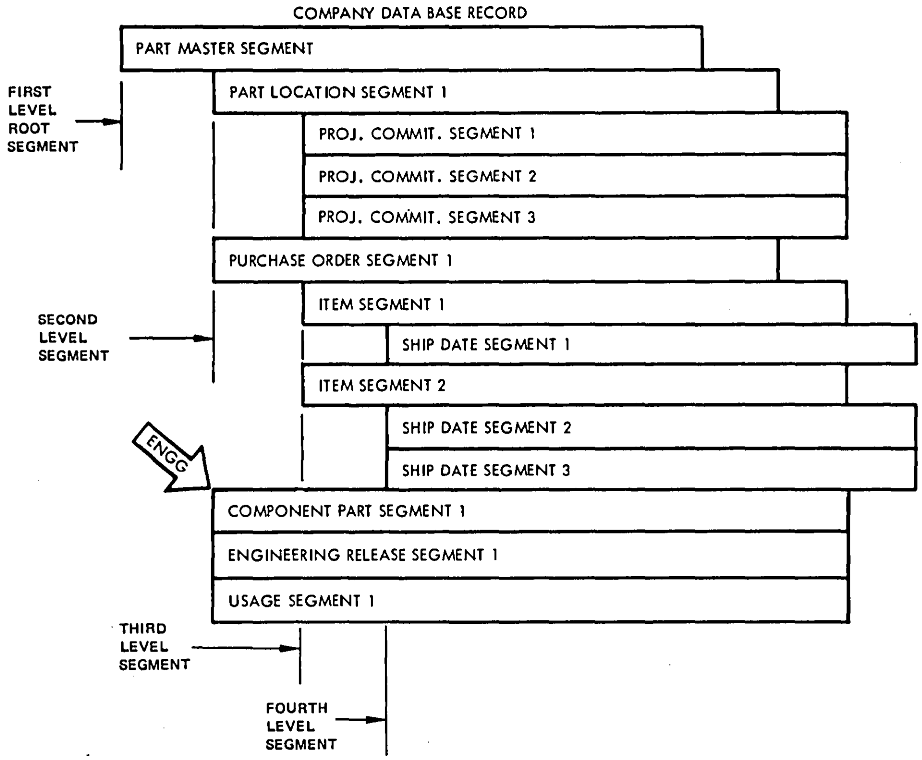


Figure 24. Company data base record segment level structure -- engineering data added

Figures 25, 26, and 27 illustrate how the new data base may be physically stored. Note that in Figure 27 a third data set group is added without disturbing either the inventory or purchasing application data.

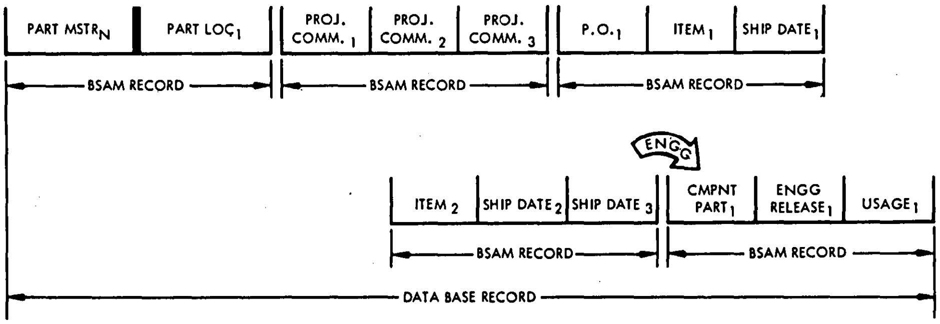
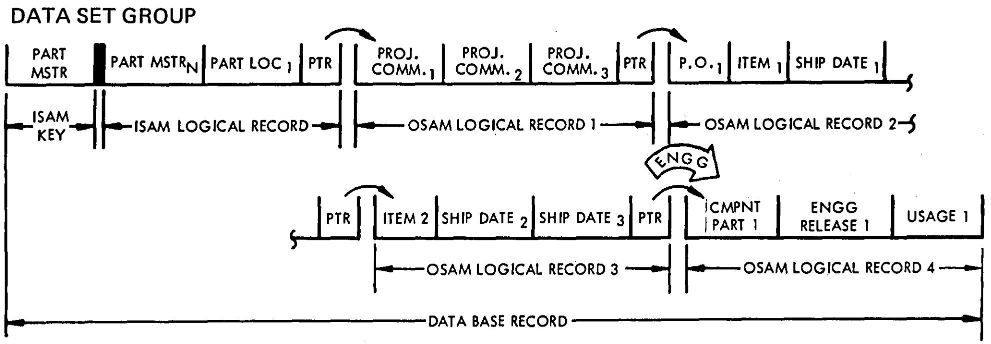
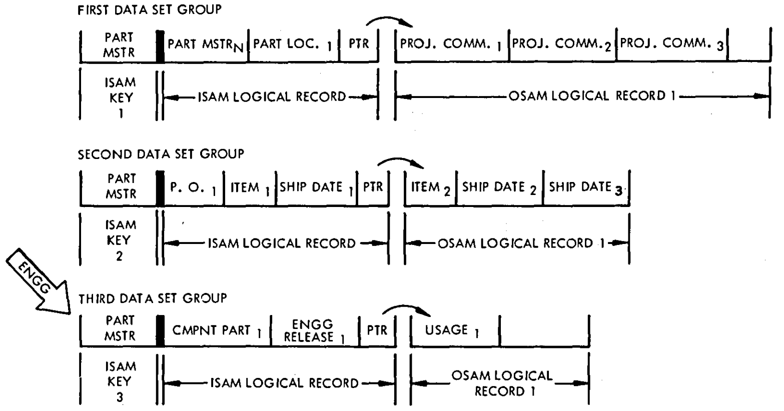


Figure 25. The Nth company data base record stored in hierarchical sequential organization -- engineering data added



• Figure 26. The Nth data base record in hierarchical indexed sequential organization -- single data set groups -- engineering data added



• Figure 27. The Nth data base record in hierarchical indexed sequential organization -- multiple data set groups -- engineering data added

DATA BASE PROCESSING

Data base processing with Data Language/I is accomplished with the data storage organizations just described and a set of input/output functional requests used by application programs.

An input/output functional request is composed of a CALL statement with a parameter list. The parameter list provides the information, which is assembled by the application program, to describe a particular input/output function and the element of data operated upon. The element of data operated upon by any Data Language/I input/output request is termed a segment. One and only one segment may be operated upon with a single input/output request or call.

A segment is composed of one or more data fields, one of which is considered the key field. Each particular segment type has a fixed length and format definition.

The parameters contained within any input/output functional request include the addresses of:

- The input/output function

- The definition of the data base to be operated upon

- The segment input/output area into or out of which the segment of data is moved

- The identifiers used to describe the segment of data to be operated upon

The input/output functions provided by Data Language/I are GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT SEGMENT, DELETE, REPLACE, and INSERT. Remember that each of these functions, within a single request, operates upon only one segment of data. Thus, GET UNIQUE causes the retrieval of a specific segment described by the identifiers in the CALL statement into the defined segment input/output area. The INSERT operation causes the segment residing in the segment input/output area and described by the segment identifiers to be added to a data base. The identifiers in a functional request used to describe the segment of data to be operated upon are called segment search arguments (SSA's). A segment search argument includes the one- to eight-character symbolic name of the segment type, the one- to eight-character symbolic name of the segment key field, an algebraic operator, and the value of the desired key field. Consider Figure 24. The generic name of the part master segment must be PARTMAST, and its key field name might be PARTNUMB (part number). Then, the segment search argument for GET UNIQUE of the part master segment with part number equal to 12345 would appear as:

PARTMAST(PARTNUMB =12345)

The portion of the segment search argument within the parentheses is called a qualification statement.

For unique retrieval or addition of a root segment, only one segment search argument must be provided. However, the unique retrieval or insert of a dependent segment requires multiple segment search arguments to be provided in the functional request. Each segment search argument in the list describes a segment to which the dependent segment to be operated upon is dependent. The SSA's for a given Data Language/I call must be in proper hierarchical relationship. If the generic name of a purchase order segment type is PURCHASE, its key field name is PURCHNO, and there is a purchase number XYZ for part 12345; unique retrieval is accomplished by two segment search arguments included within the parameter list of the Data Language/I call:

PARTMAST(PARTNUMB =12345)

PURCHASE(PURCHNO =XYZ)

The definition of the data base to be operated upon is provided in each Data Language/I call by a control block called a Program Communication Block (PCB). All PCB's used by a particular application program for data base operations are contained within the PSB for that program. At execution time, the base addresses of the PCB's are passed to the application program. Each PCB contains the one- to eight-character name of the DBD associated with the data base.

Data Base Creation

A data base is created by an application program issuing Data Language/I calls to insert data base records presorted by the key field of the root segment. When a data base record is composed of more than the root segment, all segments within the data base record must be presorted by their hierarchical relationship and key field value.

Consider the process of inserting the segments of a company data base record described in Figure 24. First, the part master (root) segment is inserted. The part location segment (first-level dependent) is inserted next. Then the three project commitment segments sorted by key field value are inserted. This continues with the purchase order segment, item segment 1, ship date segment 1, item segment 2, etc., until all segments are inserted. If this data base record represented the segments of data associated with part number X, the segments to be inserted into the data base next would be those associated with part number X + 1.

The INSERT function is used to create or load (recreate or reorganize) a data base. Prior to the execution of a Data Language/I call to cause segment insertion, the segment to be inserted must be moved into a segment input/output area, and the proper list of segment search arguments must be assembled. Assume that, in creating the company data base, the segments of data associated with part number 12345 are to be loaded. The first three segments to be loaded are part master, part location 1, and project commitment 1. The associated segment search arguments and input/output work area contents for these three Data Language/I INSERT calls are:

PART MASTER SEGMENT INSERTION

SSA$_1$ - PARTMAST

Work Area (containing part master segment)

```
|           |           |           |
| Key Field | Data Field| Data Field|
L-----------------------------------J
Key =12345
```

PART LOCATION 1 SEGMENT INSERTION

SSA$_1$ - PARTMAST(PARTNUMB =12345)

SSA$_2$ - PARTLOC

Work Area (containing part location segment)

```
|           |           |
| Key Field | Data Field|
L-----------------------J
Key =456
```

PROJECT COMMITMENT 1 SEGMENT INSERTION

```
    SSA -  PARTMAST(PARTNUMB =12345)
       1

    SSA -  PARTLOC (LOCATION =456)
       2

    SSA -  COMMIT
       3

    Work Area (containing project commitment segment)

           |           |             |            |
           | Key Field | Data Field| Data Field|
           L------------------------------------J
           Key =6185
```

Notice that the segment search arguments of a Data Language/I call for inserting a segment into a data base must describe the complete hierarchical path to the segment. Also notice that the last segment search argument within each INSERT call does not and must not include the qualification statement portion. The qualification information is taken from the image of the segment in the input/output work area.

All data base creation and reorganization must be performed in a batch processing region of IMS/360.

Data Base Retrievals

The retrieval of segments within a data base is accomplished by the three GET functions: GET UNIQUE, GET NEXT, and GET NEXT WITHIN PARENT SEGMENT. GET UNIQUE provides for the retrieval of a specific segment by direct reference into the data base. GET NEXT provides for sequential segment retrieval. Usually the GET NEXT function is used after a GET UNIQUE or GET NEXT that has provided "positioning" to a unique segment within the data base. However, a GET NEXT may be used without positioning being supplied by a previous GET UNIQUE or GET NEXT. If Data Language/I has no position established within a data base when a GET NEXT call is issued, the request is satisfied by proceeding from the beginning of the data base. The GET NEXT WITHIN PARENT SEGMENT allows sequential retrieval of all segments subordinate to a parent segment. An example, using Figure 24, is the retrieval of all item and ship date segments within the company data base for a given part and purchase order. The parent segment is a unique purchase order segment, and parentage must have been previously established with a GET UNIQUE or GET NEXT request.

Once all the item and ship date segments for a given part and purchase order have been retrieved by a succession of GET NEXT WITHIN PARENT requests, an indication is returned to the application program. This indication provides definition of the end of subordinate segments for the particular part and purchase order.

In addition to direct retrieval of a unique segment and sequential retrieval of segments, an ability to skip sequentially from one segment to another of a common type is provided. Assume that it becomes necessary to retrieve all purchase order segments within a particular part master segment. However, it is not necessary to retrieve the segments subordinate to each purchase order segment (that is, item and ship date segments). The first purchase order segment would be retrieved with a call where the function equals GET UNIQUE. The segment search arguments would be:

SSA_1 - PARTMAST(PARTNUMB =12345)

SSA_2 - PURCHASE

The remainder of the purchase order segments would be retrieved with Data Language/I calls where the I/O function parameter equaled GET NEXT. However, the Data Language/I calls would have a segment search argument:

SSA_1 - PURCHASE

In summary, the segment search arguments for all GET UNIQUE calls must start with reference to the root segment level. The GET NEXT calls may be used with or without segment search arguments. The segment search arguments for a GET NEXT call may start at any segment level. All SSA's within a single Data Language/I call must be in proper hierarchical order (that is, SSA for root first, SSA for first-level dependent second, etc.).

Data Base Updates

The updating of data within a segment of a data base is performed through the REPLACE input/output function. Before a Data Language/I call to replace a segment may be executed, the segment to be updated must be retrieved through a CALL statement with a GET function. The GET functions that may be specified are those previously discussed, but must include the addition of a HOLD definition (GET HOLD UNIQUE, GET HOLD NEXT, and GET HOLD NEXT WITHIN PARENT). The REPLACE function must then be executed in the next call by this program against the data base. Any intervening calls against the same data base by this program cause the rejection of the subsequent REPLACE call. No SSA's are permitted with the REPLACE function. The key field of the segment to be updated through the REPLACE function call must not be modified.

Data Base Deletions

The deletion of an entire segment (all fields) within a data base is performed through the DELETE input/output function. Before a Data Language/I call to delete a segment may be executed, the segment to be deleted must be retrieved through a GET HOLD call. The DELETE function must be executed as the next call against the data base; otherwise, the DELETE function is rejected. No SSA's are permitted with the DELETE function. The deletion of a parent segment causes deletion of all segments subordinate to the deleted segment.

Data Base Insertions

The addition or insertion of a new segment (all fields) into an existing data base is performed through the INSERT input/output function. The techniques used for performing an INSERT function to add a segment to an existing data base are identical to those used with the INSERT function when creating a new data base. Remember that the addition of a dependent-level segment is not permitted unless all parent segments in the complete hierarchical path already exist in the data base. An example, referring to Figure 24 is the addition of an item segment subordinate to a particular purchase order segment. The purchase order segment must already exist in the data base or be added before any item segments subordinate to that purchase order segment may be added.

Message Input/Output Calls

The messages received from terminals and placed in the message queues are accessible to a message program by Data Language/I calls. The first line of a message is obtained with a call with a function equal to GET UNIQUE. Each subsequent line of the message is obtained by a call with a GET NEXT function. A message program may normally wish to place output messages in the message queues for subsequent transmission to terminals. The output messages may be enqueued for response to the terminal that was the source of the input message, or to one or more output terminals. Alternate output terminals (other than source of input) must be known (predefined) to the message processing program. A Data Language/I call with the function parameter equal to INSERT is used to enqueue for output a line of an output message. Each line of the message enqueued through the INSERT call must be terminated with a carriage return character in the text. If a message processing program is written in a serially-reusable manner, multiple input messages may be processed serially with one load of the program. The first line of each input message is obtained with a GET UNIQUE call.

Program Specification Block (PSB)

Associated with every message or batch processing program is a Program Specification Block (PSB). This control block describes the use of data bases and terminals (if message program) by the associated application program. The PSB is constructed by the application programmer and placed in a partitioned data set generically termed the PSB library. A utility program is supplied to assist in the generation of each PSB. Each PSB is composed of one or more sub-blocks called Program Communication Blocks (PCB's). One PCB exists for each data base and each alternate output terminal with which the associated message or batch program intends to interface. Each data base PCB describes the segments to which the associated message program is sensitive, and the mode of processing that the associated message program will utilize on the data base. Processing modes include data base creation, retrieval, deletion, update, and addition. The PCB also includes the symbolic name of the data base with which it is associated.

Each alternate output terminal PCB is associated with a logical output terminal. In addition to the alternate output terminal PCB's in the PSB, an input/output terminal PCB is associated with the source terminal of the input message. However, the input/output terminal PCB is not embedded within the PSB. The PSB and its PCB's exist external to its associated message or batch program. However, these blocks are used by the program in executing Data Language/I calls. The addresses to these blocks are passed to the associated message or batch program upon entry to the program. The address of a PCB associated with a given data base or logical terminal is subsequently used by the program when issuing a Data Language/I call. The PCB address becomes a parameter in the Data Language/I call.

Data Base Segment Sensitivity

The preceding paragraph described the use of PCB's to enable an application program to execute Data Language/I calls. The PCB contains the one- to eight-character name of the associated data base. The reader must recognize that the PCB supplies to Data Language/I the logical definition of the data base upon which the requested input/output operation is to be performed (the data base name). The PCB also describes the processing mode that the associated application processing program intends to use upon the data base. However, the most important elements of data that the PCB supplies to Data Language/I are the names of the segments of data within the data base upon which the application processing program intends to operate. These represent the segments of data to which this application processing program is sensitive. Only the segments of data named in the PCB may be retrieved, updated, deleted, or added to the data base.

This concept of segment sensitivity has considerable importance with regard to the impact that the addition of new segment types into an existing data base has upon existing application programs. A data base can be expanded with new segment types by the dumping of the data base, the generation of a new data base description incorporating the new segment types and the old, and the reloading of the data base with the new data base description. The new data base now contains the old and the new segment types. The existing application programs, which operated upon the old segments in the data base, are sensitive only to the old segment types. The new segment-types appear "invisible" to the existing application processing programs. No modification is required of the existing application processing programs to operate upon the expanded data base. Of course the existing application programs, since they are insensitive to the new segment types, cannot operate upon the new segment types. Presumably, new application processing programs would be incorporated to operate with the existing programs to maintain the new segment types. A significant benefit of the concept of sensitivity is the evolutionary expansion of the data contained within a data base with minimal impact upon existing application processing programs.

Data Base Segment Definition

Each segment type within a data base is defined at Data Base Description generation. The characteristics of the segment -- length, fields, key field, etc. -- are defined. It may often be a considerable task to determine the best structure of hierarchically related segments to use in defining applications data. Several guidelines are suggested:

1. Each Data Language/I data base record has one root segment. The key field of the root segment is the primary sort key of the data base.

2. The structure (fields) within a root segment should represent data that occurs once per data base record.

3. Any dependent segment may occur zero to n times per root segment. The data within a dependent segment type should occur zero to n times for one occurrence of the root segment data. Each dependent segment type represents a lower level sequence of data.

4. Although a data base has only one sort sequence, other sort sequences or cross-reference relationships may be required. These can be accomplished with other data bases of other sort sequences.

Types of IMS/360 Processing Regions

Three types of processing regions are available to the IMS/360 user. Type 1 is used for message processing. Type 2 is used for batch processing, with Data Language/I data bases concurrently used for message processing. Type 3 is used for batch processing with Data Language/I data bases that are unrelated or not used concurrently for message processing. Within the capabilities of Operating System/360, one or more processing regions of various types may be operating concurrently.

A message processing region is used exclusively for message processing. Each message processing region may reference the input and output message queues and data bases used for message processing. The only input/output operations permitted in a message region are Data Language/I calls to the IMS/360 control program region for messages and online data base segments. No other input/output operations are permitted within the message region. A message region may output messages destined for terminals, for input to other (or the same) message programs, for input to an HSAM output file, or for input to a message queue that will subsequently be accessed by a batch program in a Type 2 processing region (Figure 28).

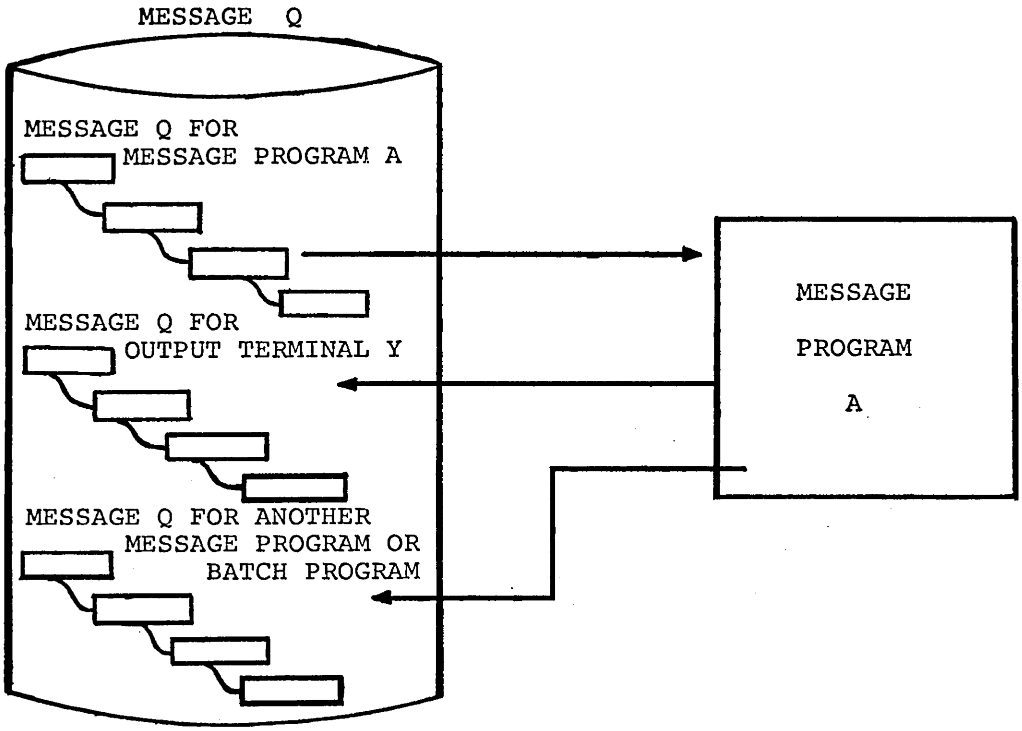


Figure 28. Message region/message queue relationship

All message processing programs must be predefined to the IMS/360 control program as described in the <u>IMS/360 Operations Manual, Volume I - Systems Operation</u>. Operating System/360 subtasking may be used in the design of an application program providing that:

1. A single copy of the COBOL - PL/I language interface module is used serially by all subtasks that comprise the application program.

2. The copy of the language interface module used to process the first call is used throughout the execution of the application program.

3. The using application passes entry and call parameter lists according to required conventions.

4. Each application program subtask group concurrently executing in separate regions uses a separate copy of the language interface module; that is, the language interface module is not reentrant and may not reside in the system link pack or resident access method areas.

Overlay may be used in the design of an application program provided the requirements in 2, 3, and 4 above are met. Only one message processing program may occur in a message processing region at any moment in time. Only one message is processed in a message processing region at a time.

If a message processing program is written in a serially reusable manner, it may process multiple input messages in serial fashion with the load of a single copy.

A Type 2 processing region is used for batch processing against data bases concurrently used for message processing. The batch program may reference the input/output message queues, the data bases used concurrently for message processing, and normal Operating System/360 data sets. A batch program in a Type 2 processing region may not reference Data Language/I data bases <u>not</u> used concurrently for message processing. Since a program in a Type 2 processing region may interface with the message queues, it may retrieve messages enqueued from terminals or enqueued as output from message programs, or the batch program may itself enqueue messages destined for terminals or message programs (Figure 29).

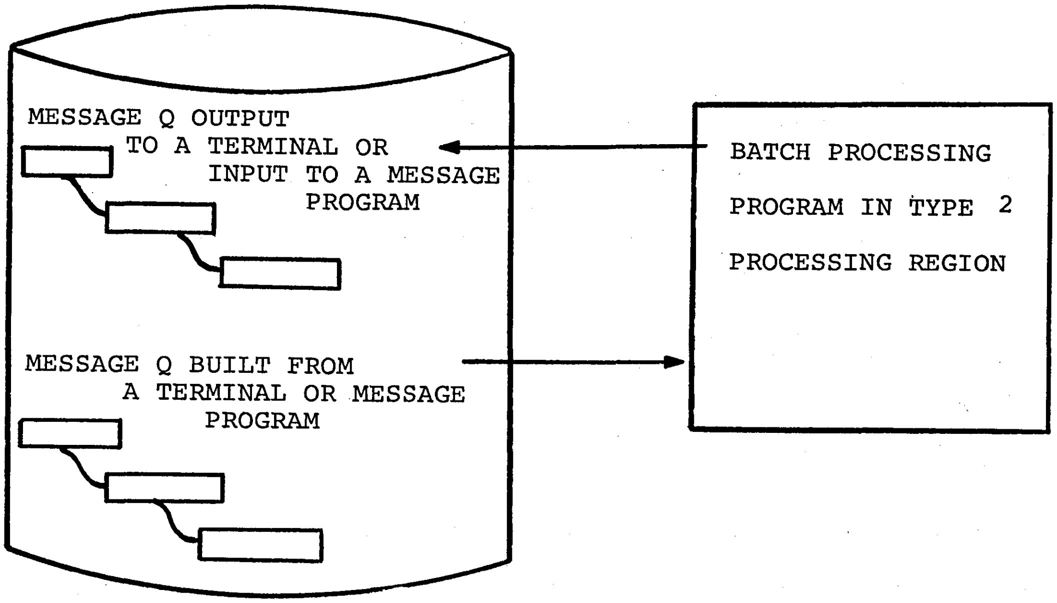


Figure 29. Type 2 processing region/message queue relationship

All programs used in a Type 2 processing region must be predefined to the IMS/360 control program like message processing programs. As with message processing regions, the Data Language/I calls from a Type 2 processing region are executed from the IMS/360 control program region. If a batch processing program is merely going to retrieve data from online data bases, it may reference the data bases with Data Language/I calls. However, if a batch processing program wishes to update data in or add data to online data bases, it issues Data Language/I data base calls or output messages to the message queues that are input to a message program. The data base updates are then performed by the message program. The message program may subsequently output messages to the message queue. These messages, the results of data base update, may be subsequently retrieved by a batch program in a Type 2 processing region (Figure 30). The latter method is preferred because data base calls performed by message programs have checkpoint/restart capabilities.

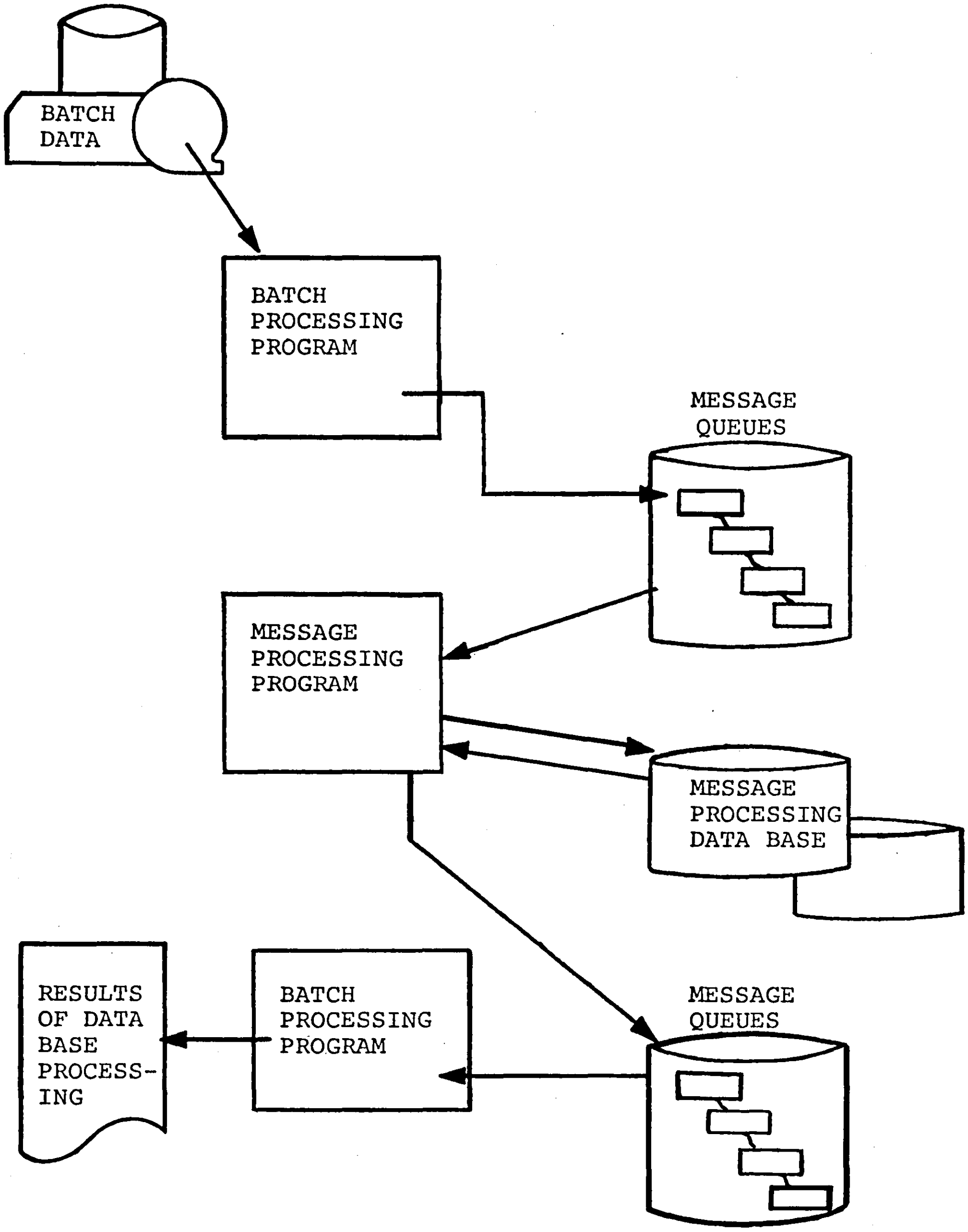


Figure 30. Batch program in Type 2 processing region

A Type 3 processing region is used for batch processing against Data Language/I data bases unrelated or not concurrently used for message processing. A batch program in a Type 3 processing region may not access IMS/360 message queues; however, it may use normal Operating System/360 data sets. The execution of Data Language/I call statements from a Type 1 or 2 processing region is performed in the IMS/360 control program region. The execution of Data Language/I call statements from a Type 3 processing region is performed by IMS/360 modules in the Type 3 processing region.

The structure of all batch processing programs utilizing Data Language/I calls is subject to the same rules as message processing programs unless otherwise stated. Whenever a Data Language/I call is executed from a processing region, the Operating System/360 task is placed in wait state until the call is completed.

The type of processing region is specified in the Job Control Language (JCL) statements for the job. The EXEC card PARM field is utilized. The IMS/360 Operations Manual, Volume I - Systems Operation provides a definition of the complete JCL required.

Application programs that execute using IMS/360 or Data Language/I may be written in any one of the following Operating System/360 programming languages: Assembler Language, COBOL, or PL/I. It is intended, however, that the programmer be able to benefit from the power of high-level languages for processing and the power of Data Language/I for data manipulation. Therefore, this discussion is oriented toward COBOL or PL/I.

The structural requirements put upon any application program by IMS/360 can be grouped into major areas and must be considered by every programmer.

- Entry
- Exit
- Calls
- Parameters for calls
- Segment I/O area
- Segment search arguments

The names listed below and shown in the examples throughout this chapter are standard and must be used by the programmer.

COBOL	PL/I	ASSEMBLER	STATEMENT USED WITH
DLITCBL	DLITPLI	ANY	ENTRY
CBLTDLI	PLITDLI	CBLTDLI	CALL

TYPE 3 REGION BATCH PROGRAM STRUCTURE

COBOL Batch Program Structure

Figure 31 illustrates in outline form all the fundamental parts in the structure of a Type 3 region batch program. Care should be taken to ensure that each item is considered when designing a batch program.

```
REF
NO.
        ENVIRONMENT DIVISION
            •
            •
        DATA DIVISION
        WORKING STORAGE SECTION
 1          77 FUNC-DB-IN   PICTURE XXXX VALUE'GU  '.
            77 FUNC-DB-OUT  PICTURE XXXX VALUE'REPL'.
            77 FUNC-DB-NEXT PICTURE XXXX VALUE'GHN '.
            77 CT PICTURE S9(5) (COMPUTATIONAL) VALUE +4.
            •
 2          01  SSA-NAME
 3          01  MAST-SEG-IO-AREA
            01  DET-SEG-IN-AREA
        LINKAGE SECTION
 4          01  DB-PCB-MAST
            01  DB-PCB-DETAIL

      ------------------------------------------------------

        PROCEDURE DIVISION

 5      ENTRY 'DLITCBL' USING DB-PCB-MAST,DB-PCB-DETAIL.
            :
 6      CALL 'CBLTDLI' USING FUNC-DB-IN, DB-PCB-DETAIL,
                DET-SEG-IN-AREA,SSA-NAME.
            :
 7      CALL 'CBLTDLI' USING CT, FUNC-DB-IN, DB-PCB-MAST,
                MAST-SEG-IO-AREA, SSA-NAME.

 8      CALL 'CBLTDLI' USING FUNC-DB-NEXT,DB-PCB-MAST,
                MAST-SEG-IO-AREA.
            :
 9      CALL 'CBLTDLI' USING FUNC-DB-OUT, DB-PCB-MAST,
                MAST-SEG-IO-AREA.
            •
            •
10      RETURN
      ------------------------------------------------------

11      COBOL - LANGUAGE INTERFACE

```

Figure 31. COBOL batch program structure

Figure 31 is a general illustration of the significant parts in the design of a COBOL batch program that retrieves data from a detail file to update a master data base. Neither the detail nor the master is a teleprocessing data base. A structure similar to the one shown must be used to create a teleprocessing or batch processing data base in a batch region.

The following explanation relates to the reference numbers along the left side of Figure 31.

1. A 77-level or 01-level working storage entry defines each of the CALL functions used by the batch program. Each picture clause is defined as four alphameric characters and has a value assigned for each function (for example, 'GUbb').

2. An 01-level working storage entry defines each segment search argument used by an application program. An example of an SSA definition, with lowercase b's representing blanks, is:

```
01 SSA-NAME

     02 SEG-NAME PICTURE X(8) VALUE 'ROOTbbbb'.

     02 SEG-QUAL PICTURE X VALUE '('.

     02 SEG-KEYNAME PICTURE X(8) VALUE 'KEYbbbbb'.

     02 SEG-OPERATOR PICTURE XX VALUE 'b='.

     02 SEG-KEY VALUE PICTURE X(6) VALUE 'vvvvvv'.

     02 SEG-END-CHAR PICTURE X VALUE ')'.
```

When the above COBOL syntax is decoded, it will be in a data string as follows:

```
ROOTbbbb(KEYbbbbb=vvvvvv)
```

3. An 01-level working storage entry defines the program segment I/O area.

4. An 01-level linkage section entry describes the PCB entry for every input or output data base. No PCB's can be included for terminals. It is through this linkage that a COBOL program may access the status codes after a Data Language/I call.

5. This is the standard entry point in the procedure division of a batch program. After the region controller has loaded and completed the PSB and one or more DBD's for the program in the batch region, it gives control to this entry point. The PSB contains all the PCB's used by the program. The USING statement at the entry point to the Type 3 region batch program must contain the same number of names in the same sequence as there are PCB's in the PSB.

   ```
   ENTER LINKAGE.

   ENTRY 'DLITCBL' USING pcbname-1,....pcbname-n.

   ENTER COBOL.
   ```

6. and 7. These are typical calls used to retrieve data from a data base using a qualified search argument.

   ```
   ENTER LINKAGE.

   CALL 'CBLTDLI' USING function, pcbname,
           segment-I/O-area,
           segment-search-argument.
   ENTER COBOL.
   ```

   Item 7 also shows the use of another parameter in the call made from COBOL to Data Language/I. This additional explicit parameter is a binary counter (fullword) of the number of remaining parameters in the current Data Language/I call. This allows the user to set up the parameters of a call in the working storage section of his data division and to truncate or expand this call through the use of the binary counter.

8. This is a typical call used to retrieve data from a data base using an unqualified search. This call is also a HOLD call for a subsequent delete or replace.

```
ENTER LINKAGE.

CALL 'CBLTDLI' USING function, pcbname,


        segment-I/O-area.

ENTER COBOL.
```

9. This call is used to replace data from a batch program onto a data base.

10. This RETURN causes the batch program to return control to the region controller. The format is:

```
ENTER LINKAGE.

RETURN.

ENTER COBOL.
```

11. A language interface is provided for all programming languages. This module is link-edited to the batch program and provides a common interface to IMS/360 and Data Language/I.

```
REF
NO.

        /* ------------------------------------------ */
        /*              ENTRY POINT                   */
        /* ------------------------------------------ */
1       DLITPLI:PROCEDURE (DB_PCB_MAST,DB_PCB_DETAIL)
            OPTIONS(MAIN);
        /* ------------------------------------------ */
        /*          DESCRIPTIVE STATEMENTS            */
        /* ------------------------------------------ */
2       DECLARE FUNC_DB_IN CHARACTER(4) INITIAL('GUbb');
        DECLARE FUNC_DB_OUT CHARACTER(4) INITIAL('REPL');
        DECLARE FUNC_DB_NEXT CHARACTER(4) INITIAL('GHNb');
          •
3       DECLARE SSA_NAME STATIC UNALIGNED,...;
4       DECLARE MAST_SEG_IO_AREA,...;
        DECLARE DET_SEG_IN_AREA,...;
          •
5       DECLARE 1 DB_PCB_MAST,...;
        DECLARE 1 DB_PCB_DETAIL,...;
          •
6       DECLARE THREE FIXED BINARY(31) INITIAL(3);
        DECLARE FOUR FIXED BINARY(31) INITIAL(4);


        /* ------------------------------------------ */
        /*    MAIN PART OF PL/I BATCH PROGRAM         */
        /* ------------------------------------------ */
          •
7       CALL PLITDLI(FOUR,FUNC_DB_IN,DB_PCB_DETAIL,
            DET_SEG_IO_AREA,SSA_NAME);
          •
8       CALL PLITDLI(FOUR,FUNC_DB_IN,DB_PCB_MAST,
            MASTER_SEG_IO_AREA,SSA_NAME);
          •
9       CALL PLITDLI(THREE,FUNC_DB_NEXT,DB_PCB_MAST,
            MAST_SEG_IO_AREA);
          •
10      CALL PLITDLI(THREE,FUNC_DB_OUT,DB_PCB_MAST,
            MAST_SEG_IO_AREA);
          •
11      END DLITPLI;

        ----------------------------------------------------

12      PL/I - LANGUAGE INTERFACE

```

Figure 32. General PL/I batch program structure

Figure 32 is a general illustration of the significant parts in the design of a PL/I Type 3 region batch program that retrieves data from a detail file to update a master data base. Neither the detail nor the master is a teleprocessing data base. A structure similar to the one shown must be used to create a teleprocessing or batch processing data base in a batch region.

The following explanation relates to the reference numbers along the left side of Figure 32:

1. This is the main standard entry point to a PL/I batch program. After the region controller has loaded and completed the PSB and one or more DBD's for the program in the batch region, it gives control to this entry point. The PSB contains all the PCB's used by the program. The entry point statement of the Type 3 region batch program must contain the same number of names in the same sequence as there are PCB's in the PSB.

   Note: When link-editing a compiled PL/I program with the language interface, the load module ENTRY should be either IHESAPB or IHESAPD with OPT=00 or 01 respectively, and the load module member should be the name of the PL/I program. The explanation is offered below.

The following entry points are to be used for PL/I object program main entry in a non-multitasking environment.

ENTRY IHESAPB

- For OPT=00, provides no optimization area.
- Provides pseudo register vector and library work space.
- Issues a SPIE macro-instruction.
- Transfers control to IHEMAIN.

ENTRY IHESAPD

- For OPT=01 reserves a 512-byte area for optimization.
- Same as the last three items for IHESAPB.

Note that neither of these entry points allows a PARM parameter to be passed from the EXEC job control language statement.

The following entry points are to be used for PL/I object programs operating in a multitasking environment:

ENTRY IHETSAA

- For OPT=00, provides no optimization area.
- Obtains storage for the Pseudo Register Vector Variable Data Area, task variable and event variable for the major task, etc.
- Attaches the PL/I major task and enters the wait state until either the event variable for the major task or the STOP ECB is completed.

ENTRY IHETSAB

Same as IHETSAA except that a 512-byte optimization area is acquired for the OPT=01 user.

2. By declaring, each working area defines each of the CALL functions used by the PL/I batch program. Each character string is defined as four alphameric characters, with a value assigned for each function (for example, 'GUbb'). Other constants and working areas may be defined in the same manner.

3. This working area defines all the segment search arguments used by the problem program. This SSA has been defined as a structure but is assumed to be a contiguous character string in storage.

   Example: (lowercase b's represent blanks)

   ```
   DCL  1  SSA_NAME STATIC UNALIGNED,
           2  SEG_NAME CHAR(8)          INIT('ROOTbbbb'),
           2  SEG_QUAL CHAR(1)          INIT('('),
           2  SEG_KEY_NAME CHAR(8)      INIT('KEYbbbbb'),
           2  SEG_OPERATOR CHAR(2)      INIT('b='),
           2  SEG_KEY_VALUE CHAR(6),
           2  SEG_END_CHAR CHAR(1)      INIT(')');
   ```

   Note: The UNALIGNED attribute is required for SSA data interchange with IMS/360. The SSA character string must reside contiguously in storage. Assignment of variables to key values, for example, could result in the construction of an invalid SSA if the key value had the ALIGNED attribute.

4. A working storage area entry defines the program segment I/O area.

5. A level 1 declarative (similar to COBOL's linkage section) describes the PCB entry for every input or output data base. No PCB's can be included for terminals. It is through this description that a PL/I program may access the status codes after a Data Language/I call.

6. This is a descriptive statement used to identify a binary number (fullword) that represents the "parameter count" of a call to Data Language/I. The parameter count value equals the remaining parameters following the parameter count set off by commas.

7. and 8. These are typical calls used to retrieve data from a data base using a qualified search argument.

   CALL PLITDLI (parameter count, function, pcbname, segment I/O area, segment search argument);

9. This is a typical call used to retrieve data from a data base using an unqualified search. This call is also a HOLD call for a subsequent delete or replace.

   ```
   CALL PLITDLI (parameter count, function, pcbname, segment I/O
           area);
   ```

10. This call is used to replace data from a Data Language/I batch program onto a data base.

11. This END statement causes the batch program to return control to the region controller. Another statement that causes the batch program to return control to the region controller is the RETURN statement. The RETURN statement may or may not immediately precede the END statement.

12. A language interface is provided for all programming languages. This module is link-edited to the batch program and provides a common interface to IMS/360 and Data Language/I.

Assembler Language Batch Program Structure

The entry point to an Assembler Language program that utilizes Data Language/I may have any desired name. However, Register 1, upon entry to the application program, contains the address of a variable-length fullword parameter list. Each word in this list contains a control block address that must be saved by the application program. The high-order byte of the last word in the parameter list has the 0 bit set to a value of 1 to indicate the end of the list. The addresses in this list are subsequently used by the application program when executing Data Language/I calls.

All Data Language/I calls from an Assembler Language program should be executed with the CALL macro-instruction. Register 1 must be constructed prior to execution of the CALL statement to point to the variable-length fullword parameter list. This may be done through operands of the CALL macro-instruction. The parameters in this list are addresses of:

- The input/output function
- The PCB control block address associated with the data base
- Input/output work area
- Zero or more segment search argument identifiers

The entry point for the CALL macro-instruction is CBLTDLI.

Application programs used in the batch Data Language/I environment may use both Data Language/I for data base processing and standard Operating System/360 data management for non-data base input/output operation.

MESSAGE OR TYPE 2 BATCH PROGRAM STRUCTURE

COBOL Message Program Structure

Figure 33 illustrates in outline form all the fundamental parts in the structure of a COBOL message or Type 2 region batch processing program. Care should be taken to ensure that each item is considered when designing a Type 1 region message program or Type 2 region batch program.

```
REF
NO.    ENVIRONMENT DIVISION.
           •
       DATA DIVISION
       WORKING - STORAGE SECTION.
 1         77 FUNC-IN PICTURE XXXX VALUE 'GU  '.
           77 FUNC-OUT PICTURE XXXX VALUE 'ISRT'.
           77 CT PICTURE S9(5)(COMPUTATIONAL) VALUE +4.
           •
 2         01  SSA -NAME.
           •
 3         01  MSG-SEG-IO-AREA.
           01  DB-SEG-IO-AREA.
           01  ALT-MSG-SEG-OUT.
       LINKAGE SECTION.
 4         01  TERM-PCB-IN.
           01  TERM-PCB-OUT.
           01  DB-PCB.
       ----------------------------------------------------------
       PROCEDURE DIVISION

 5     ENTRY 'DLITCBL' USING TERM-PCB-IN, TERM-PCB-OUT,
               DB-PCB.
           •
 6     CALL 'CBLTDLI' USING FUNC-IN, TERM-PCB-IN,
               MSG-SEG-IO-AREA.
           •
 7     CALL 'CBLTDLI' USING FUNC-IN, DB-PCB,
               DB-SEG-IO-AREA, SSA-NAME.
           •
 8     CALL 'CBLTDLI' USING CT,FUNCTION,-DB-PCB,
               DB-SEG-IO-AREA,SSA-NAME.
           •
 9     CALL 'CBLTDLI' USING FUNC-OUT, TERM-PCB-OUT,
               ALT-MSG-SEG-OUT.
           •
10     RETURN.

       ----------------------------------------------------------

11     COBOL-LANGUAGE INTERFACE
```

Figure 33. COBOL message program structure

Figure 33 is a general illustration of the steps in the design of a COBOL message program that processes an inquiry from a terminal, makes a reference to a data base for information, and sends an answer to the originating terminal or to an alternate terminal.

The following explanations are for a COBOL program and are keyed to the reference numbers along the left side of Figure 33.

1. A 77-level or 01-level working storage statement defines each of the CALL functions used by the message program. Each picture clause is defined as four alphameric characters and has a value assigned for each function (for example, 'ISRT').

2. An 01-level working storage statement describes each segment search argument used for a data base call. An example of an SSA definition, with lowercase b's representing blanks, is:

```
01  SSA-NAME.
    02  SEG-NAME    PICTURE X(8) VALUE 'ROOTbbbb'.
    02  SEG-QUAL    PICTURE X    VALUE '('.
    02  SEG-KEYNAME PICTURE X(8) VALUE 'KEYbbbbb'.
    02  SEG-OPERATOR PICTURE XX  VALUE 'b='.
    02  SEG-KEY-VALUE PICTURE X(6) VALUE'vvvvvv'.
    02  SEG-END-CHAR  PICTURE X  VALUE ')'.
```

When the above COBOL syntax is decoded and placed in storage, it will be in a data string as follows:

```
ROOTbbbb(KEYbbbbb=vvvvvv)
```

For further discussion, see the section, "Segment Search Arguments (SSA)".

3. An 01-level working storage statement describes each segment I/O area within the message program.

4. An 01-level linkage section entry describes the PCB statement first for the input terminal for the current message being processed, second for each output terminal other than the input terminal, and third for each data base (see "Program Communication Block (PCB) Formats"). It is through this linkage description that a COBOL program may access the status codes after a Data Language/I call.

5. This is the message program entry point and must be the first COBOL executable statement in the procedure division. There must be a PCB name for every PCB that will be used by the message program. The names of the PCB's used in the ENTRY statement must be specified in the same order as they are presented in the PSB generation execution for the program's associated PSB. The pcbnames can be specified in the linkage section in the same order, but this is not necessarily a requirement. The first pcbname must be for the terminal representing the source of the input message. The general format is:

   ```
   ENTER LINKAGE.

   ENTRY 'DLITCBL' USING pcbname-1,....pcbname-n.

   ENTER COBOL.
   ```

6. This is a typical call used to read the input (source) logical terminal. The first time this call is executed with function equal to GET UNIQUE, the first line of the message that caused the message program to be scheduled is brought into this program. If the input message consists of more than one line, subsequent lines can be obtained with a similar call but with the function equal to GET NEXT.

   ```
   ENTER LINKAGE.

   CALL 'CBLTDLI' USING function, pcbname, I/O-work-area.

   ENTER COBOL.
   ```

7. This call is used to access data from a data base other than a terminal data base. The format is the same as that in Item 6 above, except that the PCB refers to a data base and the segment search arguments define a particular data base segment.

8. This call is used to access data from a data base other than a terminal data base. The call performs the same function as Item

7 above, except that it illustrates the use of another parameter in the call made from COBOL to Data Language/I. This additional explicit parameter is a binary counter (fullword) of the number of remaining parameters in the current Data Language/I call. This allows the user to set up the parameters of a call in the working storage section of his data division and to truncate or expand this call through the use of the binary counter.

9. This call is used to reply to an output logical terminal (source) other than the terminal representing the source of the input message. If the output terminal is the same as the input terminal, this call utilizes the input source PCB. The format is the same as the one shown in Item 6 above, but the function must have a value of ISRT.

10. This operation causes the message program to return control to the region controller.

```
ENTER LINKAGE.

RETURN.

ENTER COBOL.
```

11. A language interface is provided for all COBOL programs. This module must be link-edited to the message processing program and provides a common interface to IMS/360 and Data Language/I for all CALL statements.

PL/I Message Program Structure

```
REF
NO.
        /* ------------------------------------------------*/
        /*             ENTRY POINT                         */
        /* ----------------------------------------------- */
1       DLITPLI:PROCEDURE(TERM_PCB_IN,TERM_PCB_OUT,
            DB_PCB)OPTIONS(MAIN);
        /* ----------------------------------------------- */
        /*             DESCRIPTIVE STATEMENTS              */
        /* ----------------------------------------------- */
2       DECLARE FUNC_IN CHARACTER(4)INITIAL('GUbb');
        DECLARE FUNC_OUT CHARACTER(4)INITIAL('ISRT');
          •
          •
3       DECLARE SSA_NAME STATIC UNALIGNED,...;
4       DECLARE 1 MSG_SEG_IO_AREA,...;
        DECLARE 1 DB_SEG_IO_AREA,...;
        DECLARE 1 ALT_MSG_SEG_OUT,...;
          •
5       DECLARE 1 TERM_PCB_IN,...;
        DECLARE 1 TERM_PCB_OUT,...;
        DECLARE 1 DB_PCB,...;
          •
6       DECLARE THREE FIXED BINARY(31) INITIAL(3);
        DECLARE FOUR FIXED BINARY(31) INITIAL(4);
          •

        /* ----------------------------------------------- */
        /*       MAIN PART OF PL/I PROGRAM                 */
        /* ----------------------------------------------- */
          •
7       CALL PLITDLI(THREE,FUNC_IN,TERM_PCB_IN,
            MSG_SEG_IO_AREA);
          •
8       CALL PLITDLI(FOUR,FUNC_IN,DB_PCB,DB_SEG_IO_AREA,
            SSA_NAME);
          •
9       CALL PLITDLI(THREE,FUNC_OUT,TERM_PCB_OUT,
            ALT_MSG_SEG_OUT);
          •
10      END DLITPLI;

        ------------------------------------------------------

            PL/I - LANGUAGE INTERFACE
```

Figure 34. General PL/I message program structure

Figure 34 is a general illustration of the steps in the design of a PL/I Type 1 message region program that processes an inquiry from a terminal, makes a reference to a data base for information, and sends an answer to the originating terminal or to an alternate terminal.

The following explanations are for a PL/I program and are keyed to the reference numbers along the left side of Figure 34:

1. This is the main standard entry point to a PL/I message program. There must be a PCB name for every PCB in the PSB associated with the message program. In addition there must be one PCB name for

the source of the input message. This must be the first PCB name.

Note: When link-editing a compiled PL/I program with the language interface, the load module ENTRY address should be either IHESAPB or IHESAPD with OPT=00 or 01 respectively, and the load module member should be the name of the PL/I program.

The following entry points are to be used for the PL/I object program main entry in a non-multitasking environment.

ENTRY IHESAPB

- For OPT=00, provides no optimization area.
- Provides pseudo register vector and library work space.
- Issues a SPIE macro-instruction.
- Transfers control to IHEMAIN.

ENTRY IHESAPD

- For OPT=01, reserves a 512-byte area for optimization.
- Same as last three items for IHESAPB.

Note that neither of these entry points allows a PARM parameter to be passed from the EXEC job control language statement.

Following entry points are to be used for PL/I object programs operating in a multitasking environment.

ENTRY IHETSAA

- For OPT=00, provides no optimization area.
- Obtains storage for the pseudo register vector variable data area, task variable and event variable for the major task, etc.
- Attaches the PL/I major task and enters the wait state until either the event variable for the major task or the STOP ECB is completed.

ENTRY IHETSAB

Same as IHETSAA except that a 512-byte optimization area is acquired for the OPT=01 user.

2. By declaring, each working area defines each of the CALL functions used by the PL/I message program. Each character string is defined as four alphameric characters and has a value assigned for each function (for example, 'ISRT'). Other constants and working areas may be defined in this manner.

3. This working area defines all the segment search arguments used by the problem program. This SSA has been defined as a structure but is assumed to be a contiguous character string in storage.

Example: (lowercase b's represent blanks)

```
DCL  1  SSA_NAME STATIC UNALIGNED,
        2  SEG_NAME CHAR(8)          INIT('ROOTbbbb'),
        2  SEG_QUAL CHAR(1)          INIT('('),
        2  SEG_KEY_NAME CHAR(8)      INIT('KEYbbbbb'),
        2  SEG_OPERATOR CHAR(2)      INIT('b='),
        2  SEG_KEY_VALUE CHAR(6),
        2  SEG_END_CHAR CHAR(1)      INIT(')');
```

Note: The UNALIGNED attribute is required for SSA data interchange with IMS/360. The SSA character string must reside contiguously in storage. Assignment of variables to key values, for example, could result in the construction of an invalid SSA if the key value had the ALIGNED attribute.

4. A working storage area entry defines the program segment I/O area. Message input and output areas should be defined as a static structure.

5. A level 1 declarative (similar to COBOL's linkage section) describes the PCB statement first for the input terminal for the current message being processed, second for each output terminal other than the input terminal, and third for each data base (see section on PCB formats). It is through this description that a PL/I program may access the status codes after a Data Language/I call.

6. This is a descriptive statement used to identify a binary number (fullword) that represents the "parameter count" of a call to Data Language/I. The parameter count value equals the remaining parameters following the parameter count set off by commas.

7. This is a typical call used to read the input (source) logical terminal. The first time this call is executed with function equal to GET UNIQUE, the first line of the message that caused the message program to be scheduled will be brought into this program. If the input message consists of more than one line, subsequent lines can be obtained with a similar call but with the function equal to GET NEXT.

   ```
   CALL PLITDLI (parameter count, function, pcbname, segment I/O
   area);
   ```

8. This call is used to access data from a data base other than a teleprocessing data base. The format is the same as the one in Item 7 above, except that the PCB refers to a data base and the segment search argument defines a particular data base segment.

9. This call is used to reply to an output logical terminal (source) other than the terminal representing the source of the input message. If the output terminal is the same as the input terminal, this call utilizes the input source PCB. The format is the same as the one shown in 7 above, but function must have a value of ISRT.

10. This END statement causes the batch program to return control to the region controller. Another statement that causes the batch program to return control to the region controller is the RETURN statement. The RETURN statement may or may not immediately precede the END statement.

11. A language interface is provided for all programming languages. This module is link-edited to the batch program and provides a common interface to IMS/360 and Data Language/I.

Assembler Language Message Program Structure

See the preceding section, "Assembler Language Batch Program Structure". The user should remember that an Assembler Language message program will receive upon entry a PCB parameter list address in register 1. The first address in this list is to the input/output terminal PCB. Any alternate output destination PCB addresses follow and finally any data base PCB addresses. The last address in the list is signed negative.

THE LANGUAGE INTERFACE

The language interface module provides the standard interface mechanism, which allows a message or batch processing program to communicate with IMS/360 for Data Language/I data base and message calls. A copy of this module must be link-edited with each message or batch processing program. When the module is entered, the structure and addresses of the Data Language/I call are verified. If an invalid call structure is received, a nonblank status code is returned to the message or batch processing program. In a Type 1, 2, or 3 processing region, if the PSB associated with the program to be executed contains information conflicting with the DBD, the task within that region will be terminated. In a Type 3 batch processing region, if a call is issued that requires a PCB address and a PSB is not provided or is invalid, the application program in that Type 3 region is terminated.

The language interface is designed to handle all supported languages that interface with IMS/360-Data Language/I. Upon entry into the language interface, a pointer to a parameter list is provided by the call structure.

Two types of parameter lists may be constructed: implicit lists and explicit lists. The COBOL program may use either type of list, and the language interface modifies the list as required to pass an implicit list to IMS/360. The list is restored to its original format before being returned to the application program. PL/I, on the other hand, allows only explicit parameter lists.

The following calls permit the standard entry points to the correct language interfaces and should be used for all data calls.

```
PL/I - CALL PLITDLI......

COBOL - CALL 'CBLTDLI'......

Assembler - CALL CBLTDLI,......
```

Parameter List Contents

The generated format of the parameter lists may be of interest to the application programmer (see Figure 35). The actual construction of these lists is accomplished by the CALL statement parameters in the high-level languages. The contents of these lists, as seen by IMS/360, are shown for information purposes.

The format for CALL parameter lists is standard and should be as shown in Figure 35.

IMPLICIT PARAMETER LIST CONTENTS

Bytes	
+0	Function address
+4	PCB address or PSB name address
+8	Segment input/output area address
+12	First Segment Search Argument address
+16	Next Segment Search Argument address
+20	Last Segment Search Argument address

The high-order byte of the word containing the last parameter in an implicit parameter list contains an X'80'

EXPLICIT PARAMETER LIST CONTENTS

Bytes	
+0	Parameter count address
+4	Function address
+8	PCB address or PSB name address
+12	Segment input/output area address
+14	First Segment Search Argument address
+20	Next Segment Search Argument address
+24	Last Segment Search Argument address

Figure 35. Parameter list contents

Parameter count is a binary fullword count of the number of other parameters that exist in the parameter list.

In PL/I, the function, PCB, segment I/O area, and segment search argument addresses are addresses of the dope vectors for the parameters.

The function and segment search argument should be defined as a character string when PL/I is used. Segment input/output area should be a static structure when using PL/I. All PCB's can be structured to any level in PL/I.

Note that in those instances where the CALL statement references an input or terminal PCB, no SSA's may be used, and their addresses must not be in the parameter list.

Using an SPIE macro-instruction, the application language interface disables any interrupt traps set by the application program.

SEGMENT SEARCH ARGUMENTS (SSA)

When an application programmer requests Data Language/I to perform data base functions, it is frequently necessary for him to specifically identify a particular segment by its key field and the key fields of all parent segments along the hierarchical path leading to that segment. These key field values do not appear directly in the CALL statement parameters provided to Data Language/I. Instead, a segment search argument name is given, which points to an area in the user's program which contains the actual segment search argument (SSA).

Segment search arguments may be used with GET calls and are required for all INSERT calls.

The SSA may consist of two pieces, the segment name and (as required) a segment qualification statement. The segment name points Data Language/I to the entry in the data base description that contains and defines the characteristics of the segment and its key field.

The qualification statement contains information that Data Language/I uses to test the value of the segment key or data field with the data base to determine whether the segment meets the user's specifications. Using this approach, Data Language/I does the data base segment searching, and the program need process only those segments in which it is interested.

A segment qualification statement is composed of several elements. Except where they are used to fill out a field, there must be no blanks in this statement. The complete qualification for each segment is contained between the left and right parentheses.

The segment search argument (SSA) structure is:

```
segment-name(segment-field-name-RO-comparative-value)
```

Segment Name

The segment name must be eight bytes long.

Segment-name

is the segment name that pertains to a specific segment in the hierarchical structure of a data base record; it is established by the Data Base Description.

Segment Qualification Statement

The segment qualification statement contains the begin-qualification-operator, the segment-field-names, the relational-operator, the comparative-value, and the end-qualification-operator. If a segment search argument has no qualification statement, the eight-byte segment name must be followed by a character other than (.

Begin-qualification-operator

is the left parenthesis, (. It indicates the beginning of a qualification statement.

Segment-field-name

is the name of a segment search field which appears in the description of that segment type in the Data Base Description. The name is eight characters long, with right-justified embedded blanks as required. If the I/O function is GET, the named field may be either the key field or a data field within a segment. It must be the key field if the segment search argument applies to a root segment. Only the last SSA may be qualified on a data field. The last SSA in the INSERT call may not have a qualification statement.

RO = Relational-Operator

is a set of two characters that express the manner in which the contents of the field, referred to by the segment-field-name, are to be tested against the comparative-value. The sequence of checking is less than, equal to, then greater than.

Operator	Meaning
b =	must be equal to
b >	must be greater than
b <	must be less than
¬ =	must be not equal to
= >	must be equal to or greater than
= <	must be equal to or less than

Note: As used above, the lowercase b represents a blank character.

If the qualification statement applies to a root segment, only the =, =>, or > relational operators may be used.

Comparative-value

is the value against which the contents of the field referred to by the segment-field-name are to be tested. The length of this entry must be equal to the length of the named field in the segment of the data base, that is, it includes leading or trailing blanks (for alphameric) or zeros (usually needed for numeric fields) as required.

End-qualification-operator

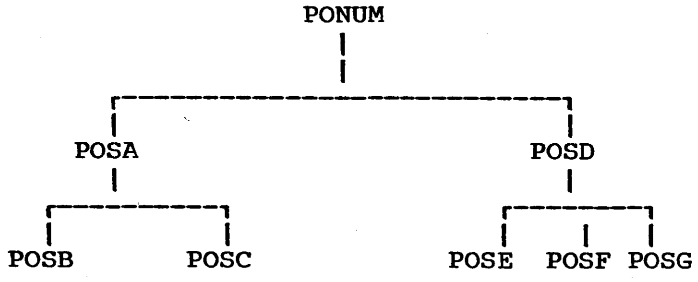
is the right parenthesis,). It indicates the end of a qualification statement.

The qualification statement test is terminated as soon as an occurrence within the data base of a segment-type is found that satisfies that qualification test. This procedure continues for all SSA's in a Data Language/I data base call until the desired segment is found.

The following are examples of segment search arguments with and without a qualification statement.

Example of SSA Usage

The data base structure and the segment names are as follows:



The segment search argument for the various degrees of qualification may then be as follows:

1. SSA with no qualification

PONUMbbbb

A call using an unqualified SSA can access the next root segment called PONUM. Note that the ninth position must not contain a left parenthesis.

2. SSA with qualification

PONUMbbb(ACTUALPOb=AB60733)

A call using this simple qualification accesses the root segment of a data base record whose root segment key field, called ACTUALPO, has a value of AB60733.

3. SSA's that form a complex qualification

   a. PONUMbbb(ACTUALPOb=AB60733) POSDbbbb(AFLDbbbbb=4234) POSFbbbb(KFLDbbbbb=24357)

   This type of qualification accesses the POSF segment whose key field, KFLD, value is 24357, whose parent segment's key field, AFLD, value equals 4234, and whose root segment's key field, ACTUALPO, equals AB60733.

   b. PONUMbbb(ACTUALPOb=AB60733) POSDbbbb(AFLDbbbbb=4234)

   This type of qualification obtains the POSD segment whose AFLD field equals 4234, and whose root segment's key field, ACTUALPO, equals AB60733.

SEGMENT INPUT/OUTPUT AREAS

The segment input/output (I/O) area is an area in the application program into which Data Language/I puts a requested segment, or from which Data Language/I takes a designated segment. If a common area is used, it must be as long as the longest segment to be processed. The segment I/O area name points to the leftmost byte of the area. Segment data is always left-justified within a common segment I/O area.

Example of Segment I/O Area

In Figure 33, the message return area for COBOL is defined in the working storage section by:

```
01 MSG-SEG-IO-AREA.          (Reference 3)
    02 CHAR-COUNT    PICTURE S99 COMPUTATIONAL.
    02 FILLER        PICTURE S99 COMPUTATIONAL.
    02  TRANS-CODE     PICTURE X(8).
    02  TEXT-AREA      PICTURE X(110).
```

When a message is to be brought into this area, the following call is used (Reference 6):

```
CALL 'CBLTDLI' USING FUNC-IN, TERM-PCB-IN, MSG-SEG-IO-AREA.
```

In Figure 34, the message return area for PL/I is defined by:

```
    DECLARE     1    MSG_SEG_IO_AREA STATIC,
                2    LL  FIXED BINARY(31),
                2    ZZ  BIT (16) INITIAL((16)'0'B),
                2    TXT_AREA CHAR(132);
```

Note: LL is a fullword, thus making it easier to access. In determining the actual length of MSG_SEG_IO_AREA, LL is still considered two bytes. The length passed by IMS/360 to LL in the above example is 136 bytes.

When a message is to be brought into this area, the following call is used (Reference 7):

```
            CALL PLITDLI (THREE,FUNC_IN,TERM_PCB_IN,
                    MSG_SEG_IO_AREA);
```

The message is located in the MSG-SEG-IO-AREA after the return from this call. Notice that the first two bytes of the segment I/O area contain a count of the number of bytes in the segment or message lines (even though the FIXED BINARY (31), indicates four bytes). The next two bytes are reserved for use by IMS/360. The count includes the length of its two bytes, the two reserved, the transaction code, and the message text.

Example of Data Base Segment I/O Area

In Figure 33, the data base segment return area for COBOL is defined in the working storage section by:

```
       01 DB-SEG-I-AREA.            (Reference 3)
           02 DB-SEGMENT PICTURE X(110).
```

When the data base segment is brought into this area, the following call is used (Reference 7 or 8):

```
            CALL 'CBLTDLI' USING FUNC-IN,DB-PCB,DB-SEG-IO-AREA,
                SSA-NAME.
```

In Figure 34, the data base segment return area for PL/I is defined by:

```
    DECLARE     1    DB_SEG_IO_AREA,
                2    DB_SEG_TXT CHAR(110);
```

When a data base segment is brought into this area, the following call is used (Reference 8):

```
            CALL PLITDLI(FOUR,FUNC_IN,DB_PCB,DB_SEG_IO_AREA,
                SSA_NAME);
```

Note: There is no count field (2 bytes) or reserved area (2 bytes) appended to the front of the data base segment as there is in the message segment return area. The programmer usually knows the length of his segments.

PROGRAM COMMUNICATION BLOCK (PCB) FORMATS

A Program Communication Block (PCB) exists external to an application program for each terminal and data base used by the program. The linkage section of the data division of a COBOL program defines an external data field for each PCB. The EXTERNAL data attribute in PL/I performs the same function.

The PCB is a set of contiguous fields that provide the application program with the ability to make Data Language/I calls supplying the following information:

- The name of the data base to be processed
- The specification of the Data Language/I functions that will be used
- Indications of the types of segments to be processed
- Areas for receiving status responses from Data Language/I

No initial values are defined in a PCB in the linkage section. The values for a PCB exist in the Program Specification Block (PSB) and are fixed at PSBGEN time. Under IMS/360, there are two types of PCB's: one type is for a data base, and the other is for an online terminal.

PCB for a Terminal

The requirements for this type of PCB are as follows. The first entry is at the first level and is the name of the PCB. The additional entries for this PCB are at the second level. The second entry is the logical terminal name, which must be a maximum of eight characters in length. If this name is less than eight characters in length, it must be padded with blanks to eight positions. The next entry is a reserved field for Data Language/I use and must be two characters in length. The following field is the status code feedback area and must be two characters in length.

For input terminals, one additional field is required. This is the input prefix and is twelve characters in length.

COBOL Example

The following COBOL example would be found in the linkage section of the data division. This example is for either an input or an input-and-output teleprocessing terminal.

```
    01     INOUT-PCB.

       02  IO-TERMINAL  PICTURE X(8).
       02  IO-RESERVE   PICTURE XX.
       02  IO-STATUS    PICTURE XX.
       02  IN-PREFIX.

           03 FILLER PICTURE X.
           03 I-JULIAN-DATE PICTURE S9(5) COMPUTATIONAL-3.
           03 INPUT-TIME    PICTURE S9(7) COMPUTATIONAL-3.
           03 FILLER PICTURE X(4).
```

Time is in two positions for hours, minutes, and seconds; and one position for tenths of seconds.

PL/I Example

The following PL/I example would be found in the descriptive statement parts of the PL/I problem program. This example is for either an input or an input-and-output teleprocessing terminal.

```
    DECLARE    1    INOUT_PCB,
               2    IO_TERMINAL CHARACTER(8),
               2    IO_RESERVE  BIT(16),
               2    IO_STATUS   CHARACTER(2),
               2    IN_PREFIX,
                    3   PRE_DATE        FIXED DECIMAL(7,0),
                    3   PRE_TIME        FIXED DECIMAL(7,0),
                    3   PRE_MSG_COUNT   FIXED BINARY(31,0);
```

A terminal that is used purely for output would have a PCB similar to the one above, but without the last level-two and level-three lines. The input prefix has no meaning for output messages.

PCB for a Data Base

The PCB provides specific areas used by Data Language/I to advise the application program of the results of its calls. At execution time, all PCB entries are Data Language/I-controlled, where control means the exclusive authority to change the contents of a PCB entry. The programmer exercises his options as to what goes into the PCB at PSB generation time.

The following fields comprise a PCB for a data base:

1. Name of the PCB - This area refers to the entire structure of PCB entries and is used in program statements.

2. Name of Data Base Description - This field provides the DBD name from the library of Data Base Descriptions. It contains character data and is eight bytes long.

3. Segment Hierarchy Level Indicator - Data Language/I loads this area with the level number of the lowest segment encountered in its attempt to satisfy a program request. When a retrieve is successfully completed, the level number of the retrieved segment is placed here. If the retrieve is unsuccessful, the level number returned is that of the last segment, along the path to the desired segment, that satisfied the segment search argument.

This field contains character data; it is two bytes long and is a right-justified numeric.

4. Data Language/I Status Code - A status code that indicates the results of a Data Language/I call is placed in this field and remains here until another Data Language/I call uses this PCB. (Specific status codes are discussed with their associated calls in a later section of this manual.) This field contains two bytes of character data. When a successful call is executed, this field is returned blank or with a warning status indication.

5. Data Language/I Processing Options - This area contains a character code which tells Data Language/I the kinds of calls that will be used by the program for data base processing. This field is four bytes long. Only one of the following processing options may be specified in a particular PCB. It is left-justified to the first byte of the four-byte field. The remaining three bytes are reserved.

Possible values for the processing options are:

G - for get function

A - for get, delete, insert, and replace functions

L - for loading a hierarchical indexed sequential or hierarchical sequential data base

If the delete, replace, or insert option is specified for a hierarchical indexed sequential data base, A must be used. The only valid options for a hierarchical sequential data base are G and L, and they are mutually exclusive in the same PCB. The L option is mutually exclusive with other options in the same PCB.

6. Reserved Area for Data Language/I - Data Language/I uses this area for its own internal linkage related to an application program. This field is one binary word.

7. Segment Name Feedback Area - Data Language/I fills this area with the name of the lowest segment encountered in its attempt to satisfy a call. When a retrieve is successful, the name of the retrieved segment is placed here. If a retrieve is unsuccessful, the name returned is that of the last segment, along the path to the desired segment, that satisfied the segment search argument. This field contains eight bytes of character data. This field may be useful in GN and GNP calls.

8. Length of Key Feedback Area - This entry specifies the length of the area required to contain the completely qualified key of any sensitive segment. This field is one binary word. The completely qualified key of a third-level segment includes the first- and second-level keys.

9. Number of Sensitive Segment Types - This entry specifies the number of segment types in the data base to which the application program is sensitive. This field is one binary word.

10. Key Feedback Area - Data Language/I places in this area the completely qualified key of the lowest segment encountered in its attempt to satisfy a call. When a retrieve is successful, the key of the requested segment and the key field of each segment along the path to the requested segment are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key and following the

hierarchical path. When a retrieve is unsuccessful, the keys of all segments along the path to the requested segment for which searching was successful are placed in this area.

COBOL Example

Figure 36 is an example of a PCB for a data base in a COBOL program. The reference numbers relate to the preceding description of entries.

```
Reference
1      01  SAMPLE PCB.
2          02 DBD-NAME          PICTURE X(8).
3          02 SEG-LEVEL         PICTURE XX.
4          02 STATUS CODE       PICTURE XX.
5          02 PROC-OPTIONS      PICTURE XXXX.
6          02 RESERVE-DLI       PICTURE S9(5) COMPUTATIONAL.
7          02 SEG-NAME-FB       PICTURE X(8).
8          02 LENGTH-FB-KEY     PICTURE S9(5) COMPUTATIONAL.
9          02 NUMB-SENS-SEGS    PICTURE S9(5) COMPUTATIONAL.
10         02 KEY-FB-AREA.      PICTURE X(n).
```

<u>Note:</u> n is set at PSBGEN time.

Figure 36. COBOL example

PL/I Example

Figure 37 is an example of a PCB for a data base in PL/I program. The reference numbers relate to the preceding description of entries.

```
Reference
Number

  1    DECLARE 1   SAMPLE PCB,
  2            2   DBD_NAME        CHARACTER(8),
  3            2   SEG_LEVEL       CHARACTER(2),
  4            2   STATUS_CODE     CHARACTER(2),
  5            2   PROC_OPTIONS    CHARACTER(4),
  6            2   RESERVE_DLI     FIXED BINARY(31,0),
  7            2   SEG_NAME_FB     CHARACTER(8),
  8            2   LENGTH_FB_KEY   FIXED BINARY(31,0),
  9            2   NUMB_SENS_SEGS  FIXED BINARY(31,0),
  10           2   KEY_FB_AREA     CHARACTER(n);
```

<u>Note:</u> n is set at PSBGEN time.

Figure 37. PL/I example

It should be noted that no initial values have been defined in the PCB. The values for the entries in the PCB exist in the Program Specification Block (PSB). The attributes are defined for reference.

CHAPTER 6. APPLICATION PROGRAM DETAILS

DESCRIBING THE PROGRAM TO IMS/360

To this point in the manual, IMS/360 and its facilities have been introduced and its data organization discussed. Chapter 5 dealt with the application program structure as it relates to IMS/360. In this chapter, more details are given.

Before these added details are presented, however, a checklist is provided that is to be used as an aid to the application programming function of IMS/360 in completion of the tasks at hand -- an attempt to present the total picture in abbreviated form.

Programmer's Checklist

This checklist is by no means to be considered chronological, nor are the checks to be accomplished in the sequence given; it is not intended as the order of an approach to the application function of IMS/360--there are many ways to start implementing the system. This checklist does provide the overall in three general breakdowns: (1) general considerations, (2) program considerations, and (3) data base considerations.

The following is an explanation of the columns of the checklist:

Column 1. Checklist item under cosideration.

Column 2. Is a teleprocessing application program affected? (X means YES.)

Column 3. Is a batch application program affected?

Column 4. Is the user's application program planning affected by this item?

Column 5. Is IMS/360's DBD generation affected by this item?

Column 6. Is IMS/360's PSB generation affected by this item?

Column 7. Is IMS/360's System Definition affected by this item?

Column 8. Is IMS/360's Security Maintenance program affected by this item?

Column 9. In which IMS/360 manual will more details be found about this particular item?

Abbreviation	Full Title
SOM	IMS/360 Operations Manual Volume I - Systems Operation
MOM	IMS/360 Operations Manual Volume II - Machine Operation
PDM	IMS/360 Program Description Manual
SM	IMS/360 System Manual Volume I - Program Logic
OS/360	Appropriate Operating System/360 Manual

PROGRAMMER'S CHECKLIST

①	②	③	④	⑤	⑥	⑦	⑧	⑨
ITEM	TELE-PROCESSING	BATCH	APPL. PLANNING	DBDGEN	PSBGEN	SYSGEN	SECURITY MAINT.	DETAIL IN WHICH MANUAL
1. When considering application program structure, have the following been considered?	X	X	X					PDM SOM
a. Core limits	X	X	X					SOM
b. Overlay structure	X	X	X					N/A
c. Program chaining	X	X	X					PDM
d. IMS/360 restart	X	X	X					PDM,SOM
e. Storage devices	X	X	X	X		X	X	PDM,SOM
2. Selection of Type I programming systems (MVT, MFT-II, or PCP)	X	X	X			X		SOM
3. Select type of IMS/360 processing region (Type 1, 2, or 3)	X	X	X			X		PDM,SOM
4. Consider how many regions or partitions the application will need at one time.			X			X		PDM,SOM
a. Within those regions or partitions, how many requests are anticipated? (Terminal I/O, Message Queues, and DL/I data base requests)			X			X		PDM,SOM
5. Select application program name.	X	X			X	X		PDM,SOM
6. Select application program language.	X	X	X		X			PDM
7. Select telecommunications system for IMS/360 teleprocessing environment.	X		X					PDM,SOM, MOM
a. Terminal hardware and network	X		X			X		↓
b. Select transaction codes for application program.	X	X	X		X	X		
(1) Specify priority for each transaction code.	X		X			X	X	
c. Are messages to be entered at remote terminals single or multiple line?	X		X			X		
(1) Select whether, after input of message, terminal is to continue input of other messages or wait until previous message has been processed.	X		X			X		
d. Specify the length of time to process the message.	X		X			X		
e. Specify the number of messages to be processed per application program load in a region.	X		X			X		
f. Specify the line groups for the same terminal types.	X		X			X		
(1) Specify whether line group is dialup (switched); if so, specify telephone numbers.	X		X			X	X	
g. If 1050 system, specify whether station control/switched or station control/nonswitched.	X		X			X		

PROGRAMMER'S CHECKLIST (continued)

(1) If station control/ switched, specify Autocall or Autoanswer.	X		X			X		PDM,SOM, MOM
(2) If station control/ nonswitched, option is Autopoll.	X		X			X		↓
h. If 2740 system, specify station control/nonswitched or no station control/ switched.	X		X			X		
(1) If station control/ nonswitched, option is Autopoll or wrap/open.	X		X			X		
(2) If no station control/ switched, option is Autocall or Autoanswer.	X		X			X		
i. Specify each communication line with physical terminals and logical terminal names, their features, and their component addresses.	X		X		X	X	X	PDM,SOM, MOM,SM
j. Describe input and output queue control record and message data sets desired.	X		X			X		SOM,SM
k. Specify master terminal name after giving consideration to master terminal operation relationship to application program.	X		X		X	X		PDM,SOM, MOM
1. Select for terminal operation whether the terminal type style be uppercase, lowercase, or a mixture of both for input or output data translation.	X		X			X		SOM
8. Specify password, terminal, and dialup (switched) password security.	X		X				X	SOM,MOM
9. Specify all data base names for this application program.	X	X	X	X	X	X		PDM ,SOM
a. Specify what type of processing region.	X	X	X			X		↓
b. Select maximum number of data bases that will be in use at one time.	X	X	X	X		X		
c. Specify how application program intends to use each data base (read-only, update, exclusive use).	X	X	X			X		
(1) Also specify application program options (get, delete, insert, replace, load).	X	X	X		X			
10. Specify what access method wanted for each data base, their data set names, and their storage types.	X	X	X	X		X		PDM ,SOM MOM
11. Specify the application program's hierarchical (sensitive) segments and parent relationships.	X	X	X	X	X			PDM
a. Describe in detail the (sensitive) segments.	X	X	X		X			PDM
12. Check the entry point to the application program.	X	X						PDM
13. Plan the residence of MACLIB, RESLIB, PGMLIB, PSBLIB, DBDLIB, and PROCLIB.			X			X		SOM,MOM
14. Plan and specify the statistical reports from IMS/360 system required for this application program.	X	X	X					PDM ,SOM MOM

ENTRY TO APPLICATION PROGRAMS

For purposes of standardization and clarity, a standard entry point is used for programs to be run under IMS/360 or Data Language/I. The first statement in the PROCEDURE DIVISION of a COBOL program should be as follows:

```
ENTER LINKAGE.

ENTRY 'DLITCBL' USING pcbname-1,.....pcbname-n.

ENTER COBOL.
```

The first procedure of a PL/I program should be:

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

The "pcbname" parameters in these statements establish a correlation between the problem program and the PCB's with which the program deals. Each pcbname must appear at the first level in either the linkage section (COBOL) or an external DECLARE statement (PL/I). If the problem program does not deal with terminals, pcbname-1 through pcbname-n correspond positionally to the PCB's specified during PSB generation. For message programs, the input terminal PCB does not appear in the PSB, but is determined by IMS/360 at process time. For message programs, therefore, pcbname-1 corresponds to the input terminal, and pcbname-2 through pcbname-n correspond positionally to the PCB's specified during PSB generation.

Note: When using PL/I and link-editing a compiled PL/I program with the language interface, the load module ENTRY should be either IHESAPB or IHESAPD, and the load module member name should be the name of the PL/I program. See Chapter 5 under description of PL/I program structure for more details.

DATA LANGUAGE/I DATA BASE CALLS

The data services of Data Language/I are available to the application program through the use of standard language calls. The following calls are used in conjunction with the function codes shown below.

For COBOL - CALL 'CBLTDLI' USING function-code, pcbname, segment I/O area, ssa......

For PL/I - CALL PLITDLI (parm-count, function-code, pcbname, segment I/O-area,ssa.....);

Valid message and batch processing program Data Language/I call functions are:

Meaning	Function Code	Usage
GET UNIQUE	'GUbb'	Message or Data Base Segment
GET NEXT	'GNbb'	Message or Data Base Segment
GET NEXT WITHIN PARENT	'GNPb'	Data Base Segment Only
GET HOLD UNIQUE	'GHUb'	Data Base Segment Only

GET HOLD NEXT	'GHNb'	Data Base Segment Only
GET HOLD NEXT WITHIN PARENT	'GHNP'	Data Base Segment Only
INSERT	'ISRT'	Message or Data Base Segment
DELETE	'DLET'	Data Base Segment Only
REPLACE	'REPL'	Data Base Segment Only

The GET UNIQUE Call (GUbb) - Data Base

The GET UNIQUE call is used to retrieve a unique statement occurrence from the data base described in the PCB. The GET UNIQUE call can be used for random processing, or it can be used to establish the position in the data base where sequential processing is to begin.

SSA's in GET UNIQUE calls must conform to the following rules:

1. The call must have SSA's.
2. The first SSA must be for the root segment, and any following SSA's must proceed down a hierarchical path with no missing intermediate levels. The first SSA must be qualified, but lower level SSA's may be qualified or unqualified.
3. The search field must be the key field in the qualification statement of the root SSA, and the operator must be =, >, or = >.
4. The search field for level-2 and lower SSA's may be any defined field within the segment if the organization is HISAM. If the organization is HSAM, only the last SSA may be qualified on a data field. The operator may be =,¬=, <, >, =>, or <=. A field is defined if it is described by a FLDK or FLD card at DBD generation time for the data base. All comparisons on key or data fields are logical bit-for-bit compares.

   One method that could be used to accomplish positioning at the beginning of a data base is to issue a qualified GU call against the root segment. The qualification should use an = > (equal to or greater than) operator for a value less than the key field of the first root. Binary zeros or EBCDIC blanks are suggested.

Status Codes for GET UNIQUE Calls

At the completion of a GET call, a status code indicating the results of the call made is available in the PCB status code field to the programmer. The status code should always be interrogated upon completion of a call.

If the GET call was completed as requested, the two-byte status code is blank; otherwise, the status code is one of those described later in this chapter under the heading "Status Codes for Data Language/I".

The GET NEXT Call (GNbb) - Data Base

The GET NEXT call is used to retrieve the next desired segment from the data base as described by the DBD name and sensitive segments in the PCB.

SSA's in GET NEXT calls must conform to the following rules:

1. The call may or may not have SSA's.

2. SSA's may or may not have qualification statements.

3. The first SSA may be for any level of segment, but any following SSA's must proceed down a hierarchical path with no missing intermediate levels.

4. The search field may be any field within the segment (key or data) if the organization is HISAM. If the organization is HSAM, only the last SSA may be qualified on a data field. The operator may be =, ¬=, <, =<, =>, or > . All comparisons are logical bit-by-bit compares.

The execution of a GET NEXT call without SSA's returns the next segment occurrence within the data base relative to the positioning of the data base during the previous GU, GN, or GNP call. An uninterrupted series of these call statements could be used to retrieve each segment occurrence from the data base, beginning with the first and proceeding sequentially through the last for all sensitive segments. The parameters for this form of a GET NEXT are the function, PCB name, and segment I/O work area.

The GET NEXT call progresses only forward from the position in the data base established in the preceding call, in an attempt to satisfy the current call requirements.

Status Codes for GET NEXT Calls

At the completion of a GET call, a status code indicating the results of the call made is available in the PCB status code field to the programmer. The status code should always be interrogated upon completion of a call.

If the GET call was completed as requested, the two-byte status code is blank or GA or GK; otherwise, the status code is one of those listed later in this chapter under "Status Codes for Data Language/I".

Definition of Cross-Hierarchical Boundary

The GA status code is a warning indication. When a GN or GNP call without SSA's is issued, Data Language/I may return this status code to indicate the crossing of hierarchical boundaries. This status code indicates that Data Language/I has passed from one segment in the data base at level X to another segment in the data base at level Y, where Y is less than X. In other words, it has proceeded upward in the hierarchy toward the root segment. This code is not returned to the using application program when a GU, GN with SSA's, or GNP with SSA's is issued, because the user is explicitly asking, through the presence of the SSA's, to traverse a known path in the data base. Thus the GA status code is a warning (to the user of the GN or GNP call to move sequentially through a portion of the data bases) that Data Language/I has taken him implicitly from a segment at one level of the hierarchy to a segment at another, higher, level of the hierarchy.

The GET NEXT WITHIN PARENT Call (GNPb) - Data Base

The GNP call obtains lower level segment occurrences within the family of a parent segment. It may be used to retrieve all segments or specific segments within the family of the given parent segment.

At the issuance of the first GNP call, the relevant parent is established by looking back to the last GET UNIQUE or GET NEXT call, which must have been successfully completed. No intervening ISRT calls are permitted; however, DLET or REPL calls do not affect parentage. The parentage established with a GU or GN call remains constant for successive GNP calls. However, the parentage will be destroyed whenever a GU or GN call is executed. The parent segment may be at any level in the hierarchical structure.

Note: If the GNP follows a GU or GN that returned a GE (not found) status code, no parent can be established, and the status code GP is returned for the GNP call. A GNP qualified or unqualified that results in a GE status code does not affect parentage.

SSA's in GET NEXT WITHIN PARENT calls must conform to the following rules:

1. The call may or may not have SSA's.

2. SSA's may or may not have qualification statements.

3. The first SSA may be for any level segment except root, but any following SSA's must proceed down a hierarchical path with no missing intermediate levels.

4. The search field may be any field within the segment (key or data) if the organization is HISAM. If the organization is HSAM, only the last SSA may be qualified on a data field. The operator may be =,¬=, <, =<, =>, or > . All comparisons are logical bit-by-bit compares.

If a GNP call without SSA's is repeated, this call will read all segment occurrences under the relevant parent segment, going up and down hierarchical levels and crossing boundaries in the structure beneath the parent for all sensitive segments. A not-found condition results when Data Language/I encounters the next segment occurrence that is at the same level as the parent or higher.

Status Codes for GET NEXT WITHIN PARENT Calls

At the completion of a GET call, a status code indicating the results of the call made is available in the PCB status code field to the programmer. The status code should always be interrogated upon completion of a call.

If the GET call was completed as requested, the two-byte status code is blank or GA or GK; otherwise, the status code is one of those listed in this chapter under "Status Codes for Data Language/I".

The GET HOLD Calls - Data Base

To change the contents of a segment in a data base through a DLET or REPL call, the program must first obtain the segment. It then changes its contents and requests Data Language/I to place the segment back in the data base.

When a segment is to be changed, this must be indicated to Data Language/I at the time the segment is obtained. This indication is given by using the GET HOLD calls. These function codes are like the standard GET function, except the letter H immediately follows the letter G in the code; that is, the hold form of the standard GET NEXT WITHIN PARENT (GNPb) is GHNP. There are three GET HOLD calls: GHUb, GHNb, and GHNP. They function like the standard GET calls. They also indicate to Data Language/I that the segment may be changed or deleted. (See the sections on GET UNIQUE and GET NEXT - Data Base calls and their status codes.) The HOLD forms of GET permit Data Language/I to make certain that the segment to be placed back into the data base is the same segment that Data Language/I returned on completion of the last GET HOLD call.

After Data Language/I has returned the requested segment to the user, one or more fields, but not the key field, in the segment may be changed.

The user should also guard against changing data from one type to another type; for example, binary data should not be replaced with decimal data.

After the user has changed the segment contents, he is ready to call Data Language/I to return the segment to the data base. If, after issuing a GET HOLD call, the program determines that it is not necessary to change the retrieved segment, the program may proceed with other processing.

If the user's application program intends to modify any segment within a data base record with a REPL or DLET call, all segment retrieval within that data base record should be performed with GET HOLD calls. This is true for all segment retrieval including those segments for which no modification is intended. Although retrieval by GET HOLD calls is not required for segments that are not to be implied, the above technique should result in more efficient performance.

Status Codes for GET HOLD Calls

At the completion of a GET call, a status code indicating the results of the call made is available in the PCB status code field to the programmer. The status code should always be interrogated upon completion of a call.

The actual status codes for the GET HOLD calls are the same as for the type of GET call. That is, for GHU (Get Hold Unique) see the GU status codes, for GHN (Get Hold Next) see the GN status codes, and for GHNP (Get Hold Next Within Parent) see the GNP status codes.

The INSERT Call (ISRT) - Data Base

The Data Language/I INSERT call is used for two distinct purposes. It is used to initially load the segments for creation of a data base. It is also used in the hierarchical indexed sequential organization to insert new occurrences of an existing segment type into an established data base. The processing options field in the PCB associated with the data base dictates Data Language/I execution of the call. The format of the INSERT call is identical for either use.

| In a message processing program, it is not possible to perform a HISAM load. The program to load an HISAM data base must be a Type 3 batch program. (Specifications for using the INSERT call in this type of program are provided later in this manual.)

The INSERT call may be used with other Data Language/I segment processing calls in a message processing program. In this environment, the INSERT call is used to place new occurrences of existing segment types into an established hierarchical indexed sequential data base.

When a segment is inserted into the data base, the user must tell Data Language/I precisely where the segment is to be logically placed. This placement is given to Data Language/I by referring to one or more SSA's in the call. Through the SSA's, the user tells Data Language/I the segment name and qualification statement for each segment along the path. However, the SSA for the segment to be inserted _must_ _contain_ _only_ the segment name. The name may not be followed by the character (. The path begins with the root segment and proceeds to each segment, down the hierarchical path, upon which the inserted segment depends for its full meaning.

1. The COBOL call format for inserting segments is:

   CALL 'CBLTDLI' USING function, pcbname, segment-I/O-area, ssa-1,...,ssa-n.

2. The PL/I call format for inserting segments is:

   CALL PLITDLI (parm-count, function, pcbname, segment-I/O-area,ssa-1,.....,ssa-n)

When inserting to a hierarchical sequential data base, INSERT means to load an output data base. The PCB processing option L is used. Option A is invalid for the hierarchical sequential organization. Inserts to an established hierarchical sequential data base cannot be made without reprocessing the whole file or by adding to the end, and must be in sequence.

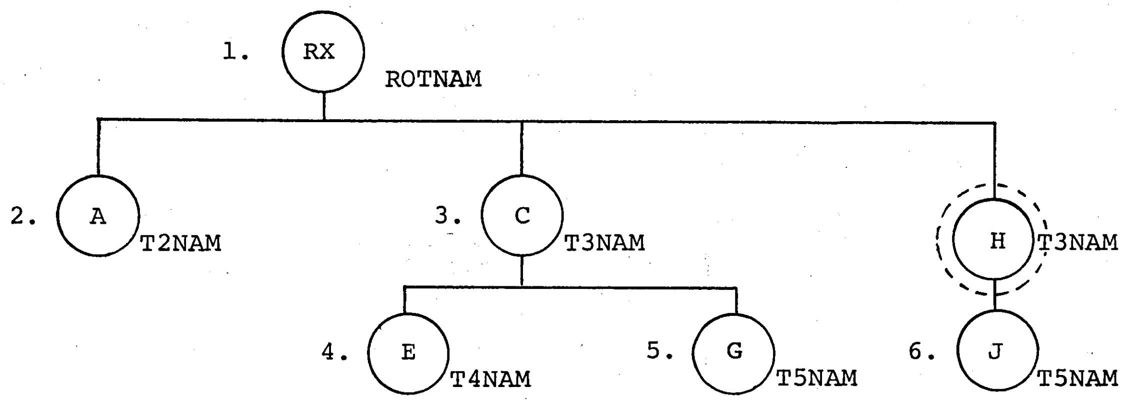
The user must follow each INSERT call in his program with statements which examine the returned status codes in the PCB, to determine if the requested action was completed properly.

When inserting a segment into a data base it is not necessary to cause the prior positioning and holding of a segment using the GET HOLD calls. The INSERT call itself contains all the qualification necessary to cause automatic positioning.

Status Codes for INSERT Calls

If the segment is inserted properly, the INSERT module places a blank status code in the PCB; otherwise, one of the following status codes will be returned to the problem programmer (see the section in this chapter titled "Status Codes for Data Language/I"). The following diagrams attempt to explain some of the more complete status codes.

EXAMPLE OF LD STATUS CODE:



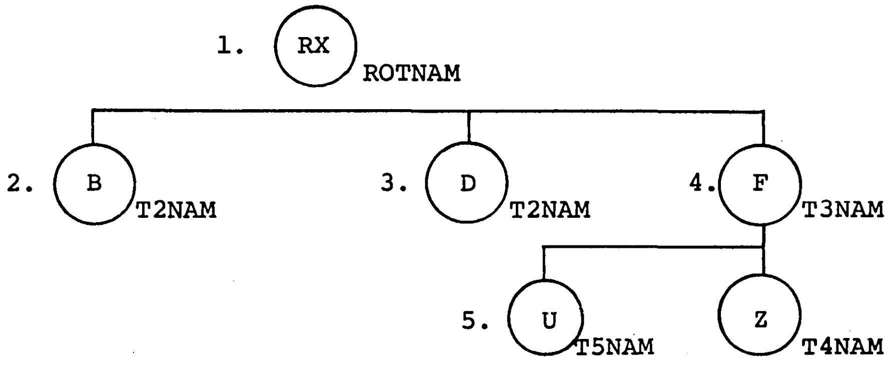
1. User loads segments RX, A, C, E, and G.

2. Next segment presented is a level three (J).

3. When segment J is presented, Data Language/I checks whether the last segment type at level two is a segment type H.

4. If the parent name (H) of the new segment (J) is not the same as the name last added at the parent level (G), status code LD is returned to the user.

EXAMPLE OF LE STATUS CODE:



1. User loads segments RX, B, D, F, and U.

2. Next segment presented is a level three (Z) with segment name T4NAM.

3. From the DBD hierarchy definition, Data Language/I finds that the previously loaded same-level segment type (U) is defined in the DBD after Z. The segments must be loaded in the same sequence as they were defined in the DBD. However, the user is attempting to add a Z after a U for a common parent F. This is the inverse of segment sequence definition in the DBD.

4. This sequencing error of common level segments thus generates status code LE.

The DELETE Call (DLET) - Data Base

To delete the occurrence of a segment from the data base, the program must first get the segment and then ask Data Language/I to delete it. When a segment has been deleted, it is no longer available to any program. Before the program can ask Data Language/I to delete a segment, however, the segment must first be obtained by issuing a GET HOLD call through Data Language/I. Once the segment has been acquired, the DELETE call may be issued.

There must be no Data Language/I calls that use the same PCB intervening between the GET HOLD call and the DELETE call; otherwise, the DELETE call is rejected. Quite often a program may want to process a segment before deleting it. This is permitted as long as the processing does not involve a Data Language/I call that refers to the same PCB used to get the segment.

Data Language/I is advised that a segment is to be deleted when the user issues a call that has the function DLET. When the DELETE call is executed, the specified segment occurrence is not physically deleted, but simply flagged as being deleted. The occurrence is physically deleted when the file is reorganized. The deletion of a parent, in effect, deletes all the segment occurrences beneath that parent. If the segment being deleted is a root segment, all dependent segments under that root in relevant data set groups are also flagged as deleted.

The segment to be deleted must occupy the area referred to by the segment-I/O-area in the DELETE call.

Note that the SSA has no meaning to the DELETE call, since positioning is accomplished by the previous GET HOLD call. SSA's should not be included in a DELETE call.

1. The delete call format for a COBOL program is:

   CALL 'CBLTDLI' USING function, pcbname, segment-I/O-area.

2. The delete call format for a PL/I program is:

   CALL PLITDLI (parm-count, function, pcbname, segment-I/O-area);

For a program that processes hierarchical sequential data bases where each record is rewritten on a new data base, the DLET call has no meaning and will be rejected as an invalid function. If a segment occurrence is to be deleted, it is simply not written to the output data base.

The REPLACE Call (REPL) - Data Base

The purpose of the REPLACE call is to allow a segment that has been retrieved through a GET HOLD call and modified through program processing, to be replaced in the data base. The segment to be modified and replaced must first be obtained by a GET HOLD call. No intervening calls involving the associated PCB may be made between the GET HOLD and the REPLACE call. If this rule is violated, the REPLACE call is rejected.

In the modification of a segment to be replaced in the data base, care must be taken not to modify the segment key field. If modification of the key field is attempted, the REPLACE call is rejected.

The segment to be replaced must occupy the area referred to by segment-I/O-area in the REPLACE call. The segment in the Data Language/I buffer area is overlaid with the segment-I/O-area in the REPLACE call.

1. The replace call format for a COBOL program is:

   CALL 'CBLTDLI' USING function, pcbname, segment-I/O-area.

2. The replace call format for a PL/I program is:

   CALL PLITDLI (parm-count, function, pcbname, segment-I/O-area);

For a program that processes hierarchical sequential data bases where each record is rewritten on a new data base, the REPLACE call has no meaning. If a segment occurrence is to be replaced, it is simply placed in the output data base with an INSERT call.

No segment search arguments are allowed on a REPLACE call.

Status Codes for DELETE/REPLACE Calls

Error codes may be generated as a result of either a DELETE or a REPLACE call, and are placed into the PCB status code field. For these status codes see the section titled "Status Codes for Data Language/I" in this chapter.

MESSAGE FORMATS AND STRUCTURES

There are three basic message formats used within IMS/360:

- Input message
- Output message
- Program-to-program messages

The formats shown represent message segments as they would be received or constructed in the segment I/O area. A message segment and a single message line are synonymous.

Input Message Format

Input message segments originate at a communications terminal and are delivered to the application program's segment I/O area by means of a GU or GN call to Data Language/I. An input message segment may be a maximum of 131 bytes for the 1050 or 2740 terminals, including the count and the halfword reserved area. An input message segment may be a maximum of 84 bytes for the 2260, including the count and halfword reserved area. An input message from a 2260 may be a maximum of 959 characters (plus the SMI symbol). The format of each input message is:

LL	ZZ	TEXT

where:

LL

is a halfword binary field containing the total number of characters in the message line, including LL and ZZ. The value of LL = number of characters in text + 4. This count entry is made by IMS/360 for input messages. When PL/I is used, this count is also placed in the dope vector's current length field segment I/O area. Further, with PL/I, the LL field is defined as a fullword but used as a halfword (length of LL for total input message is two bytes). See the section titled "Segment Input/Output Areas" in Chapter 5 of this manual.

ZZ

is a two-byte field whose value and use are reserved by IMS/360.

TEXT

is the message line exactly as it was entered at the terminal in EBCDIC. The text includes transaction code, message text, and CR (carriage return). If the message consists of multiple lines of text, each subsequent line has the same format. The message consists of an unlimited number of segments. The transaction code appears only in the first line. If a password is entered with the message from the terminal, it is edited out upon presentation to the application program, and the text is left-justified, as required. A transaction code must be followed by a blank or a left parenthesis if there is a password. These are the only two acceptable delimiters for the transaction code.

When the remote mode IBM 2260 Display Station, Model 1, is used as the input message device, the following are input message considerations:

- The length of the message is variable from 1 to 960 characters.

- The input message is broken into segments. The segments will be either 80 characters in length or a length less than 80 characters if a new-line symbol is placed in the segment by the operator. Maximum length of segments is 80 characters. (The Model 1 2260 Display Station allows 80 characters horizontally and 12 lines vertically.)

- Therefore, the same Data Language/I rules apply to obtain each segment that makes up the input message.

- Only single screen input is allowed.

- The /EXCLUSIVE command should be used when entering an input message from a 2260 Display Station. If not, any BROADCAST or system messages will be displayed on the screen and may erase an entry or an answer. When the /EXCLUSIVE command is used, the BROADCAST system messages will remain on the queue until an /END command is entered.

- <u>WARNING</u>: A START MI symbol must precede any entry of an input message. The operator of the 2260 can enter the START MI symbol from his keyboard, or the application program can place it on the display screen. (If PL/I is used, the symbol must be one character multipunched, hexadecimal 4A.) Only one START MI symbol is allowed per screen.

- The input message for a 2260 is considered to be that data contained between the START MI symbol ( ) and the position of the CURSOR symbol ( ) at the time the ENTER key is depressed. All other data displayed on the screen at this time is ignored and is not transmitted to the CPU. If no START MI symbol is displayed at the time the ENTER key is depressed, no data is sent to the CPU.

- It is recommended that, after a transaction is input, the operator should await his reply, if one is expected, before entering another transaction. This will prevent the reply from one program from overlaying the reply from another before the operator has viewed it.

<u>Output Message Format</u>

Output message segments or lines originate within the application program and are sent to a communications logical terminal, which is defined by a terminal PCB. Each output message segment is enqueued to be sent by means of an ISRT call to Data Language/I. The format of each segment is:

LL	Z1	Z2	TEXT

where:

LL

is a two-byte binary field containing the total number of characters in the message segment, including LL, Z1, and Z2. The value of LL = number of characters in text + 4. The application program must fill in this count. When PL/I is used, the LL field is defined as a binary fullword. The PL/I user must also place the length of the text to be written in this field. The values must represent the total of

2	for the count field (even though it is physically four bytes in the PL/I environment)
1	for the Z1 field
1	for the Z2 field
n	for the text length

or the text character count plus 4.

Z1

is a one-byte field containing binary zeros whose value and use are reserved by IMS/360.

Z2

For 1050 and 2740 terminals, Z2 is a one-byte field containing binary zeros whose value and use are reserved by IMS/360. For 2260 Display Stations, Z2 is a one-byte binary field that denotes the type of WRITE command back to the 2260 display screen. These types of WRITE commands affect the format of 2260 display screens. The 2848 Display Control must have the Line Addressing Feature #4787 to accomplish Items 2 and 3 below.

	WRITE Command	Description	Designation
1.	WRITE INITIAL (WI)	Indicates that it will begin writing output segment at the position at which the cursor symbol was last left	Binary zeros
2.	WRITE AT LINE ADDRESS (WALA)	Indicates that it will begin writing at the line specified (from one through twelve)	Hexadecimal 01 through 0C, depending on which of the twelve lines
3.	ERASE SCREEN START AT LINE	Indicates that the screen will be erased first; the output segment will then be written at line address specified (line one through twelve)	Hexadecimal 11 through 1C, depending on which of the twelve lines
4.	WRITE ERASE (WE)	Indicates the screen will be erased first; the output segment will then start being written on the upper left corner of the screen	Hexadecimal 20

Note: Any code not the same as designated for the WRITE commands above will default to binary zeros. No error messages will be given.

TEXT

is the output message segment in EBCDIC as it is transmitted to a specific logical terminal. The length of an output message line segment must not exceed 132 characters of text for the 1050 or 2740 terminals. It is the application programmer's responsibility to insert the required carriage-return character in the body of the message line segment. The length of an output message line segment must not exceed 960 characters of text for the 2260 Display Station. An output message can contain multiple segments. It is not necessary to include the logical terminal name in the output message, as the destination is determined by the PCB.

Certain device control characters must be inserted into the message where desired to format the message at the terminal output device. (In PL/I, these control characters must be initialized to one character and multipunched.) These are described below in hexadecimal format:

05:	skip to tab stop, but stay on same line
15:	start new line at left margin (carriage return)
25:	skip to new line, but stay at same print position

When the remote mode IBM 2260 Display Station, Model 1, is used as the output message device, the following are output message considerations:

- An output message may be composed of multiple segments that make up a single display screen (12 lines times 80 characters equals 960 characters).
- Each output segment can have its own WRITE command. However, a WRITE ERASE (WE) will be ignored except on the first output segment.
- Only single screen output is allowed.

Examples of 2260 WRITE Commands

Example 1:

Segment 1:

has LL=9, Z1= binary zeros, Z2= hexadecimal 15, and TEXT=ABCDE. From the Z2 indication, this means erase screen and start writing at line 5.

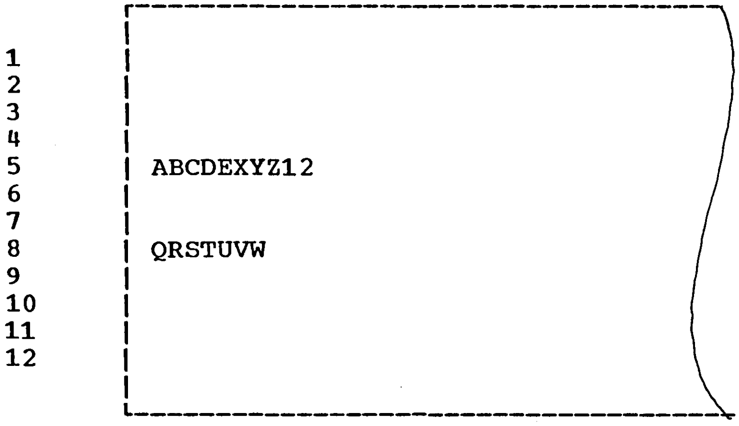
Segment 2:

has LL=9, Z1 and Z2= binary zeros, and TEXT=XYZ12. From the Z2 designation, this means continue writing TEXT from the last cursor position.

Segment 3:

has LL=11, Z1= binary zeros, Z2= hexadecimal 08, and TEXT=QRSTUVW. From the Z2 indication, this means writing TEXT at line 8.

SCREEN DISPLAY



```
1
2
3
4
5     ABCDEXYZ12
6
7
8     QRSTUVW
9
10
11
12
```

Example 2:

Segment 1:

has LL=11, Z1= binary zeros, Z2= hexadecimal 20, and TEXT=123456▲. Z2 indicates that the screen shall be erased, and writing will start at the top left corner, ending the text with a new-line symbol.

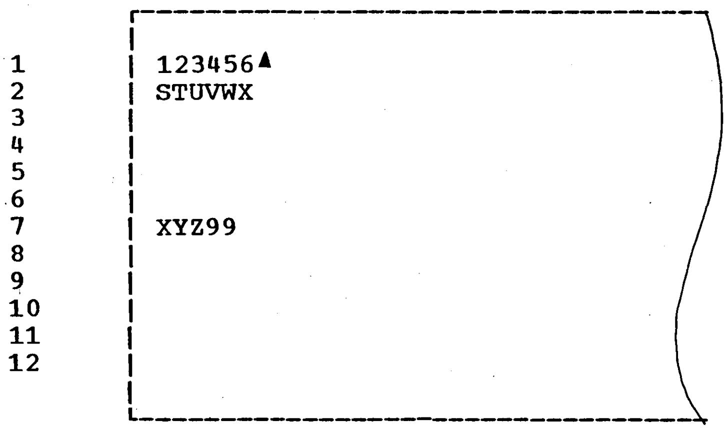
Segment 2:

has LL=10, Z1=binary zeros, Z2= hexadecimal 20, and TEXT=STUVWX. Z2 indicates that the screen should be erased and writing should start at the top left corner of the screen, but, since there has already been a WRITE ERASE, this command will be ignored, putting the TEXT on the second line.

Segment 3:

has LL=9, Z1= binary zeros, Z2= hexadecimal 17, and TEXT=XYZ99. Z2 indicates that the screen should be erased (WRITE ERASE) and TEXT placed on line 7. Since this is not the first segment command in this message, the WRITE ERASE will be ignored and the TEXT placed on line 7.

SCREEN DISPLAY



```
1     123456▲
2     STUVWX
3
4
5
6
7     XYZ99
8
9
10
11
12
```

A 2260 application program done in PL/I is included as an example of a message program in the Appendix of this manual.

Program-to-Program Message Switching Format

This facility has been included to allow messages to be sent from one program to another. The format of this type of message is similar to that for output messages. It is:

LL	Z1	Z2	TEXT

The description for output messages applies here. The following areas should be noted:

1. LL is the same as for output message.

2. Z1 and Z2 are one-byte fields.

3. TEXT: The destination of a program-to-program message is a control block in the IMS/360 scheduling facility. This control block has a one- to eight-character transaction code. The transaction name must appear as the initial characters of the text, followed by a blank. A transaction name is required to enable reconstruction of message queues during restart.

Notes:

1. The Data Language/I ISRT call which processes this output message segment must refer to a PCB whose output terminal name is equal to the eight-character transaction code referenced in statement 3.

2. Message security (password or terminal) is not available for program-to-program messages.

There are some limitations on program-to-program message switching with regard to data base recovery. The reader should refer to the APPLCTN macro-instruction in Chapter 4 of the IMS/360 Operations Manual, Volume I - Systems Operation (SH20-0635) for a description of the limitations.

CALL Definitions for Messages

The call functions that are available when referencing input and output message are:

1. 'GUbb'
2. 'GNbb'
3. 'ISRT'

The use of these functions is limited when compared with the functions used to reference data bases. In all three of the functions related to messages, Data Language/I works only with message lines or segments. The call format is standard and simple because there is no hierarchical structure with which to be concerned. SSA's must not be used for Data Language/I message calls.

The GET Calls (GU, GN)

The retrieval of an input message is accomplished with GU and GN calls. When a message processing program is scheduled, the program knows that there is an input message to be processed. The structure of the terminal PCB is described in the message processing program, but the actual PCB values are furnished by IMS/360. The first segment or line (commonly called the "header segment") of an input message is always obtained by a GET UNIQUE call against the input PCB, which is provided by IMS/360 at scheduling time.

1. The format for a COBOL program is:

   ```
   CALL 'CBLTDLI' USING GET-UNIQUE-FUNC, IN-TERM-PCB,
   SEGMENT-IO-AREA.
   ```

2. The format for a PL/I program is:

   ```
   CALL PLITDLI (PARM_COUNT,GET_UNIQUE_FUNC,IN_TERM_PCB,
   SEGMENT_IO_AREA);
   ```

The exact message entered at the terminal is placed in the SEGMENT-IO-AREA. Since the maximum length of a message can vary, this area should normally be 136 bytes long.

For programs that process multiple message types, the text of the input message must be examined to determine the transaction code.

For input messages with multiple segments or lines, the GN call must be used to get the subsequent segments (commonly called "trailer segments") of an input message.

a. The format for a COBOL program is:

   ```
   CALL 'CBLTDLI' USING GET-NEXT-FUNC, IN-TERM-PCB, SEGMENT-IO-AREA.
   ```

b. The format for a PL/I program is:

   ```
   CALL PLITDLI (PARM_COUNT,GET_NEXT_FUNC,IN_TERM_PCB,
   SEGMENT_IO_AREA);
   ```

When a message includes trailer segments, two successive GU calls will cause the subsequent segments to be lost.

When a GU call is given and no more messages are available, a "QC" code is returned in the input terminal PCB status code field. A status code of "QD" is returned with a GN call following the last subsequent segment of an input message.

The INSERT Call (ISRT)

The INSERT call is used to send output messages to terminals. If a message is to be sent back to the terminal from which the input message originated, the input PCB should be used. If output messages are to be sent to terminals other than the input terminal, they must be predetermined and specified with terminal PCB's at PSB generation time. The designation for an output message is determined by the PCB used in the Data Language/I INSERT call; the PCB that is addressed contains an eight-character logical terminal name or transaction code. If the insert is to an SMB, the inserted message must contain the message destination in its high-order bytes, followed by at least one blank. If the insert is to a CNT, this is not a requirement.

The call format is similar to the message GET calls because there is no hierarchical structure to consider.

1. The format for a COBOL program is:

   ```
   CALL 'CBLTDLI' USING INSERT-FUNC, TERM-PCB, SEGMENT-IO-AREA.
   ```

2. The format for a PL/I program is:

```
CALL PLITDLI (PARM_COUNT,INSERT_FUNC,TERM_PCB,SEGMENT_IO_AREA);
```

Output message segments are not distinguishable as headers and trailers by the ISRT call of IMS/360. If a distinction must be made, the programmer must do so. SSA's may not be used with message ISRT calls. All message segments inserted to a given output terminal (terminal PCB) during the processing of a single input message are treated by IMS/360 as a single message. Output message segments may be enqueued by INSERT calls for output prior to trailer segment retrieval of the input message. Output message segments are not sent to their destination (terminal) until the application program issues a GU for another input message or returns to IMS/360 on conclusion of its processing.

Status Codes for Input and Output Messages

At the completion of a GET or INSERT call related to a message segment, a return code indicating the results of the call is available to the programmer in the associated PCB status code field. The status code always should be interrogated at the completion of a call.

If a GET or INSERT call is completed successfully, the two-byte status code is blank; otherwise, the status code is one of those listed in this chapter under "Status Codes for Data Language/I".

STATUS CODES FOR DATA LANGUAGE/I CALLS

	DATA BASE CALLS						MSG CALLS						
STATUS	GU	GN	GNP	DLET	ISRT	ISRT				CALL	ERROR	I/O OR	
CODE	GHU	GHN	GHNP	REPL	(LOAD)	(ADD)	GU	GN	ISRT	COMPLETED	IN CALL	SYST.ERROR	DESCRIPTION
AB	X	X	X	X	X	X	X	X	X		X		SEGMENT I/O AREA REQUIRED, NONE SPECIFIED IN CALL
AC	X	X	X		X	X					X		HIERARCHICAL ERROR IN SSA'S
AD											X		INVALID FUNCTION PARAMETER
AE			X								X		ROOT SEGMENT SPECIFIED BY THIS CALL, NOT ALLOWED GNP CALLS
AF				X							X		DLET OR REPL CALLS CANNOT HAVE SSA'S SPECIFIED
AG	X				X	X					X		FIRST SSA SPECIFIED IS NOT LEVEL 1
AH	X				X	X					X		CALL REQUIRES SSA'S. NONE PROVIDED
AI	X	X	X	X	X	X						X	DATA MANAGEMENT OPEN ERROR
AJ	X	X	X		X	X					X		INVALID SSA QUALIFICATION FORMAT
AK	X	X	X		X	X					X		INVALID FIELD NAME IN CALL
AL	X	X	X	X	X	X					X		CALL USING TERM PCB IN TYPE 3 (BATCH)
AM	X	X	X	X	X	X					X		CALL FUNCTION NOT COMPATIBLE W/ PROCESSING OPTION
AN		X	X								X		GN CALL FOLLOWING ISRT CALL IS INVALID
AO	X	X	X	X	X	X						X	I/O ERROR ISAM OR BSAM
AP	X	X	X	X	X	X			X			X	I/O ERROR OSAM
AQ							X	X				X	READ I/O ERROR. MESSAGE CHAIN CANNOT BE FOLLOWED. MINIMUM OF ONE MESSAGE LOST
AR							X	X				X	READ I/O ERROR. MESSAGE SEGMENT HAS BEEN LOST. MESSAGE CHAIN IS STILL INTACT.
AS							X	X				X	QUEUES NOT AVAILABLE
AT									X		X		TRANSACTION CODE DOES NOT MATCH PCB NAME IN PGM-TO-PGM MSG SWITCH
DA				X							X		SEGMENT KEY FIELD HAS BEEN CHANGED
DJ				X							X		NO PRECEDING SUCCESSFUL GET HOLD CALL
GA		X	X							X			CROSSED HIERARCHICAL BOUNDARY INTO HIGHER LEVEL * (RETURNED ON UNQUALIFIED CALLS ONLY)
GB		X											→ END OF DATA SET. LAST SEGMENT REACHED.
GE	X	X	X			X							→ SEGMENT NOT FOUND
GK		X	X							X			DIFFERENT SEGMENT TYPE AT SAME LEVEL RETURNED (RETURNED ON UNQUALIFIED CALLS ONLY)
GP			X								X		A GNP CALL AND NO PARENT ESTABLISHED, OR REQUESTED SEGMENT LEVEL NOT LOWER THAN PARENT LEVEL
II						X							SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE
LB					X								SEGMENT TO INSERT ALREADY EXISTS IN DATA BASE
LC					X								KEY FIELD OF SEGMENTS OUT OF SEQUENCE
LD					X								NO PARENT FOR THIS SEGMENT HAS BEEN LOADED
LE					X								SEQUENCE OF SIBLING SEGMENTS NOT THE SAME AS DBD SEQUENCE
QC							X						NO MORE INPUT MESSAGES
QD								X					NO MORE SEGMENTS FOR THIS MESSAGE
QE								X			X		GET NEXT REQUEST BEFORE GET UNIQUE
QF									X		X		SEGMENT LESS THAN FIVE CHARACTERS (SEG LENGTH IS MSG TEXT LENGTH PLUS FOUR CONTROL CHARACTERS)
QH									X		X		TERMINAL SYMBOLIC ERROR - OUTPUT DESIGNATION UNKNOWN TO IMS/360 (LOGICAL TERMINALS OR TRANSACTION CODE)
QI								X			X		GET NEXT AFTER END OF MESSAGE
BB MEANING BLANK BLANK	X	X	X	X	X	X	X	X	X	X			GOOD! NO STATUS CODE RETURNED. PROCEED!

* SEE PARAGRAPH ON CROSS-HIERARCHICAL BOUNDARY DEFINITION IN IMS/360 PDM

INPUT MESSAGE EDITOR

The input message editor is not supplied as a part of the IBM IMS/360 program.

The variable-length, variable format of data in an input message is difficult for a program written in COBOL to manipulate. The solution to the problem may be a message-editing subroutine written in Assembler Language. The user of IMS/360 can construct such a subroutine.

An input editor can accept a variable-content character string, edit that string, and produce an output composed of a fixed number of fixed-length fields. The input editor can be designed to process input messages from online terminals, or to edit any desired character string. In the following discussion, the terms character string and message are used interchangeably.

The input editor is invoked by a standard subroutine call with three parameters - the name of the input area, the name of the edit table, and the name of the output area. The calling program may be written in either COBOL or PL/I.

The input editor processes all the separate fields in a message. These fields must be separated by characters that are defined as delimiters for this message. Within the message, all fields except the first must begin with a delimiter, which means, effectively, that all fields except the last must end with a delimiter.

The following are the two distinct types of fields which the input editor recognizes and extracts from messages. These two types of fields can be used to construct three types of messages.

- Positional fields - those fields which should always occur in an input message and which will have meaning to the application program because of their relative positioning within the message.

- Fields with keywords - those fields which may or may not occur in a specific input message; with an occurrence, such a field is accompanied by an identifying keyword. The keyword must immediately follow the delimiter that defines the left end of the field. For example, PNO= could be the keyword for part number, and the message entry might be PNO=12345. The combination of keyword and field data is considered as one field.

The first type of message recognized and processed by the input editor contains all positional fields. To the input editor, this type of message contains data fields in a specified order and separated by delimiters. Positional fields should always be present in the character string being edited and must be in the order specified in the edit table.

The second type of message contains a mixture of positional fields and fields with keywords. All positional fields must occur in the first part of the message, and the fields with keywords must occur in the second part of the message.

The third type of message is characterized as one in which all the fields have keywords. With IMS/360 this is feasible, since the positional transaction code may not be a part of the character string that is edited.

The input editor goes through the submitted character string, one field at a time, and compares each field to the fields specified in an edit table.

The input editor subroutine could be invoked by an application program with a CALL statement using the following parameters. Within the call, the parameters should be given in the same order as in the following list:

- The name of the area containing the input string to be edited. Each input string should have a binary word appended to the front of the text. This word has the same format as the first word in an input message from a remote terminal. The appended binary word is not considered part of the text during editing.

- The name of the edit table to be used in editing.

- The name of the output area that will contain the edited data at the completion of the execution of the subroutine. Positional offsets in the output area are given by the field entries in the edit table and determine the positions of the edited data.

- A complete input editor example is in the appendix.

The Edit Table

Three kinds of entries comprise the edit table. Each of these entries has multiple coding statements. The first entry is the table header entry and contains data concerning later entries in the edit table. This entry occurs only once. The second kind of entry, is the delimiter entry which defines a character string as a delimiter between fields. There may be multiple delimiter entries in an edit table. The third kind of entry is a field entry. This entry identifies a field and specifies the kind of editing that is to be done on it. There must be a field entry for each field in the string to be edited. The three components of the edit table are discussed in the sequence in which a programmer would formulate his coding.

Delimiter Entry

The first statement in a delimiter entry is the name of the entry. The second statement specifies the length of the delimiter, which can be any number between 1 and 255 and is specified in a binary halfword. The next field specifies the delimiter itself. A final field may be required because each delimiter entry must contain an even number of bytes, and all delimiter entries must be the same length.

The following are examples in COBOL of delimiter entries:

```
02 DELIMITER-ENTRY-1.
     03  LENGTH PICTURE S999   COMPUTATIONAL   VALUE 1.
     03  DELIMITER  PICTURE X  VALUE ';'.
     03  FILLER  PICTURE X.
```

In this case, the filler is needed to make the entry an even number of bytes long.

```
02 DELIM-1.
     03  LENGTH  PICTURE S999   COMPUTATIONAL   VALUE 4.
     03  DELIMITER  PICTURE XXXX   VALUE 'PNO='.

02 DELIM-2.
     03  LENGTH  PICTURE S999   COMPUTATIONAL   VALUE 1.
     03  DELIMITER  PICTURE X,  VALUE ','.
     03  FILLER  PICTURE XXX.
```

Note that DELIM-1 is an even length without filler and DELIM-2 is padded to the same length.

Field Entry

A field entry relates to one field in the input string that is to be edited. There must be a field entry in the edit table for each field that is to be edited in the input string. All field entries in the edit table must be the same length, and this length must be an even number. The first statement in the field entry is the name of the entry, and the following data items must be specified in order:

- A one-byte character item which specifies whether or not leading fill characters are to be deleted from this field. The value of this item must be Y for yes and N for no.

- A one-byte item which identifies the leading fill character if the first item is Y. If the first item is N, this item is blank. If the fill character exists inside the data field, it will not be removed from there.

- A one-byte item which specifies whether or not trailing fill characters are to be deleted from this field. The value of this item must be Y for yes and N for no.

- A one-byte item which identifies the trailing fill character if the third item is Y. If the third item is N, this item is blank. If the fill character exists inside the data field, it will not be removed from there.

- A binary halfword which specifies the minimum length of the field that is to be edited.

- A binary halfword which specifies the maximum length of the field that is to be edited. The maximum length must be greater than or equal to the minimum length. The maximum length is also the size of the edited field in the output string.

- A binary halfword which specifies the offset of starting position of the edited field, that is, the leftmost position of the edited field, in the output area. The output area can be a number between 1 and 32,767.

- A one-byte item which specifies whether the field is right- or left-justified in the output string. The value of this item is R for right and L for left.

- A one-byte item which holds the field edit status code after the execution of the string editor. This item should be initialized with the value of zero and may contain the following values after editing:

  0 Field entry was not used for editing

  1 Valid field with no errors

  2 Null field

  3 Length of field less than minimum size

  4 Length of field greater than maximum size

  5 Invalid type of data in field

- A one-byte item which specifies the action to be taken with respect to the output string when the edited field is valid. The code N indicates that no action is to be taken; F signals the string editor to fill the output field with the valid output fill character; M signals the string editor to move the valid field to the output field in the output string; and Y indicates that the valid field should be moved and padded to the output field size with the valid output fill character.

- A one-byte item which specifies the valid output fill character.
- A one-byte item which specifies the action to be taken with respect to the output string when the edited field is invalid. The code N indicates that no action is to be taken; F signals the string editor to fill the output field with the invalid output fill character; M signals the string editor to move the invalid field to the output field in the output string; and Y indicates that the invalid field should be moved, properly justified, and padded to the output field size with the invalid output fill character. If there is a field error type 4 (the field is too large) and the action code is M or Y, the input editor will move only as many characters as the output field area will hold.
- A one-byte item which specifies the invalid output fill character.
- A binary halfword which specifies the length of the keyword for this field. If this is a positional field and therefore has no keyword, the value of this halfword must be zero. If there is a keyword, its length must be between 1 and 255 bytes.
- A binary halfword which specifies the number of check characters and symbols used in the edit to determine the type of the field. The value of this item must be a number from 0 to 30.
- An item of the length specified in the 14th entry that contains the value of the keyword for this field. The keyword for a field may be any string of characters. If a positional field is being described, this item will not exist in the edit table field entry.
- An item of the length specified in the 15th entry that contains the check characters which are used in the edit to determine the type of the field. The characters in the field should be the type indicated. The following check characters may be used for their stated purposes:

A	Alphabetic check, from A to Z
N	Numeric check, from 0 to 9 without signs
Z	Zoned decimal, from zero to 9, with an optional sign in the rightmost byte
P	Packed decimal
B	Blank
E	Extended alphabetic, A to Z, plus #, $, &

In addition to these check characters, it is possible to include a check for special characters. This is accomplished by placing an S in this entry and following the S with the desired special characters. The characters following the S are interpreted literally and must be the last entries in this item.

A "not" symbol may be used in checking the field. When the not symbol occurs in this entry, the field is edited to determine that none of the specified checks following the not symbol are satisfied. Special characters may be used after an S following a not symbol; however, it is not possible to check a single field for both the existence and absence of special characters. When special characters are used following a not symbol, those characters must be in ascending EBCDIC order.

The field is edited one character at a time. Multiple test specifications are "or" conditions. If all characters of the field pass any of the specified tests, the test of the field is successful. However, if any one character in the field fails all the specified tests, the whole field fails the test.

Figure 38 is a COBOL example of a field entry. The field numbers on the left correspond to the previous discussion.

```
Field
No.

       02  FIELD-ENTRY-1.
1          03  DELETE-LEAD-FILL-CHAR  PICTURE X  VALUE 'Y'.
2          03  LEADING-FILL-CHAR  PICTURE X  VALUE ' '.
3          03  DELETE-TRAIL-FILL-CHAR PICTURE X  VALUE 'Y'.
4          03  TRAILING-FILL-CHAR PICTURE X  VALUE ' '.
5          03  MINIMUM-FIELD-LENGTH   PICTURE S999
                               COMPUTATIONAL  VALUE 4.
6          03  MAXIMUM-FIELD-LENGTH   PICTURE S999
                               COMPUTATIONAL  VALUE 10.
7          03  OUTPUT-START    PICTURE S999   COMPUTATIONAL
                               VALUE 7.
8          03  OUTPUT-JUSTIFICATION    PICTURE X   VALUE 'R'.
9          03  FIELD-EDIT-STATUS-CODE  PICTURE X   VALUE '0'.
10         03  VALID-OUTPUT-ACTION-CODE   PICTURE X  VALUE 'Y'.
11         03  VALID-OUTPUT-FILL-CHAR  PICTURE X   VALUE ' '.
12         03  INVALID-OUTPUT-ACTION-CODE PICTURE X  VALUE 'F'.
13         03  INVALID-OUTPUT-FILL-CHAR   PICTURE X  VALUE '*'.
14         03  LENGTH-FIELD-KEYWORD    PICTURE S999
                               COMPUTATIONAL  VALUE 3.
15         03  NUMBER-OF-CHECK-CHARS   PICTURE S999
                               COMPUTATIONAL  VALUE 6.
16         03  FIELD-KEYWORD   PICTURE XXX VALUE 'PNO'.
17         03  CHECK-CHARACTERS   PICTURE X(6)  VALUE 'APZS*/'.
18         03  FILLER  PICTURE X.
```

Figure 38. Field entry example

Edit Table Header Entry

The edit table header contains information on the edit table itself and on the editing process. The header must be the first component in an edit table. The following fields comprise the edit table header:

- A binary fullword with an initial value of zero. This entry is reserved for use by the editor and is required so that PL/I with its dope vectors can be distinguished from other languages.

- A one-byte item which specifies the calling language that will involve the input editor. This item must have a value of C for COBOL or a value of P for PL/I.

- A one-byte item which specifies the type of audit desired. At the present time, this field should be initialized to N to indicate no audit trail.

- A one-byte item that has a value of Y for yes and N for no to indicate whether the field feedback areas in each field entry are to be set to zero before editing.

- A one-byte item that contains the edit table status code. This item should have an initial value of zero. Upon return from the input editor to the calling program, this item will have one of the following values:

  0 The table has not been used for editing

  1 The input string was edited successfully without error

  2 The input string was null

  3 The input string was too short

  4 The input string was too long

  5 The input string had a recognizable field keyword

  6 Invalid fields encountered in the input string, but no errors 2 through 5 above

  * Invalid edit table

- A binary halfword that specifies the length of the table header. This entry must have a value of 28.

- A binary halfword that specifies the starting position in the input string at which the editing is to begin.

- A binary halfword that indicates the number of delimiter entries existing in the edit table. This number must be positive.

- A binary halfword that indicates the length of each of the delimiter entries in the table. All delimiter entries must be the same length, and this length must be an even number. Therefore, the length of a delimiter entry in the table is the length of the longest delimiter, plus 2 for the halfword that contains the length of the delimiter, then rounded up to the nearest even number. Short delimiter entries must have filler at the end to maintain a uniform length.

- A binary halfword entry that contains the number of field entries in the edit table. This entry must have a positive value.

- A binary halfword that specifies the length of each field entry in the table. All field entries must be the same length, and this length must be an even number. Therefore, the length of a field entry in the table is the length of the longest field entry, rounded up to the nearest even number. Short field entries must have filler at the end in order to maintain a uniform field entry length.

- A binary halfword that is filled by the input editor to indicate the number of valid positional fields that were edited.

- A binary halfword that is filled by the input editor to indicate the number of invalid positional fields that were found.

- A binary halfword that is filled by the input editor to indicate the number of valid fields with keywords that were edited.

- A binary halfword that is filled by the input editor to indicate the number of invalid fields with keywords that were found.

Figure 39 is a COBOL example of an edit table header. The field numbers on the left correspond to the previous discussion.

```
Field
No.

      02  TABLE-HEADER.
1         03  EDITOR-RESERVE  PICTURE S9(5)    COMPUTATIONAL
                    VALUE 0.
2         03  LANGUAGE    PICTURE X   VALUE 'C'.
3         03  AUDIT-TRAIL-CODE    PICTURE X   VALUE 'N'.
4         03  FIELD-FEEDBACK-RESET    PICTURE X   VALUE 'Y'.
5         03  EDIT-TABLE-FEEDBACK PICTURE X   VALUE '0'.
6         03  TABLE-HEADER-LENGTH PICTURE S999    COMPUTATIONAL
                    VALUE 28.
7         03  EDIT-START-POSITION PICTURE S999    COMPUTATIONAL
                    VALUE 1.
8         03  NUMB-DELIMITER-ENTRIES  PICTURE S999
                    COMPUTATIONAL   VALUE 3.
9         03  LENGTH-OF-A-DELIM-END   PICTURE S999
                    COMPUTATIONAL   VALUE 6.
10        03  NUMBER-OF-FIELD-ENTRIES PICTURE S999
                    COMPUTATIONAL   VALUE 1.
11        03  LENGTH-OF-A-FIELD-END   PICTURE S999
                    COMPUTATIONAL   VALUE 30.
12        03  NO-OF-VALID-POSTL-FLDS  PICTURE S999
                    COMPUTATIONAL   VALUE 0.
13        03  NO-OF-INVALID-POSTL-FLDS    PICTURE S999
                    COMPUTATIONAL   VALUE 0.
14        03  NO-OF-VALID-FLDS-W-KEYS PICTURE S999
                    COMPUTATIONAL   VALUE 0.
15        03  NO-OF-INVALID-FLDS-W-KEYS     PICTURE S999
                    COMPUTATIONAL   VALUE 0.
```

Figure 39. Edit table example

TERMINATION OF AN APPLICATION PROGRAM

At the completion of processing of any application program (message or batch), control must be returned to the region controller. The RETURN statement must be given in every program as follows:

COBOL

```
ENTER LINKAGE.

RETURN.

ENTER COBOL.
```

PL/I

```
RETURN;
```

ASSEMBLER

```
RETURN (14,12),RC=0
```

The return statement in a message processing program causes control to return to the region controller. The region controller records accounting information and passes control to IMS/360 for a request for a rescheduling.

The return statement in a batch-type program also returns control to the region controller. However, the region controller subsequently returns control to the Operating System/360 job terminator after Data Language/I resources are released.

The return statement from an application program written in Assembler Language should contain Register 15 equal to zero if the program is terminating normally.

MESSAGE PROCESSING REGION SIMULATION

Message processing region simulation is not supplied as a part of the IBM IMS/360 program.

The checkout of any message processing program in the online terminal environment is often impractical. To enable a more practical and efficient checkout environment, a message processing region simulation might be used. The object of the simulator would enable checkout of a message processing program in a Type 3 processing region with a set of test data bases. Messages could be read and written with unit record, tape, or disk data sets as opposed to input and output message queues. To be effective the simulator should incur no or minimal change to the message processing program when it is moved from the simulated to the actual message processing region environment.

Simulation is accomplished by appending two modules to the message processing program in addition to the language interface (Simulator Interface A and Simulator Interface B, Figure 40).

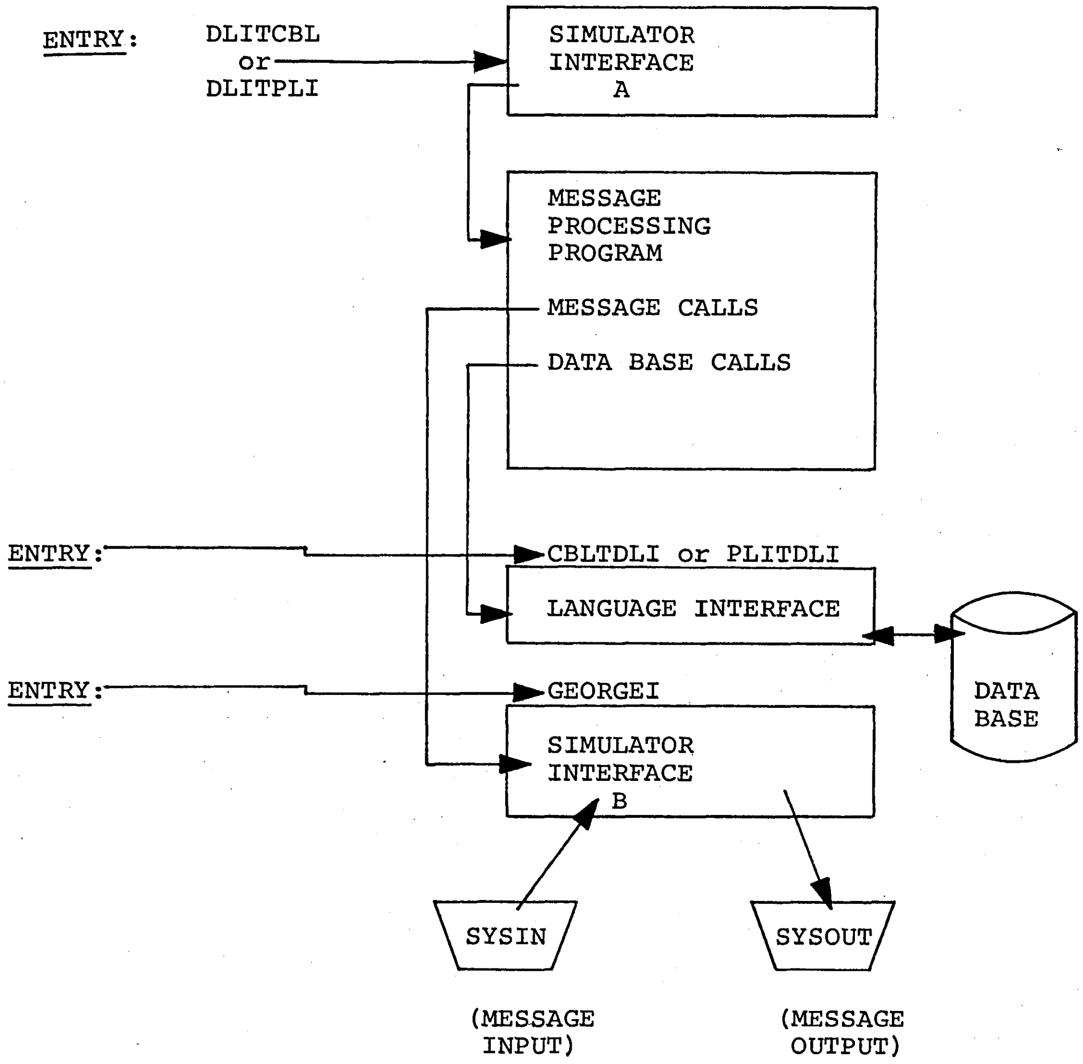


Figure 40. Message processing region simulation

When the PSB is generated for the associated message program, the PCB's within the PSB would normally be for Data Language/I data bases only. No PCB for an input and output terminal is provided. When the message program is loaded into a Type 3 processing region, the PCB addresses are passed to the message program. No terminal PCB is provided.

When Simulator Interface A is link-edited with the message program, with entry point DLITCBL or DLITPLI, the Simulator Interface A is entered. The interface prefixes the PCB address list with an input/output terminal PCB address. The terminal PCB exists within Simulator Interface A, and its address is added as the first PCB address in the PCB address list passed to the message program. This PCB address is used by the message program just like the other PCB addresses in the list, except that this PCB address is used in calls from the message program to Simulator Interface B.

When a call is made from the message program to Simulator Interface B, the message program makes a Data Language/I call with the terminal PCB address provided by Simulator Interface A. Simulator Interface B then utilizes Operating System/360 SYSIN and SYSOUT data sets as if messages were being read from and written to message queues. The user may include alternate terminal PSB's within his PSB generation. The addresses for these PCB's are provided upon entry to the user message program in the order specified by PCB cards in PSB generation. If a Data Language/I call (CALL CBLTDLI) is issued with an alternate terminal PCB address in an IMS/360 Type 3 region, an AL status code is returned in the PCB.

Data Language/I data base calls are executed with the appropriate PCB's to link-edited language interface.

The following changes must be made when the message processing program is moved to a Type 1 processing region:

- Both Simulator Interface modules should be omitted.
- The entry point name of the message program must be renamed DLITCBL (COBOL or Assembler) or DLITPLI (PL/I).
- The call statement operand must be renamed from GEORGEI to the language interface entry point CBLTDLI or PLITDLI.

The appendix to this manual includes examples of the simulator modules.

PROCESSING REGION ABENDS

Comp. Code	Issuing Component	Explanation
0004	DFSIRC00	An attempt was made to initiate an IMS/360 Type 1 or Type 2 processing region when the IMS/360 control program (Region Type 0) was not active in the Operating System.
0016	DFSIRC00	The IMS/360 region control program was unable to complete initiation of a Type 1 or Type 2 processing region. The addition of another region to the number then executing would have exceeded the value specified in the MAXTASK operand of the IMSCTRL macro-instruction at IMS/360 system definition time.
0032	DFSIRA00	PARM field was omitted from the EXEC statement. PARM field controls type of execution. (See Chapter 4 of the IMS/360 Operations Manual, Volume I - Systems Operation for explanation.)
0036	DFSIRA00	Program (PSB) name was omitted from the PARM field on the EXEC statement of an IMS/360 Type 2 or 3 region (batch).
0040	DFSIRA00	PARM field of EXEC card is invalid format for Type 2 or 3 IMS/360 region. Comma does not follow first positional parameter.
0044	DFSIRA00	PARM field of EXEC card specifies an invalid region-type code.

0048	DFSIRA00	PARM field of the EXEC card contains an excessive number of positional parameters.
0052	DFSIRA00	First character of a positional parameter in PARM field of the EXEC card is blank or invalid.
0056	DFSIRA00	A positional parameter in the PARM field of the EXEC card exceeds maximum allowable length.
0060	DFSIRA00	A required positional parameter is omitted from the PARM field of the EXEC card.
0064	DFSIRC00	Dispatching priority of a message partition running in an MFT-II environment is higher than that of the IMS/360 control program (Type 0 region).
0150	DFSIDBA0	PCB address passed in the USING list of a Type 3 batch program is not the same as any passed to the program by IMS/360 at first entry. The PCB referred to in the CALL statement may not have been defined at PSBGEN time. The USING list of the CALL statement may be improperly constructed.
0151	DFSIDBA0	USING list of CALL statements in a Type 3 batch program is truncated at the function position. There is no PCB address in the call. Call list has only one entry.
0200	DFSIDLK0	The available dynamic main storage in the Operating System/360 region or partition in which a Type 1, 2, or 3 region is operating is not sufficient to allow the Data Language/I block loader to fetch the required PSB's and DBD's. Increase region size.
0201	DFSIDLK0	PSB loaded in the application program processing region has invalid or inconsistent processing options specified. Check PSB generation.
0202	DFSIDLK0	The data bases named at PSBGEN do not agree with those specified for the same PSB name at IMS/360 online system definition. Check IMS/360 online Stage 1 DMB directories and PSB generation.
0203	DFSIDLK0	The first defined segment in DBD is not a root segment. Register 2 points to the DBD. Add 8 to the contents of Register 10. This points to the segment name in question. Check DBD generation.
0204	DFSIDLK0	Error in implied hierarchical definition of sensitive segments in PSB. Register 10 points to segment name at which error was discovered. Check both DBD and PSB generation for conflicting definitions.
0206	DFSIDLK0	Unable to open PSB and DBD libraries. Check proper allocation for DD name IMS/360 in JCL for Type 1, 2, or 3 region.

0208	DFSIDLK0	A sensitive segment is named in PSBGEN for which no corresponding segment name was defined in the associated DBDGEN. Register 3 at ABEND points to the unmatched sensitive segment name. Register 9 points to the DBD name. Register 8 points to the data base PCB name in the PSB. Check PSB and DBDGEN.
0209	DFSIDLK0	DBD specifies an unsupported or unknown access method. Register 8 or register 4 points to the DBD name. An offset of 8 from the address pointed to by register 11 is the specific DCB within the DBD which is in error. Check DBD generation.
0210	DFSIDLK0	System error. DBD does not contain a DCB type required to construct the DMB. DCB type required is pointed to by register 3. An offset of 12 from register 2 points to the first DCBTAB in the group of DCB's examined.
0211	DFSIDLK0	System error. DSB (SEGM) is followed by more than one key (FLDK) definition. Register 11 points to FLDTAB, register 6 to SDB, and register 7 to FDB in error. Register 2 points to DBD.
0212	DFSIDLK0	System error. The first FDB is not the key FDB (FLDK) definition, yet physical codes for field and SDB are equal. Register contents same as 0211.
0213	DFSIDLK0	System error. SDB has no key field defined. SEGM statement not followed by FLDK or FLD statement. Register contents same as for 0211.
0229	DFSIBDP0	Cannot find key field for a segment in the DBD. Will appear as a message region ABEND on the master terminal.
0240	DFSIPC00	Message processing application exceeded allowable execution time in a Type 1 message region. See Chapter 3 of the <u>IMS/360 Operations Manual, Volume I - Systems Operation</u> under the heading "TRANSACT Macro-Instruction" for a further explanation.
0260	DFSIPR00	Number of parameters (data items named in USING list) in the application program call exceeds the allowable limit.
0261	DFSIPR00	One of the values passed in the USING list of the application program Data Language/I call is invalid. It either exceeds object machine size, does not meet alignment requirements, or violates storage protection boundaries.
0404	DFSIPR00	During execution of a Type 1 message processing program or a Type 2 batch program, the IMS/360 control program (Type 0 region) terminated abnormally.
0408	DFSIPR00	During execution of a message processing application or a Type 2 batch program, an invalid event control block address was

		passed to the IMS/360 interregion communication SVC.
0424	DFSIPR00	System error. A message or Type 2 batch region has been activated asynchronously because of an error in the IMS/360 control program (Type 0 region). System error during the application program communication cycle of the program request handler.
0428	DSFIAS00	A Type 2 batch step could not be initiated because the program named in the second positional operand of the PARM field was not defined at system definition time.
0432	DFSIAS00	A Type 2 batch step could not be initiated because the program named in the second positional operand of the PARM field was not defined as a Type 2 program at system definition time.
0436	DFSIAS00	A Type 2 batch step could not be initiated because the input symbolic queue named in the fourth positional operand of the PARM field was not defined at system definition time. Check PARM field to ensure that input symbolic name is correct.
0440	DFSIAS00	A Type 2 batch step could not be initiated because the input symbolic queue named in the fourth positional operand of the PARM field was a logical terminal name. It may only be a transaction code.
0444	DFSIAS00	A Type 2 batch step could not be initiated because the output symbolic queue named in the fifth positional operand of the PARM field was not defined at system definition time. Check PARM field to ensure that output symbolic name is correct.
0448	DFSIAS00	A Type 2 batch step could not be initiated because the input transaction code named in the fourth positional operand of the PARM field had a nonzero limit, normal, or current priority. All priorities for a transaction code to be used as input by a Type 2 batch program must be zero.
0452	DFSIAS00	A Type 2 batch step could not be initiated because the transaction named in the fourth positional operand of the PARM field has been stopped or locked by a command or by a prior program failure.
0456	DFSIAS00	A Type 2 batch step could not be initiated because the program named in the second positional operand of the PARM field has been stopped or locked by a command or by a prior program failure.
0460	DFSIASE0	Type 2 batch region was canceled or terminated abnormally while a LOAD request was in process.

0464	DFSIASE0	A Type 1 message processing region was canceled or abnormally terminated by other than a /STOP REGION command.
0468	DFSIASE0	Type 1 or 2 processing region canceled or abnormally terminated while a Data Language/I call was in process in the Type 0 region.
0476	DFSIDLA0	A Type 1 or 2 processing application provided an invalid PCB address in a Data Language/I call.
0710	DFSIOS60	During OPEN of a Data Language/I overflow data set, the calculated block length exceeded the maximum track length for the device allocated. Check DD cards for OSAM data set allocation. Register 3 points to the Data Control Block (DCB) at ABEND time.
0713	DFSIAS00	Unable to schedule an application program because insufficient data base buffer space is available. Check TP and OSAM buffer pool size specified in the PARM field of the Type 0 EXEC statement

Commonly Encountered OS/360 System ABENDs

System ABEND Code	Usual Problem
806	JOBLIB card omitted from library containing IMS/360 modules or user's application program library.
213	DD cards for data sets representing data bases are missing or do not have proper DD name. DD names for data bases must correspond to those used in DMAN cards of DBD generation. or Data sets to be opened as DISP=OLD do not exist or cannot be opened.

CHAPTER 7. SYSTEMS OPERATION INTERFACE

The internal functions of IMS/360 are provided through the use of system control blocks. The information for some of these blocks must be provided by the application programmer to the Systems Operation function for application integration with the system. Other information may be either provided by the programmer or provided to him.

The application programmer must provide the following:

- Parameters describing data bases (Data Base Description -- DBD)
- Parameters describing programs (Program Specification Block - PSB)
- Parameters describing messages and terminals

ASSIGNMENT OF TRANSACTION CODES/LOGICAL TERMINALS

IMS/360 requires a one- to eight-character (followed by a blank) transaction code as the first element in all input messages from terminals. With the IMS/360 capability of message switching, logical terminal names may also be used as transaction codes.

The following rules apply to transaction codes and logical terminal names:

1. All transaction codes and logical terminal names must have unique values.
2. Transaction codes may be from one to eight bytes long.
3. To minimize the time required to enter a transaction at a terminal, transaction codes should be as short as possible.
4. Logical terminal names may be from two to eight bytes long.
5. The first character of transaction codes and logical terminal names must be any of the 29 characters (A through Z, $, #, and @) as defined by IBM System/360 Operating System: Assembler Language (GC28-6514).

DATA BASE DESCRIPTIONS (DBD)

The Data Base Description (DBD) is the Data Language/I control block used to describe in detail the structure and storage organization of every data base. All the information about a data base is available in its DBD and it is from this pool of information that other internal Data Language/I control blocks are built at execution time.

It is not the responsibility of the application programmer to generate the DBD's that affect the data bases he uses. However, it is imperative that he be able to understand the contents of a DBD in order to utilize the data bases that already exist and are available to him.

Before Data Language/I can be used to process data base information, or even before the data base can be created, the data base must be described to Data Language/I. The organization of the data within the data base must be completely described so that it can properly set up the tables and control blocks that determine how the data is to be stored.

The result of the procedures described here is the creation of a Data Language/I data base description (DBD) table that is required when the data for the data base is actually loaded into the system and when it is retrieved and manipulated during execution of any application program. Each DBD is stored as a member of a load module library generically titled the DBD library.

This procedure must precede any processing that will in any way reference the data base it describes. The data base cannot exist until the Data Base Description is completed.

In general, the control statements for DBD generation appear as follows:

1	[PRINT	NOGEN]
2	DBD	NAME=,ACCESS=
3	DMAN	DD1=,DEV1=,[DD2=],[DLIOF=]
4	SEGM	NAME=,PARENT=,BYTES=,FREQ=
5	FLDK	NAME=,TYPE=,BYTES=,START=
	[FLD	NAME=,TYPE=,BYTES=,START=]
6	DBDGEN	
7	FINISH	
8	END	

Note: At least one DMAN, SEGM and FLDK card must exist within the Data Base Description. Following each DMAN card there may be one or more SEGM and FLDK cards. For each SEGM card there must be one and only one FLDK card.

[ ] denotes optional statement or parameter

The following are the generalized rules for the DBD generation job step:

A - Number of DMAN cards determines whether the data base is composed of a single or multiple data set groups. One DMAN card per data set group.

B - ACCESS - (INDX) - Hierarchical indexed sequential organization

(SEQ) - Hierarchical sequential organization

C - Only one PRINT NOGEN card, - eliminates object listing from DBD generation

One DBD card

One DBDGEN card

One FINISH card

One END card

D - if INDX - only DD1, DLIOF, omit DD2=

if SEQ - both DD1, DD2, omit DLIOF=

E - follow the rules in the paragraph titled "DBD Control Card Requirements"

An example of a hierarchical description of control cards 3, 4, and 5 is:

```
1-DMAN (DATA SET GROUP 1)

2-SEGM (ROOT SEGMENT)

        3-FLDK
        3-FLD
        3-FLD

2-SEGM (LEVEL 2)

        3-FLDK
        3-FLD

2-SEGM (LEVEL 2)

        3-FLDK

1-DMAN (DATA SET GROUP 2)

2-SEGM (LEVEL 2)

        3-FLDK
        3-FLD

2-SEGM (LEVEL 2 or 3)
```

The job step itself consists of eight types of Data Language/I control cards arranged in a specific order. Each control card is described individually in detail in the following section.

DBD CONTROL CARD REQUIREMENTS

The description of the data base is presented to Data Language/I on eight types of control cards.

1. Each control card must be identified by a name, called a "card-type code", which comprises three to six characters: PRINT, DBD, DMAN, SEGM, FLD, DBDGEN, FINISH, or END.

2. In the generalized example shown in the following descriptions of the control cards, these conventions apply:

   a. Words written in all capital letters must appear exactly as written.

   b. Words written in lowercase letters are to be replaced by a user-specified value.

   c. The control cards are free form but must begin <u>after</u> column 1.

   d. The symbols [ ], { }, and ,... are used as an aid in defining the instructions. THESE SYMBOLS ARE NOT CODED; they act only to indicate how an instruction may be written.

      [ ] indicates optional operands. The operand enclosed in the brackets (for example, [VL]) may be coded or not, depending on whether the associated option is desired. If more than one item is enclosed in brackets (for example, $\begin{bmatrix}\text{REREAD}\\ \text{LEAVE}\end{bmatrix}$), one or more may be coded.

      { } indicates that a choice must be made. One of the operands from the vertical stack within braces (for example, $\begin{Bmatrix}\text{input}\\ \text{output}\end{Bmatrix}$) must be coded, depending on which of the associated services is desired.

   e. All DBDGEN control card parameters except for the print card are keyword parameters and therefore may appear in any sequence on the associated control card.

      ,...indicates that more than one set of operands may be designated in the same instruction.

PRINT Control Card

```
┌──────────────────────────────────────────────────────────────┐
│              │               │                               │
│              │   [PRINT      │  NOGEN]                       │
│              │               │                               │
└──────────────────────────────────────────────────────────────┘
```

The PRINT NOGEN card is an Operating System/360 macro generator control card used to eliminate a printout of the object listing resulting from a DBD generation. With the PRINT card present, a source statement summary is provided for each DBD defined.

DBD Control Card

This must be the first Data Language/I control card in the job step after the PRINT NOGEN. This card names the data base to be described and provides Data Language/I with preliminary information concerning its organization. There can be only one DBD control card in the control card deck. The parameters must be contained on one card.

```
|              |   DBD      |   NAME=name,
|              |            |
|              |            |              ( ISAM )
|              |            |   ACCESS=    { INDX }
|              |            |              { SAM  }
|              |            |              ( SEQ  )
```

where:

DBD=

identifies this control card as the DBD control card. This is the card-type code.

NAME=name

is the name of the DBD for the data base being described. This name may be from one to eight alphameric characters, must be left-justified, and must not have trailing blanks since it is not the last parameter. Normally, this name would be the same as that specified in the DD1 parameter of the DMAN control card, although this is not required.

ACCESS=

specifies the Data Language/I access method to be used in conjunction with this set and must be one of the following values:

ISAM or INDX -- Hierarchical indexed sequential organization

SAM or SEQ -- Hierarchical sequential organization

DMAN Control Card

A DMAN control card must immediately follow the DBD card. Each DMAN control card describes one data set group that is to be set up by Data Language/I as part of the data base being described. There are one primary data set group and zero to eight dependent data set groups in a single data base for the hierarchical indexed sequential organization. There is only one data set group for a hierarchical sequential data base.

Since the DMAN card parameters may appear on more than one card, provision has been made to accommodate the overflow of parameters to more cards. When this occurs:

1. Enter a nonblank character in Column 72 of each continued card.

2. A particular parameter or operand may not span two cards. If there is not space for the entire parameter on the current card, place the whole parameter on the next card. When continuation cards are required, a comma must follow each parameter except the last on the last continuation card.

3. Continue statement in Column 16 of next card.

4. The continued condition may occur in DMAN, SEGM, and FLD cards.

The number and arrangement of DMAN control cards allowed in a job step are greatly dependent upon the data base organization specified. For instance, for a SINGLE DATA SET GROUP - DATA BASE, only one DMAN control card and its associated overflow cards (if any) are allowed; for a MULTIPLE DATA SET GROUP - DATA BASE, up to ten DMAN cards are allowable. For hierarchical sequential data bases, only one DMAN card is allowed. The format of the DMAN control card where the DBD card has ACCESS=INDX or ISAM is:

```
|            | DMAN    | DD1=name,
|            |         |
|            |         | [DLIOF=name,]
|            |         |
|            |         | DEV1=device
```

The format of the DMAN control card where the DBD card has ACCESS=SEQ or SAM is:

```
|            | DMAN    | DD1=name,
|            |         |
|            |         | DEV1=device
|            |         |
|            |         | [,DD2=name]
```

where:

DMAN

identifies this Data Language/I control card as a DMAN control card. (See the setup examples at the end of this section.)

DD1=name

is a one- to eight-character alphameric name that is the ddname of the DD card for an ISAM data set, or an input data set under SEQ organization. This parameter must be specified regardless of the data base organization.

DEV1=device

designates the physical storage device type on which the prime area for this data set is to be stored. A list of the possible physical devices follows:

Device Name	DEV1=	
DRUM	2301	
DISK FILE	2302	
DISK PACK	2311	
DISK FACILITY	2314	
DATA CELL	2321	
TAPE	2400	(only when DBD ACCESS=SEQ)

(The underlined value may be used for DEV1=.)

DLIOF=name

a one- to eight-character alphameric name that is the ddname of the DD card. It is required only if INDX was specified in the DBD card ACCESS parameter. This 8-character name becomes the ddname on the DD card for the OSAM data set. Omit this parameter if the DBD card ACCESS parameter equals SEQ or SAM.

DD2=name

a one- to eight-character alphameric name that is the ddname of the DD card for the output data set under SEQ. This parameter must be omitted if the DBD card ACCESS card equals INDX or ISAM. The DD2 parameter should be specified only if the data base organization is hierarchical sequential.

The following table summarizes the parameters required on the DMAN control card for each of the Data Language/I access methods.

Access Method	Parameters Required
SEQ or SAM	DD1, DEV1, DD2
INDX or ISAM	DD1, DEV1, DLIOF

SEGM Control Card

At least one SEGM control card must immediately follow a DMAN set. The SEGM control card defines a segment to be contained in the data set group defined by a preceding DMAN control card. There may be a maximum of 255 segments defined. SEGM control cards must be entered in hierarchical order. The segments are physically stored in the data base record in the same order in which these cards are entered.

Provision has been made to accommodate the overflow of parameters on a SEGM control card to more cards. When this occurs, follow the rules stated above for overflow on the DMAN card.

The format of the SEGM control card is:

	SEGM	NAME=name,
		PARENT=parent,
		BYTES=bytes,
		FREQ=frequency

where:

SEGM

is the card-type code which identifies this as the SEGM control card.

NAME=name

is a one- to eight-character alphameric name of the segment. Within one DBD, duplicate segment names are not allowed.

PARENT=parent

a one- to eight-character alphameric name of the parent segment of this segment; left-justified and must not have trailing blanks. The first SEGM control card for this job step is assumed to be the root segment, and the "parent name" for the root segment must be a zero (see section titled "DBDGEN Examples").

BYTES=bytes

is the number of bytes of storage required to accommodate a single occurrence of this segment. If all the fields of this segment are defined by FLD control cards, and if none of the fields defined on the following FLD control card(s) overlap, this will be the sum of the lengths specified for these fields. A segment length may not exceed the length of one direct access device track.

FREQ=frequency

an estimate of the number of times this segment is likely to occur for each occurrence of its parent segment. If this is the root segment, it is the estimate of the maximum number of data base records that appear in the data base being defined. If this is the root segment, this parameter must be an integer in the range 1-99999999.

Note: Commas are not allowed in the frequency value.

The values given for dependent SEGM's are used in the computation of LRECL, blocking factor, and BLKSIZE. The frequency of occurrence of the root segment is used to determine the allocation of space required for cylinder index and prime ISAM storage at DBD generation execution. The output of DBD generation (the listing) specifies the number of tracks required for cylinder index and prime ISAM area allocation. If the root segment frequency is greater than 99999999, DBDGEN will not provide the definition of the prime space allocation required. The user must calculate this on the basis of the number of root segments that will actually reside in the data base. The number produced by DBDGEN will be erroneous. These three figures are shown on the generation output and help the Systems Operation function in determining the SPACE parameters of the DD cards for data bases. The parameter is an estimate, not a limit.

FLDK Control Card

The format of the FLDK control card is:

```
|              | FLDK     | NAME=name,
|              |          |
|              |          |       (X)
|              |          | TYPE= {P}  ,
|              |          |       (C)
|              |          |
|              |          | BYTES=bytes,
|              |          |
|              |          | START=position
```

where:

FLDK

is the card-type code that identifies this as the key field for this segment. The occurrences of this segment are kept in sort order on this field behind each occurrence of its parent segment. There must be one and only one key field defined for each segment. Each segment defined must have a key field defined by an FLDK card. A maximum of one FLDK card per SEGM card is allowed.

NAME=name

is a one- to eight-character alphameric name of this field. Within one segment, duplicate FLDK names are not allowed.

TYPE=X, P, or C

designates the type of data that is to be contained in this field. The value of this parameter specifies that one of the following types of data is to be contained in this field:

X - hexadecimal data

P - packed decimal data

C - alphameric data or a combination of types of data

For Data Language/I calls involving GET or INSERT functions, all comparisons upon key field or data field values are done on a byte-by-byte binary basis. However, during DBDGEN, the user may define different types of data within a field. No use is made by Data Language/I of this information.

BYTES=bytes

specifies the length of this field in terms of bytes.

If TYPE = X, BYTES should equal either 2 or 4.

If TYPE = P, BYTES should not exceed a maximum of 16.

If TYPE = C, BYTES should not exceed a maximum of 256.

The field lengths described above are warnings to the user who might execute the associated Operating System/360 instructions, such as full- or halfword fixed-length instructions on hexadecimal field. No checking is made within IMS/360 to ensure that the field length corresponds to these values.

START=position

specifies the starting position of this field in terms of bytes relative to the beginning of the segment. "Position" for the first byte of a segment is 1. Overlapping fields are permitted. It must be remembered, however, that the sum of bytes (including bytes for fields that are not defined, and not including any common bytes of overlapping fields) cannot exceed the length of this segment as specified on the SEGM control card.

FLD Control Card

One or more FLD control cards may follow the FLDK control card. This card defines each of the fields, in the segment defined by the preceding

SEGM control card, that may appear as part of a Data Language/I call qualification statement. All fields do not have to be defined. A maximum of 1000 FLDK and FLD cards may be defined in the entire DBD. A maximum of 254 FLD cards is allowed per SEGM card.

The format and parameters of the FLD control card are as outlined for the FLDK control card, above.

	[FLD]	

If a nonkey field is not referred to with a Data Language/I GET call, no field control card need be included in the DBDGEN.

FLD

> is the card-type code that identifies this as the control card for an ordinary data field. There can be many data fields for any given Data Language/I segment. If a field is to be used in a segment search argument, it must be defined with a field control card.

DBDGEN Control Card

	DBDGEN	

Since it is the key to generating the data base description from the parameters specified above, this control card must be included.

FINISH Control Card

	FINISH	

This control card must be included. It sets the on-zero condition code for link-edit if there are DBD generation errors.

END Control Card

	END	

Since it signifies the end of the DBDGEN, this control card must be entered.

DBDGEN Examples

1. Set up a hierarchical indexed sequential data base consisting of a single data set group (see Figure 41). Each data base record will contain two segments of two and three fields, respectively.

The data base will be stored on a 2311 disk pack. The access method to be used is indexed.

```
PRINT              NOGEN
DBD                NAME=DB,ACCESS=INDX
DMAN               DD1=DB,DEV1=2311,DLIOF=OVFL1
SEGM               NAME=S1,PARENT=0,BYTES=15,FREQ=100
FLDK               NAME=KEY,TYPE=X,BYTES=4,START=1
FLD                NAME=DATA,TYPE=C,BYTES=11,START=5
SEGM               NAME=S2,PARENT=S1,BYTES=20,FREQ=1
FLDK               NAME=KEY1,TYPE=X,BYTES=4,START=1
FLD                NAME=DATA1,TYPE=C,BYTES=12,START=5
FLD                NAME=DATA2,TYPE=P,BYTES=4,START=17
DBDGEN
FINISH
END
```

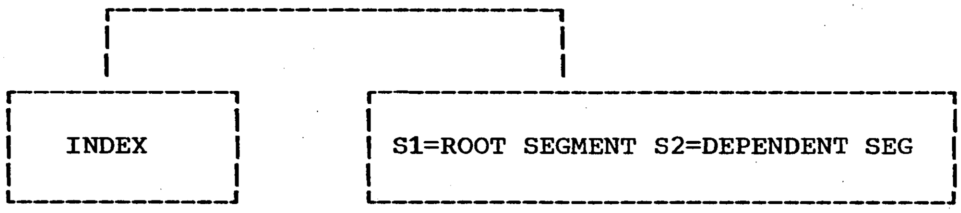


Figure 41. Single data set group

2. Set up a hierarchical indexed sequential data base consisting of multiple (2) data set groups (see Figure 42). Each data base record will contain two segments of two and three fields, respectively. The data base will be stored on a 2311 disk pack. An * indicates the differences between single and multiple data set group organizations but is not physically punched on the DMAN card. The access method to be used is indexed.

```
PRINT              NOGEN
DBD                NAME=DB,ACCESS=INDX
DMAN               DD1=DB,DEV1=2311,DLIOF=OVLF1
SEGM               NAME=S1,PARENT=0,BYTES=15,FREQ=100
FLDK               NAME=KEY,TYPE=X,BYTES=4,START=1
FLD                NAME=DATA,TYPE=C,BYTES=11,START=5
*DMAN              DD1=DS22,DEV1=2311,DLIOF=OVFL4
SEGM               NAME=S2,PARENT=S1,BYTES=20,FREQ=1
FLDK               NAME=KEY1,TYPE=X,BYTES=4,START=1
FLD                NAME=DATA1,TYPE=C,BYTES=12,START=5
FLD                NAME=DATA2,TYPE=P,BYTES=4,START=17
DBDGEN
FINISH
END
```

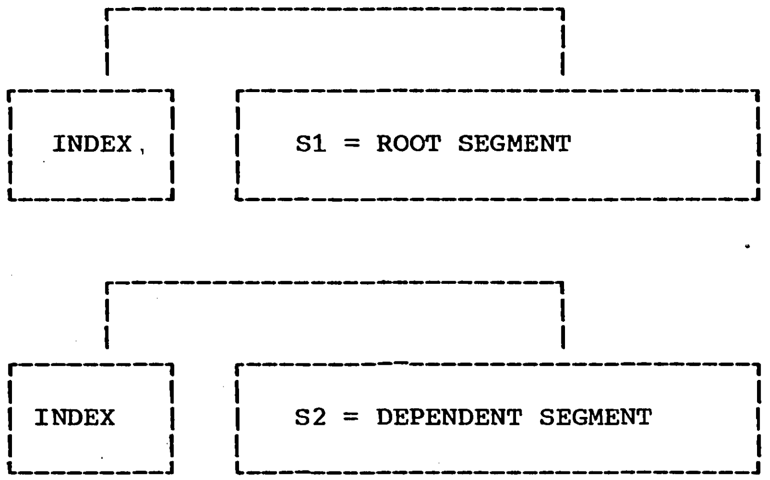


Figure 42. Multiple data set groups

3. Set up a hierarchical sequential data base. (A hierarchical sequential data base contains only a single data set group.) The data base record contains two segments with two and three fields, respectively (see Figure 43). The access method used is sequential, and the data base is stored on 2400 series magnetic tape.

```
PRINT          NOGEN
DBD            NAME=DB2,ACCESS=SEQ
DMAN           DD1=DB2,DEV1=TAPE,DD2=DB3
SEGM           NAME=S1,PARENT=0,BYTES=15,FREQ=100
FLDK           NAME=KEY,TYPE=X,START=1,BYTES=4
FLD            NAME=DATA,TYPE=C,START=5,BYTES=11
SEGM           NAME=S2,PARENT=S1,FREQ=1,BYTES=20
FLDK           NAME=KEY1,TYPE=X,START=1,BYTES=4
FLD            NAME=DATA1,TYPE=C,START=5,BYTES=12
FLD            NAME=DATA2,TYPE=P,START=17,BYTES=4
DBDGEN
FINISH
END
```

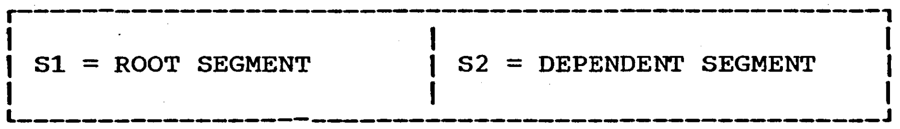


Figure 43. Hierarchical sequential data base

DESCRIPTION OF DBD GENERATION OUTPUT

Three types of printed output and a load module which becomes a member of the partitioned data set with the generic name of DBD library are produced by a DBD generation. Each of these outputs is described in the following paragraphs.

Control Card Listing

This is an exact reproduction of the character representation of the contents of each of the 80-column control cards. That is, it is a listing of the input card images to this job step.

Diagnostics

Errors discovered during the processing of each control card will result in diagnostic messages being printed immediately following the image of the last control card read before the error was discovered. The message may reference either the control card immediately preceding it or the preceding group of control cards. It is also possible for more than one message to be printed per control card. In this case, they follow each other on the output listing. After all the control cards have been read, a further check is made of the reasonableness of the entire deck. This may result in one or more additional diagnostic messages.

Any discovered error will result in the diagnostic message(s) being printed, the control cards being listed, and the other outputs being suppressed. However, all the control cards will be read and checked before the DBD generation execution is terminated. The link-edit step of DBD generation will not be processed if a control card error has been found.

Assembly Listing

An Operating System/360 Assembler Language listing of the DBD macro expansion created by DBD generation execution is provided. The inclusion of the PRINT NOGEN eliminates assembly information and provides a synopsis of the DBD control information.

Load Module

DBD generation is a two-step Operating System/360 job. Step 1 is a macro assembly execution which produces an object module. Step 2 is a link-edit of the object module which produces a load module, that becomes a member of the generic DBD library.

DBD Generation Error Conditions

The following are the DBD generation error conditions and the messages displayed for these conditions:

Error Message	Condition
DBD	---DBD010---Incorrect or missing access method
DBD	---DBD020---DBD name parameter not specified
DBD	---DBD030---Too many DBD cards
DBD	---DBD040---DBD name must begin with an alpha character
DMAN	---DMAN010---Incorrect device specification
DMAN	---DMAN020---Incorrect access specification
DMAN	---DMAN030---DD2 parameter invalid with ACCESS equal to ISAM
DMAN	---DMAN040---Too many DMAN cards
DMAN	---DMAN050---BLKFACT specified but no LRECL

DMAN	---DMAN060---LRECL specified but no BLKFACT operand
DMAN	---DMAN070---LRECLxBLKFACT greater than track length
DMAN	---DMAN080---Missing DLIOF operand with access equal to ISAM
DMAN	---DMAN090---DLIOF is present or DD2 is missing with access equal to SAM
DMAN	---DMAN100---DD1 operand omitted
DMAN	---DMAN110---DD1 and DD2 have same ddnames for HSAM
DMAN	---DMAN120---DD1/DLIOF duplicate ddnames for HISAM
SEGM	---SEGM10---Segment name not specified
SEGM	---SEGM20---Segment bytes parameter not specified
SEGM	---SEGM30---Segment frequency parameter not specified
SEGM	---SEGM40---Root segment parent must equal zero
SEGM	---SEGM50---Parent operand not specified for dependent segment
SEGM	---SEGM60---Too many SEGM cards; 255 maximum
SEGM	---SEGM70---Segment length greater than DASD track
SEGM	---SEGM80---Segment length specified as zero
SEGM	---SEGM90---Segment frequency of zero invalid
SEGM	---SEGM100---Duplicate segment names
SEGM	---SEGM110---Segment length greater than specified LRECL
FLD	---FLD010---Field name parameter not specified or invalid (that is, more than 8 characters)
FLD	---FLD040---Type parameter not specified or invalid
FLD	---FLD050---FLDK card not first after SEGM card
FLD	---FLD060---Too many FLD or FLDK cards specified

FLD	---FLD070---Field length extends beyond segment end
FLD	---FLD080---First byte of segment is 1.
FLD	---FLD100---Duplicate field name in segment
FLD	---FLD110---Bytes parameter invalid (that is, a nonnumeric field, 0 or less, or greater than 256).
FLD	---FLD120---Start parameter is invalid: 1 - The size of the field is greater than the size of the segment that it is in. 2 - Size of the start parameter is a nonnumeric field.
FLD	--FLD130---Specified fields in segment exceed 255
FLDK	---FLDK010---Key field specified inappropriately
DBDGEN	---DGEN010---Segment X Parent Y not found
DBDGEN	---DGEN020---Invalid number of DMAN cards for access method specified
DBDGEN	---DGEN030---DAM not supported
DBDGEN	---DGEN040---No segments for DMAN X
DBDGEN	---DGEN050---DAM not specified
DBDGEN	---DGEN060---Errors in this DBD
DBDGEN	---DGEN070---Too many levels in data base segment hierarchy
DBDGEN	---DGEN080---First segment in secondary data set group lower than level two
FINISH	---FINI10---No successful DBD's in this run

Because DBD generation is composed of Operating System/360 Assembler Language macro-instructions, omission or invalid sequence in DBD control cards, or invalid key word parameters will result in error statements from the Operating System/360 assembler.

PROGRAM SPECIFICATION BLOCK (PSB) GENERATION

PSB Requirements

Before an application program can be executed under IMS/360, it is necessary to describe that program and its use of terminals and data bases through a PSB generation. The PSB generation control cards supply the identification and characteristics of the PCB's (Program Communication Blocks) representing terminals and data bases to be used in the application program. There must also be a control card supplying characteristics of the application program itself. There must be one PSB for each message or batch processing program. The name of the PSB and its associated program must be the same. Coordinate with the Systems Operation function if this is not practical.

PSB generation places the PSB in the PSB library. The PSB and its PCB's (all PCB's to be used by the program are contained within the PSB) are stored in the library so that they can be used by IMS/360 related messages or data bases in a message or batch region. During IMS/360 system definition, a PSB directory is constructed. One entry exists in the PSB directory for each program which uses one or more data bases required to process messages. The PSB directory remains resident in the IMS/360 partition, while the PSB's are retained in main storage only when required for message processing and as the size of core storage allows.

Four basic types of control cards are used for a PSB generation:

- PCB control cards for output messages

- PCB control cards for Data Language/I data bases

- SENSEG control card for data base sensitive segments

- PSBGEN control card for each PSB

Note that the above list does not include a PCB for the input message. Upon entry to the application program, a PCB pointer to the source of the input message is provided as the first entry in a list of PCB pointers. The remainder of the PCB list has a direct relationship to the PCB's as defined within the associated PSB and must be in the same order. These PCB's are used by the application program when making Data Language/I message and data base calls.

In the case of a batch program, there is no input message PCB. Therefore, the PCB list provided to the program has a direct relationship to the PCB's within the PSB. No terminal PCB's should be contained in a PSB for batch processing in a Type 3 processing region.

The PCB list passed to the application program upon entry should be referenced within the processing program by the included names for making Data Language/I calls and interrogating PCB information (that is, status codes and feedback information).

Except that coding must not begin in Column 1, the format of the four PSB generation control cards is free form. The operation code (PCB, SENSEG, PSBGEN) must be followed by at least one blank. The keyword operands must contain no blanks and must be separated by commas.

PCB Control Card - Output Message PCB

The output message PCB type describes a PCB, which is associated with a logical distribution other than the source of input messages, to which the application program intends to send output messages. These messages may be sent either to an output terminal or to a transaction-type block to be handled by another program. There must be a separate output message PCB for every output message destination.

	PCB	TYPE=TP,
		LTERM=name

TYPE=TP

is a required keyword parameter for all output message PCB's.

LTERM=name

is the parameter keyword for the output message destination. The "name" is the actual destination of the message and is either a logical terminal name or a transaction-type name. When the name is a transaction-type name, output messages to this PCB are enqueued for input to the program used by that transaction type. The name is from one to eight alphameric characters in length. No special characters may appear in the logical terminal name.

Message PCB control cards must be first in the PSB generation deck, followed by the control cards identifying PCB's associated with Data Language/I data bases.

PCB Control Card - Data Language/I Data Base PCB

The second type of control card in a PSB generation deck is one that specifies a description of a PCB for a Data Language/I data base. The format for this type of control card is:

	PCB	TYPE=DB,
		DBDNAME=name,
		PROCOPT=X,
		KEYLEN=value

TYPE=DB

is a required keyword parameter for all Data Language/I data base PCB's.

DBDNAME=name

is the parameter keyword for the name of a data base description that was produced by a DBD generation run. This DBD name associates this PCB with a particular Data Language/I data base. The value for "name" must be eight characters or less in length.

PROCOPT=X

is the parameter keyword for the processing options that will be used by the processing program. The value for "X" must be one character. Possible values for the processing options are:

G - for GET function

A - for GET, DELETE, INSERT, and REPLACE

L - for loading a data base

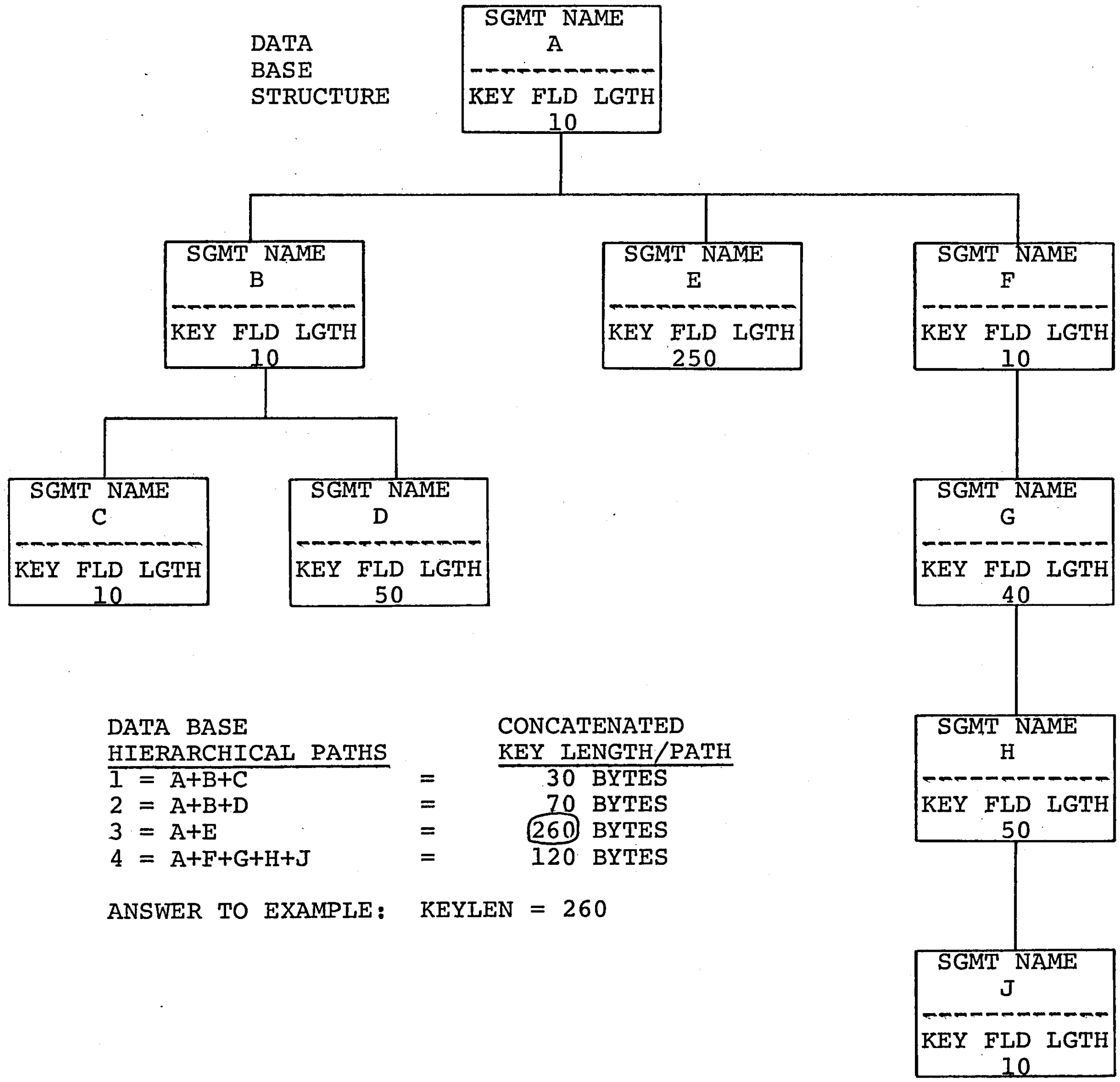
Only valid are:

```
HISAM - L
        G
        A

HSAM  - G
        L
```

KEYLEN = value

is the value specified in bytes of the longest concatenated key in bytes for a hierarchical path of sensitive segments used by the application program in the data base. The example shown in Figure 44 explains the definition of KEYLEN.



DATA BASE HIERARCHICAL PATHS		CONCATENATED KEY LENGTH/PATH
1 = A+B+C	=	30 BYTES
2 = A+B+D	=	70 BYTES
3 = A+E	=	260 BYTES
4 = A+F+G+H+J	=	120 BYTES

ANSWER TO EXAMPLE: KEYLEN = 260

Figure 44. Example of KEYLEN definition

SENSEG Control Card - Sensitive Segments

The SENSEG control card is used in conjunction with the PCB card for a Data Language/I data base and describes the segments in the data base to which the program is "sensitive". There must be one or more SENSEG cards for each data base PCB card, and they must immediately follow the PCB card to which they are related. There must be one card for each segment. The format of the SENSEG card is:

```
┌───────────────────────────────────────────────────────────────┐
│              │ SENSEG       │                                 │
│              │              │  sensitive-seg-name=XXXXXXXX,   │
│              │              │                                 │
│              │              │  *parent-seg-name=YYYYYYYY      │
│              │              │                                 │
└───────────────────────────────────────────────────────────────┘
```

*Omit on root segment.

sensitive-seg-name = XXXXXXXX

is the name of the segment as defined in the SEGM card at DBD generation time. The field contains from one to eight alphameric characters.

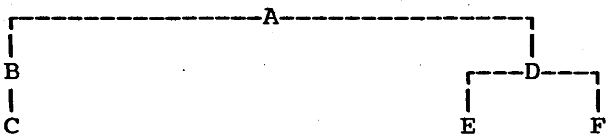
parent-seg-name = YYYYYYYY

is the name of the segment which is the parent to the sensitive segment above. The field contains from one to eight alphameric characters. The parent name must also agree with the parent name in the SEGM card at DBD generation time.

Within the definition of each sensitive segment type, the required format is to first specify the name of the segment being identified, follow that by a comma, and then give the segment name of this segment's parent. Since the root segment has no parent, its parent name is omitted.

The order in which sensitive segment cards are arranged must follow the hierarchical structure specified in the DBD generation. The definition should begin with the root segment and proceed down the leftmost path to the lowest level of the structure, then back up to a higher level and down again, continuing toward the right until the entire structure has been specified.

EXAMPLE OF SEGMENT DEFINITION: Assume that the structure of the data is:



and the program is sensitive to the whole structure. The complete PCB and SENSEG set for this Data Language/I data base structure may then be written as follows:

```
Col. 10     Col. 16                                  Col. 72

PCB         TYPE=DB,DBDNAME=DATABASE,                X
            PROCOPT=G,KEYLEN=22
SENSEG      A
SENSEG      B,A
SENSEG      C,B
SENSEG      D,A
SENSEG      E,D
SENSEG      F,D
```

PSBGEN Control Card

The third type of control card required for a PSB generation is one that specifies characteristics of the application program. The format for this card is:

```
|            | PSBGEN   |  LANG=XXXXX,          |
|            |          |                       |
|            |          |  PSBNAME=YYYYYYYY     |
```

LANG=XXXXX

is the parameter keyword for the Compiler Language in which this message processing program is written. The XXXXX value for this parameter must be either COBOL, PL/I, or ASSEM, with no trailing blanks.

PSBNAME=YYYYYYYY

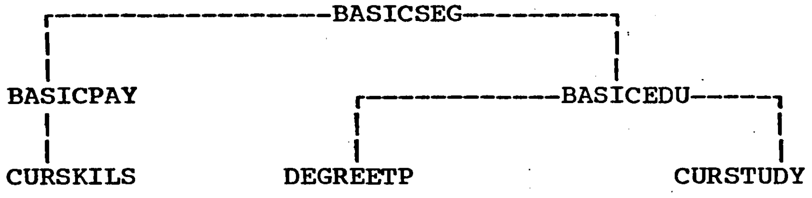
is the parameter keyword for the alphameric name of this PSB. The YYYYYYYY value for the PSBNAME must be eight characters or less in length. This name becomes the load module name for the PSB in the PSB library. This name must be the same as the program load module name in the program library. No special characters may be used in the name.

It should be noted that there may be several PCB-TYPE-TP control cards and several PCB-TYPE-DB control cards, but only one PSBGEN control card in a PSB generation card deck. The PSBGEN card must be the last control card in the deck preceding the END card.

The four types of PSB generation control card must be followed by an END card. The END card is required by the macro assembler to indicate the end of the assembly data.

Sample Deck for PSB Generation

A PSB generation is to be done for a message processing program to process the following hierarchical data base structure. Output messages are to be transmitted to logical terminals NUMBER15 and OFF35 in addition to the source terminal.



Sample:

```
PCB     TYPE=TP,LTERM=NUMBER15

PCB     TYPE=TP,LTERM=OFF35

PCB     TYPE=DB,DBDNAME=PAYRPERS,PROCOPT=A,KEYLEN=16

SENSEG BASICSEG

SENSEG BASICPAY,BASICSEG

SENSEG CURSKILS,BASICPAY

SENSEG BASICEDU,BASICSEG

SENSEG DEGREETP,BASICEDU

SENSEG CURSTUDY,BASICEDU

PSBGEN LANG=COBOL,PSBNAME=U842M004

END
```

DESCRIPTION OF PSB GENERATION OUTPUT

PSB generation produces three types of printed output and one load module that becomes a member of the partitioned data set with the generic name of PSB library. Each of these outputs is described in the following paragraphs.

Control Card Listing

This is an exact reproduction of the character representation of the contents of each of the 80-column control cards. That is, it is a listing of the input card images to this job step.

Diagnostics

Errors discovered during the processing of each control card will result in diagnostic messages being printed immediately following the image of the last control card read before the error was discovered. The message may reference either the control card immediately preceding it or the preceding group of control cards. It is also possible for more than one message to be printed for each control card. In this case, they follow each other on the output listing. After all the control cards have been read, a further check is made of the reasonableness of the entire deck. This may result in one or more additional diagnostic messages.

Any discovered error will result in the diagnostic message(s) being printed, the control cards being listed, and the other outputs being suppressed. However, all the control cards will be read and checked before the PSB generation execution is terminated. The link-edit step of PSB generation will not be processed if a control card error has been found.

Assembly Listing

An Operating System/360 Assembler Language listing of the PSB created by DBD generation execution is provided.

Load Module

PSB generation is a two-step Operating System/360 job. Step 1 is a macro assembly execution which produces an object module. Step 2 is a link-edit of the object module, which produces a load module that in turn becomes a member of the generic PSB library.

PSB Generation Error Conditions

Erroneous Control Card	Error Message
PCB	---PCB010---PCB type parameter missing or invalid
PCB	---PCB020---PCB LTERM parameter not specified for TP PCB
PCB	---PCB030---DBDNAME parameter not specified for DB PCB
PCB	---PCB040---KEYLEN parameter not specified for DB PCB
PCB	---PCB050---PROCOPT parameter not specified for DB PCB
PCB	---PCB060---DBDNAME specified for TP PCB
PCB	---PCB070---PROCOPT specified for TP PCB
PCB	---PCB080---KEYLEN operand for TP PCB
PCB	---PCB090---LTERM operand specified for DB PCB
PCB	---PCB100---Invalid processing option in PCB
PCB	---PCB110---TP PCB must occur before any DB PCB's
SENSEG	---SEG010---Segment name parameter is invalid
SENSEG	---SEG020---Too many SENSEG cards, 255 maximum
SENSEG	---SEG030---SENSEG is invalid for TP PCB's
SENSEG	---SEG040---Parent name parameter invalid
SENSEG	---SEG050---Parent segment is not predefined
SENSEG	---SEG060---Parent name parameter is omitted or invalid
SENSEG	---SEG070---Duplicate segment name

PSBGEN	---PSB010---PCB in error, generation terminated
PSBGEN	---PSB020---PSBNAME not specified
PSBGEN	---PSB030---Invalid Language Operand
PSBGEN	---PSB040---No sensitive segments for DB PCB
PSBGEN	---PSB050---PSB name must begin with alpha character
PSBGEN	---PSB099---System error, generation terminated

Because PSB generation is composed of Operating System/360 Assembler Language macro-instructions, errors in omission or invalid sequence of control cards, or invalid parameters on control cards will result in additional errors specified by Operating System/360 during PSB generation.

IMS/360 MESSAGE PROCESSING APPLICATION INTEGRATION CONSIDERATIONS

When a terminal user enters a message, it is held in the communications control facility until it is completed and checked. Once completely received, checked, and queued, the message is available for processing. The program that processes it is loaded and executed.

There is no limit to the number of transaction codes that a single message program may process (space excepted). However, if a message program processes multiple transaction codes, the message program must differentiate between them. Communications control provides validity checking and security control.

The SMB is a core resident block within the IMS/360 region. One SMB exists for each transaction type. The SMB is the internal definition of the transaction type.

The following parameters are required for the description of a transaction type and its associated SMB:

Limit Priority

is a value between 0 and 14. This keyword creates a scheduling priority higher than the normal priority to guarantee that no transaction type is left at a low priority if its message queue becomes long and it is not being serviced frequently enough under load conditions. Once the priority is boosted, it stays at the limit priority until its message queue is emptied.

Normal Priority

is a value between 0 and 14 that designates the priority level at which this transaction is scheduled and serviced during normal operating conditions.

Limit Count

is a count value less than 65,000 against which the number of waiting messages may be compared. When the number of messages waiting exceeds the limit count value, the transaction priority is boosted to the limit priority.

Program

is the name of the program which processes this transaction type. It is the same name as the PSB.

Transaction Code

is a one- to eight-character transaction code. This is the code used by the terminal operator when he enters his message.

Message Count

is a value less than 65,000 which indicates the maximum number of messages of the type that the associated message program is allowed to process during each program load. Twenty is the default value.

Message Time

is the maximum time in seconds allowed for the associated message program to process each message. Message time x message count is used internally as a time slice for message program processing loop control. Time is not accumulated during any Data Language/I message or data base operation. Therefore, it does not include any time consumed by IMS/360 services or Operating System/360 overhead.

Note: It is the responsibility of the application programmer to provide the above values to the Systems Operation function.

CHAPTER 8. TERMINAL OPERATIONS INTERFACE

TERMINAL COMMAND LANGUAGE

General Description

A remote terminal command language exists within the framework of IMS/360 to provide the user with a limited degree of control over the operation and status of his terminal. The objective of this chapter is to describe the usage of this remote terminal command language. This chapter should, however, be used in conjunction with the System/360 Operating System, Operator's Guide for the type of terminal being used. Although the remote terminal command language is similar to the master terminal command language (the details of this command language can be found in the IMS/360 Operations Manual, Volume II - Machine Operations), the function of each language is different. The function of the master terminal command language is the interrogation, alteration, and control of the overall IMS/360 system. The entry of these commands is closely regulated through the use of passwords.

The IMS/360 security maintenance program (SMP) provides both password and terminal protection of an online IMS/360 system. The generated IMS/360 system has only a minimum subset of terminal security to protect DISPLAY, NRESTART, CHECKPOINT, ERESTART, START, CHANGE, STOP, PURGE, DBRECOVERY, DBLOG, DBNOLOG, DBDUMP, ASSIGN, DELETE, and PSTOP commands. The security maintenance program creates password and terminal security for transactions and additional commands entered from terminals. It also creates password security on data bases and programs. The control of the security maintenance program is such that the user may view his system in terms of resources and which password may have access to those resources, or he may view the system as a security profile system, that is, define a password that has access to a set of resources. The detailed explanation will be found in Chapter 5 of the IMS/360 Operations Manual, Volume I - Systems Operation.

The function of the remote terminal command language is to change the status or mode of operation of the user's own terminal in order to provide extended security facilities, as illustrated by the /LOCK verb, and to provide extended user message entry facilities, as illustrated by the /CANCEL verb.

Note that remote terminal commands may be entered from any remote terminal or from the master terminal. Note also that the remote terminal command applies only to the terminal from which the command is entered (with the exception of /BROADCAST), whether or not the issuing terminal is the master terminal. The entry of any remote terminal command will result in the issuance of a message to the originating terminal. The message that is a response to a terminal command will override the generated status of the line, terminal, etc.

Remote terminal commands are limited to one line.

Structure of Remote Terminal Command Statements

The generalized format and description of the remote terminal command statement are as follows:

```
/VERB [(Password)] KEYWORD P1, . . . CR or EOB
```

After the command has been typed, the EOB key should be depressed (the carriage-return key (CR) can serve the same purpose if the terminal has the automatic EOB feature installed).

The /VERB (such as /LOCK) is the first element. For many of the remote terminal commands, such as /CANCEL or /TEST, the /VERB is the only element. The carriage-return key on the keyboard may be depressed (in order to position the print element at the left margin) before entering the /VERB. Unless the /VERB is the only element in the command, the user next enters one or more spaces before entering the first keyword.

The keyword may be separated from its first parameter (as designated by P1 above) by one or more spaces, a dash (-), or an equal sign (=).

The /LOCK command, in particular, provides password security at the parameter level, such as for a given data base or a given logical terminal, as defined at initial START time. If required, the password must be entered directly after the related parameter. The password is normally enclosed in parentheses. However, when entry is being made from a 1050 terminal, it may not be desired to print the password. Therefore, as an alternative, the password may be enclosed between bypass and restore characters: %PASSWORD*. No spaces or intervening characters may be entered between the parameter and the left parenthesis or bypass character. Unless otherwise noted, multiple parameters may be attached to a given keyword. Multiple parameters are separated by commas, or by commas followed by one or more blanks. If the parameter is not secured, the comma immediately follows the parameter. If the parameter is secured, the comma immediately follows the right parenthesis or the restore character that encloses the password. The last parameter that is attached to a given keyword must be followed by one or more blanks or by a period, not by a comma, as the comma designates a continuing series.

For purposes of documentation, comments or notes may be added at the end of a terminal command. However, to mark the end of the command, a period must be entered following the last required word of the command when comments are to be added.

```
/VERB [(Password)] KEYWORD P1, P2. COMMENT  CR or EOB

(COMMENT cannot overflow to next line.)
```

Correction of Remote Terminal Commands

The following are methods to be used if an error has been made when entering a remote terminal command:

Backspacing - If the EOB or CR (carriage-return) key has not been depressed, a typing error may be corrected by depressing the backspace key to the incorrect character, retyping it correctly, and retyping all subsequent characters.

Single line deletion - If it is necessary to delete a typed line, ** must be typed before the EOB key or CR key is depressed.

Remote Terminal Command Key Definitions

As several of the commands utilize the same keywords and parameters, reference should be made to the following directory when reviewing the commands:

LINE

is the keyword referring to a communication line; correct parameters are one- to three-character line numbers.

PTERM

is a keyword referring to a physical terminal; correct parameters are one- or two-character physical terminal addresses.

LTERM

is a keyword referring to a logical terminal; correct parameters are one- to eight-character logical terminal names.

TRAN

is a keyword referring to a transaction code; correct parameters are one- to eight-alphameric-character transaction codes.

PROGRAM

is a keyword referring to a program; correct parameters are one- to eight-alphameric-character program names.

DATABASE

is a keyword referring to a data base; correct parameters are one- to eight-character data base names.

ALL

ALL may be used as a parameter with many keywords. The specific acceptable uses of this parameter are noted in the descriptions of the individual commands.

P1, P2, etc.

are abbreviations used to designate possible parameters in the descriptions of the various verbs.

Remote Terminal Commands

1. /LOCK and /UNLOCK

These two commands are discussed together, as they are opposites. For example, /LOCK stops the sending and receiving of messages relative to a particular communications line or terminal, stops the scheduling of messages containing a specific transaction code, stops the scheduling of a specific program, and/or stops the scheduling or use of a given data base. This command allows the queuing of output messages relative to a particular communications line or terminal and/or allows the queuing of messages containing a specific transaction code.

If the terminals are on a switched network, these are the LOCK and UNLOCK command considerations: an implied /UNLOCK command is processed against a switched network PTERM and inquiry logical terminal whenever a physical or logical terminal disconnect occurs between a remote terminal and the IMS/360 system. Subpool logical terminals, however, are not affected by a disconnect. If further explanation is required, see Chapter 3 of the IMS/360 Operations Manual, Volume I - Systems Operation, the paragraph titled "IMS/360 Telecommunication Facilities".

COMMAND	TERMINAL/LINE			TRANSACTION		PROG	DATABASE
	REC	SEND	Q O/P	SCHED	Q	EXECUTE	USE
/LOCK	NO	NO	YES	NO	YES	NO	NO
/UNLOCK	YES	YES	YES	YES	YES	YES	YES

where:

REC allows receipt of input messages

SEND initiates sending of output messages

Q O/P allows output message queuing from processing

SCHED allows scheduling of messages for processing

Q allows input queuing of messages

EXECUTE allows use of a program for processing messages

USE allows use of data base for processing messages

Note: that /START and /UNLOCK, /STOP and /LOCK, or /PSTOP and /LOCK are not the same. Entry of the /START, /STOP, and /PSTOP commands would normally be restricted to the master terminal but could be allowed from a remote terminal. /LOCK and /UNLOCK, relative to a physical terminal, are applicable only to the physical terminal from which the command is entered. These two commands (/LOCK and /UNLOCK), relative to logical terminals, are applicable only to logical terminals that are assigned to the physical terminal from which the command is entered. The objective of the /LOCK command is to allow the terminal user to secure a specific physical terminal, one or more logical terminals associated with a specific physical terminal, one or more data bases, one or more programs, and/or one or more transaction codes.

Since the formats of /LOCK and /UNLOCK are identical, only one verb is shown in the following examples. Exception: /LOCK LTERM ALL is the only acceptable use of the parameter ALL relative to these commands. The following /LOCK and /UNLOCK formats are acceptable:

/LOCK PTERM (Password)

This command secures the user's physical terminal. Note that no keyword parameters are acceptable, as the user can lock only his own physical terminal.

/LOCK LTERM P1 (Password), P2 (Password)

This command secures one or more logical terminals associated with the user's physical terminal.

/LOCK LTERM ALL

This command secures all the logical terminals associated with the user's physical terminal.

/LOCK TRAN P1 (Password), P2 (Password)

This command secures one or more transaction codes.

/LOCK PROGRAM P1 (Password), P2 (Password)

This command secures one or more programs.

/LOCK DATABASE P1 (Password), P2 (Password)

This command secures one or more data bases.

2. /BROADCAST

Entry of this command would normally be restricted to the master terminal but could be allowed from any remote terminal. It is used to transmit a keyed warning or informational message to one or more terminals. This command results in the transmission of a specific message to one or more physical terminals. The message can be only one line in length. An end-of-block key must be depressed prior to the keying of the message to be broadcast. Refer to the master terminal section of the IMS/360 Operations Manual, Volume II - Machine Operations for a detailed description of this command.

The next two /BROADCAST commands are identical. They result in the transmission of the broadcast message to the physical terminals to which logical terminals P1, P2, and P3 are assigned. The following command formats are acceptable:

```
/BROADCAST (Password) TO LTERM P1, P2, P3 (EOB)

MSG------------------------ (EOB)

/BROADCAST (Password) TO P1, P2, P3 (EOB)

MSG------------------------ (EOB)
```

The next four /BROADCAST commands are functionally identical. They result in the transmission of the broadcast message to all the physical terminals in the system.

```
/BROADCAST (Password) TO LTERM ALL (EOB)

MSG------------------------ (EOB)

/BROADCAST (Password) TO ALL (EOB)

MSG------------------------ (EOB)

/BROADCAST (Password) TO PTERM ALL (EOB)

MSG------------------------ (EOB)

/BROADCAST (Password) TO LINE ALL (EOB)

MSG------------------------ (EOB)
```

The next two commands are functionally identical. They result in the transmission of the broadcast message to all the physical terminals located in line P1.

```
/BROADCAST TO LINE P1 (EOB)

MSG------------------------- (EOB)

/BROADCAST TO LINE P1 PTERM ALL (EOB)

MSG------------------------- (EOB)
```

The following command results in the transmission of the broadcast message to relative physical terminals P2 and P3 located on line P1.

```
/BROADCAST (Password) to LINE P1 PTERM P2, P3 (EOB)

MSG------------------------- (EOB)
```

3. /TEST

This terminal command is used to place the user's own terminal into test mode, which implies that no independent output messages will be transmitted to the user's terminal. Any input messages entered into the user's terminal will be transmitted back to the user's terminal. After the /TEST verb is entered, the user's terminal will remain in the test mode until such time as an /END command has been received from the user's terminal. Any messages which result from command processing or message switching or as the result of message processing output are enqueued for the terminal in test mode and are held until the terminal is removed from test mode. The /TEST command can apply only to the user's terminal. There are no acceptable keywords or parameters. The only acceptable format is the verb itself, as follows:

```
/TEST [(Password)]
```

4. /EXCLUSIVE

This command is used to place the user's own terminal into exclusive use or inquiry mode. The user enters this mode, through the entry of the /EXCLUSIVE verb, if he desires to enter one or more inquiries into his terminal and wants to receive only the response to his inquiries, without receiving output from other miscellaneous sources. Scheduling and queuing are allowed to continue. After the command has been entered, the user's terminal will remain in the inquiry mode until such time as an /END command has been received from the user's terminal. The /EXCLUSIVE command can apply only to the user's terminal. There are no acceptable keywords or parameters.

Since messages are displayed as soon as possible after queuing, the /EXCLUSIVE command is recommended for proper operation of the 2260 terminal. It will protect the screen of information from being overlaid by message switching, system messages, and messages generated by processing programs initiated by other terminals while the operator is viewing a response he initiated. These messages will remain on the queue until a /END command is entered. The only acceptable format is the verb itself, as follows:

```
/EXCLUSIVE [(Password)]
```

5. /END

This command is used to terminate the mode that was originally initiated through the entry of /TEST or /EXCLUSIVE. This command can apply only to the user's terminal. There are no acceptable keywords or parameters. The only acceptable format is the verb itself, as follows:

```
/END [(Password)]
```

6. /LOG

This terminal command is limited to one line in length, as is any command (slash-type) message. The function of the command is to cause the contents of the entered message to be logged, not processed by a program, with the slash (/LOG) being the first character logged. This command applies only to the currently entered message line and does not establish a continuing operational mode. There are no acceptable keywords or parameters as such. One or more spaces must separate the verb from the first letter of the message to be logged. The first word of the message, following the /LOG verb, may be a transaction code. To log the message "Today is Monday", the following format would be acceptable:

```
/LOG [(Password)] TODAY IS MONDAY
```

7. /CANCEL

The function of this command is to cause the cancellation of all lines of a multiple-line message that is currently being entered into this same terminal. Note that this command causes the cancellation of a complete message. It cannot be used to cancel a single-line input message. An erroneous single line can be canceled through the entry of two asterisks (**) immediately followed by an end-of-block (EOB) character at the end of the line to be canceled. There are no acceptable keywords or parameters. The only acceptable format is the verb itself, as follows:

```
/CANCEL [(Password)]
```

8. /SET

This terminal command sets the destination of all messages entered into this terminal to another terminal (/SET mode to LTERM master) or to an SMB (/SET MODE to TRAN IMS) (password). It may be changed by /RESET, /START LINE for the line on which the subject terminal is attached, or by the /IAM command. If the transaction is secured by password, checking is done at the time of processing the command. The allowable format is:

```
| /SET          |   [MODE] {TRAN  }  name    [(Password)]  |
|               |          {LTERM }  name                  |
| [(Password)]  |                                          |
```

Message editing considerations: Destination (TRAN name or LTERM name) of the SET command is edited as the leading field of the message. A 'blank' separator is inserted between the destination name and the text entered from the terminal, unless the first character entered from the terminal is a 'blank'.

9. /RESET

This terminal command eliminates the preset destination mode. The allowable format is:

```
| /RESET  |   [(Password)]                          |
```

10. /IAM

The /IAM command is applicable only to a switched communications network on dialup facilities. This command must be entered before any input transaction codes or other remote terminal commands will be accepted. The following formats are acceptable:

```
/IAM            LTERM P1 [(Password)]
                - - - - - - - - - - - - - - - - - - - - - -
[(Password)]    LTERM P2 [(Password)]
                - - - - - - - - - - - - - - - - - - - - - -
                PTERM [(Password)] LTERM P1 [(Password2)]
```

where:

LTERM P1

means this command automatically accomplishes the attachment of the subpool logical terminal P1 to the switched (dialup) communication line over which the call was received from the remote (physical) terminal.

LTERM P2

means this command automatically accomplishes the logical attachment of the Inquiry logical terminal to the switched (dialup) communication line over which the call was received from the remote (physical) terminal. Only the first four characters of the Inquiry logical terminal name are compared with the first four characters of the P2 parameter.

For a further detail discussion of logical terminals, see Chapter 3 of the IMS/360 Operations Manual, Volume I - Systems Operation.

PTERM (Password) LTERM P1 (Password)

means the same as the above operand, but accomplishes the attachment of all logical terminals associated with the subpool in which P1 exists.

11. /RDISPLAY

This command provides the ability from a remote terminal to display the logical terminal name, the relative physical terminal number, and the line number assigned as the master terminal. The entry on the remote terminal is:

```
/RDISPLAY [(Password)] MASTER
```

Recommended Contents

The application programming function should choose these terminal operator commands and make them known to their remote terminal operators. All necessary additional instructions, such as the following, should be placed in the procedures manual:

- What to do in case of trouble.
- What diagnostics can be run at the remote terminal to help determine the trouble. Perhaps a trouble checklist.
- Telephone numbers of different people to aid the operator.

GENERAL MESSAGE PROGRAM

The result of a user entering a valid input message is that IMS/360 schedules the associated message processing program. This sample program processes the input messages and returns a reply to the user at his terminal. The example is written in COBOL. It should be restated that the purpose of presenting this program is not to demonstrate all application programming aspects of IMS/360, but to give an illustration of a realistic but simple message processing program. The program's statements are numbered for discussion purposes. The COMMENT statements in the Data Division are artificial and would not compile in an actual program.

```
 1 IDENTIFICATION DIVISION.
 2 PROGRAM-ID.  'POLRPROG'.
 3  REMARKS.  THIS IS AN EXAMPLE OF AN IMS/360 MESSAGE
 4  PROCESSING PROGRAM.  THIS PROGRAM
 5  WILL SERVICE AN INQUIRY FOR
 6  THE LATEST STATUS OF A SPECIFIC
 7  PRODUCTION ORDER.

 8  ENVIRONMENT DIVISION.
 9  CONFIGURATION SECTION.
10 SOURCE-COMPUTER.  IBM-360.
11 OBJECT-COMPUTER.  IBM-360.

12 DATA DIVISION.
13     COMMENT -- NORMAL COBOL SPECIFICATIONS
14          FOR FILES ARE ABSENT SINCE DATA
15          LANGUAGE/I IS USED FOR
16        FILE HANDLING IN A MESSAGE PROGRAM.

17 WORKING-STORAGE SECTION.
18   77    GET-UNIQUE, PICTURE IS X(4), VALUE IS 'GU  '.
19   77  DLI-INSERT, PICTURE IS X(4), VALUE IS 'ISRT'.

20    COMMENT--STATEMENTS 18 and 19
21        NAME AREAS WHICH CONTAIN LITERAL
22        VALUES FOR THE DATA LANGUAGE/I FUNCTIONS
23        OF GET UNIQUE AND INSERT.  THE
24        NAMES OF THESE AREAS ARE USED IN
25        DATA LANGUAGE/I CALLS.

26 01 POLR-SSA.
27      02 SEG-NAME, PICTURE X(8), VALUE 'ORGERSEG'.
28      02 BEGIN-OP, PICTURE X, VALUE '('.
29      02 KEY-NAME, PICTURE X(8), VALUE 'ORDERNUM'.
30      02 RELATION-OP, PICTURE XX, VALUE ' ='.
31      02 KEY-VALUE, PICTURE X(6).
32      02 END-OP, PICTURE X, VALUE ')'.

33      COMMENT--THE ABOVE STATEMENTS (26-32)
34      FORM THE SEGMENT SEARCH ARGUMENTS
35      THAT ARE USED IN RETRIEVING
36      THE STORED PRODUCTION ORDER
37      SEGMENT.  NOTICE THE RELATIONAL OPERATOR IN
        STATEMENT 30 IS TWO CHARACTERS, THE FIRST OF
        WHICH IS BLANK.
```

```
38  01 TERMINAL-IN-AREA.
39       02 CHARAC-COUNT, PICTURE IS S99, COMPUTATIONAL.
40       02 FILLER, PICTURE IS S99, COMPUTATIONAL.
41       02 TRANS-CODE, PICTURE IS XX.
42       02 FILLER, PICTURE IS X.
43       02 TER-IN-OCN, PICTURE IS X(6).

44       COMMENT -- THE ABOVE (38-43) IS THE INPUT
45          AREA FOR THE MESSAGE FROM THE TERMINAL
46          AS IT WAS KEYED IN BY THE
47          USER.  THE MESSAGE PROCESSING
48          PROGRAM OBTAINS THE MESSAGE
49          THROUGH A DATA LANGUAGE/I GET CALL.

50  01   TERMINAL-OUT-AREA.
51       02 CHAR-COUNT, PICTURE S99, COMPUTATIONAL, VALUE
         + 133.
52       02 FILLER, PICTURE IS S99, COMPUTATIONAL, VALUE +00.
53       02 ORDER-CON-NUM, PICTURE X(6).
54       02 FILLER, PICTURE X(3), VALUE SPACES.
55       02 DEPT, PICTURE X(6).
56       02 FILLER, PICTURE X(3), VALUE SPACES.
57       02 STATUS-CODE, PICTURE XX.
58       02 FILLER, PICTURE X(3), VALUE SPACES.
59       02 SHIP-TO-DEPT, PICTURE X(6).
60       02 FILLER, PICTURE X(3), VALUE SPACES.
61       02 INITIAL-STORES, PICTURE X(6).
62       02 FILLER, PICTURE X(3), VALUE SPACES.
63       02 FINAL-STORES, PICTURE X(6).
64       02 FILLER, PICTURE X(3), VALUE SPACES.
65       02 PART-NUMBER, PICTURE X(20).
66       02 FILLER, PICTURE X(3), VALUE SPACES.
67       02 QUANTITY, PICTURE X(8).
68       02 FILLER, PICTURE X(3), VALUE SPACES.
69       02 TYPE-ORDER-CODE, PICTURE X(6).
70       02 FILLER, PICTURE X(3), VALUE SPACES.
71       02 ACCOUNT-NUM, PICTURE X(5).
72       02 FILLER, PICTURE X(3), VALUE SPACES.
73       02 IN-WORK-DATA, PICTURE X(4).
74       02 FILLER, PICTURE X(3), VALUE SPACES.
75       02 COMPL-DATE, PICTURE X(4).
76       02 FILLER, PICTURE X(3), VALUE SPACES.
77       02 IND-OF-UNDER, PICTURE X(4).
78       02 FILLER, PICTURE X(3), VALUE SPACES.
79       02 DATA-AS-OF, PICTURE X(4).
80       02 FILLER, PICTURE X(3), VALUE SPACES.

81       02 COMMENT -- THE ABOVE (50-80) IS THE
82       02 FORMAT FOR THE OUTPUT MESSAGE
83       02 THAT WILL BE SENT TO THE
84       02 TERMINAL USER.  THE OUTPUT LINE
85       02 IS TYPED ON PRE-PRINTED
86       02 FORM PAPER WITH FIELD HEADINGS.

87  01   POLR-SRA.
88       02 ORDER-CON-NUM, PICTURE X(6).
89       02 DEPT, PICTURE X(6).
90       02 STATUS-CODE, PICTURE XX.
91       02 SHIP-TO-DEPT, PICTURE X(6).
92       02 INITIAL-STORES, PICTURE X(6).
93       02 FINAL-STORES, PICTURE X(6).
94       02 PART-NUMBER, PICTURE X(20).
95       02 QUANTITY, PICTURE X(8).
96       02 TYPE-ORDER, PICTURE X(6).
97       02 ACCOUNT-NUM, PICTURE X(5).
```

```
 98       02 IN-WORK-DATE, PICTURE X(4).
 99       02 COMPL-DATE, PICTURE X(4).
100          02 IND-OF-UNDER, PICTURE X(4).
101          02 DATA-AS-OF, PICTURE X(4).

102          COMMENT - THE ABOVE (87-101) IS THE SEGMENT
103             RETURN AREA INTO WHICH
104             THE RETRIEVED PRODUCTION
105             ORDER SEGMENT IS PLACED FOLLOWING
106             THE EXECUTION OF A DATA LANGUAGE/I
107             GET CALL.

108     LINKAGE SECTION.
109         COMMENT -- THE LINKAGE SECTION DESCRIBES
110            DATA THAT IS MADE AVAILABLE TO THE
111            MESSAGE PROCESSING PROGRAM FROM IMS --
114            STORAGE SPACE IS NOT RESERVED
115            WITHIN THE MESSAGE PROCESSING PROGRAM
116            SINCE THE DATA EXISTS IN THE
117            IMS REGION --
119            THE PROGRAM COMMUNICATION BLOCK
120            FOR THE ON-LINE TERMINAL (127-131)
121            CONTAINS THE NAME OF THE TERMINAL,
122            THE RETURN STATUS CODE FOLLOWING
123            A DATA LANGUAGE/I CALL INVOLVING THE
124            TERMINAL, AND AN INPUT-PREFIX
125            WITH THE DATE AND TIME THE
126            MESSAGE WAS ENTERED.

127    01   TERMINAL-PCB.
128         02 IO-TERMINAL, PICTURE IS X(8).
129         02 IO-RESERVE, PICTURE IS XX.
130         02 IO-STATUS, PICTURE IS XX.
131         02 INPUT-PREFIX, PICTURE IS X(12).

132         COMMENT -- THE PROGRAM COMMUNICATION
133            BLOCK FOR THE DATA BASE
134            PROVIDES SPECIFICALLY DESIGNATED
135            AREAS FROM WHICH DATA LANGUAGE/I OBTAINS
136            INFORMATION IT NEEDS TO PROCESS THE
137            PROGRAM'S DATA REQUESTS.  THE PCB
138            ALSO PROVIDES SPECIFIC AREAS USED
139            BY DATA LANGUAGE/I TO ADVISE THE APPLICATION
140            PROGRAMMER OF THE RESULTS OF HIS
141            REQUESTS.

142    01   POLRDATA-PCB.
143         02 PK-DBD-NAME, PICTURE IS X(8).
144         02 PD-SEG-LEVEL-IND, PICTURE IS XX.
145         02 PD-STATUS-CODE, PICTURE IS XX.
146         02 PD-PROC-OPTIONS, PICTURE IS XXXX.
147         02 DLI-USE, PICTURE IS S9(5), USAGE
               COMPUTATIONAL.
148         02 PD-SEGMENT-FDBACK, PICTURE IS X(8).
149         02 PD-LENGTH-FDBACK, PICTURE IS S9(5), USAGE
               COMPUTATIONAL.
150         02 PD-NUM-SENS-SEG, PICTURE IS S9(5), USAGE
               COMPUTATIONAL.
151         02 KEY-FEEDBACK-AREA, PICTURE IS X(6).

153    PROCEDURE DIVISION

154         NOTE -- THE APPROACH OF THE PROCEDURE
155            DIVISION IS TO READ THE
156            INPUT MESSAGE, EXTRACT THE
```

```
       ORDER CONTROL NUMBER FROM THE
       INPUT MESSAGE, RETRIEVE A
     PRODUCTION ORDER SEGMENT FROM
       THE DIRECT ACCESS STORAGE
       DEVICE USING THE ORDER CONTROL
       NUMBER, CONSTRUCT THE OUTPUT
       MESSAGE, AND SEND THE MESSAGE
       TO THE TERMINAL.

ENTRY-POINT.
    ENTER LINKAGE.
    ENTRY 'DLITCBL' USING TERMINAL-PCB, POLRDATA-PCB.
  ENTER COBOL.
    NOTE -- THE ENTRY STATEMENT PASSES
     THE ADDRESSES FOR THE
 TERMINAL-PCB AND THE POLRDATA-PCB
     TO THE MESSAGE PROCESSING PROGRAM --
       THESE VALUES ARE PHYSICALLY
       LOCATED IN THE IMS/360 CORE REGION.

  ENTER LINKAGE.
  CALL 'CBLTDLI' USING GET-UNIQUE,
                       TERMINAL-PCB,
                       TERMINAL-IN-AREA.
  ENTER COBOL.
IF IO-STATUS NOT = ' ', GO TO ERROR-ANALYSIS.

  NOTE -- A DATA LANGUAGE/I CALL TO READ THE INPUT
   MESSAGE HAS BEEN ISSUED,
     AND FOLLOWING THAT, THE RETURN STATUS
     CODE CHECKED TO DETERMINE IF
     THE READ WAS SUCCESSFUL OR NOT --
     IF SUCCESSFUL, THE NEXT STEP IN THE PROGRAM
     IS TO MOVE THE ORDER CONTROL
     NUMBER FROM THE INPUT MESSAGE
   TO A SEGMENT SEARCH ARGUMENT.

  MOVE TER-IN-OCN TO KEY-VALUE.
  ENTER LINKAGE.
  CALL 'CBLTDLI' USING GET-UNIQUE,
                       POLRDATA-PCB,
                   POLR-SRA,
                       POLR-SSA.
  ENTER COBOL.
  IF PD-STATUS-CODE NOT =' ', GO TO ERROR-ANALYSIS.

NOTE -- A CALL TO READ THE PRODUCTION
  ORDER SEGMENT FROM THE DIRECT ACCESS
  STORAGE DEVICE HAS BEEN ISSUED --
  FOLLOWING THAT, THE STATUS CODE
  WAS CHECKED TO DETERMINE IF THE
    READ WAS SUCCESSFUL OR NOT.

MOVE CORRESPONDING POLR-SRA TO
         TERMINAL-OUT-AREA.
ENTER LINKAGE.
  CALL 'CBLTDLI' USING DLI-INSERT,
                       TERMINAL-PCB,
                  TERMINAL-OUT-AREA.
  ENTER COBOL.
  IF IO-STATUS NOT = ' ', GO TO ERROR-ANALYSIS.
  NOTE -- THE OUTPUT MESSAGE WAS
     CONSTRUCTED WITH A MOVE
     CORRESPONDING AND THEN SENT
     TO THE IMS/360 OUTPUT QUEUE WITH
     A DATA LANGUAGE/I INSERT CALL -- IMS/360
     WILL REMOVE THE MESSAGE FROM
     THE QUEUE AND SEND IT TO THE
       ON-LINE TERMINAL.  IT IS RECOMMENDED THAT
        EACH OUTPUT MESSAGE LINE CONTAIN CONTROL
        CHARACTERS FOR LINE FEED AND CARRIAGE RETURN.
    ENTER LINKAGE.
    RETURN.
    ENTER COBOL.


    NOTE -- THIS STATEMENT RETURNS
       CONTROL TO THE INFORMATION
       MANAGEMENT SYSTEM WHEN THE
       MESSAGE PROCESSING PROGRAM
     HAS COMPLETED EXECUTION.


ERROR-ANALYSIS.
    NOTE -- THE PURPOSE OF THIS BLOCK
        OF CODE, WHEN COMPLETED, IS
        TO ANALYZE A NON-BLANK
        STATUS CODE THAT WAS RETURNED
        AFTER THE EXECUTION OF A DL/I
        CALL -- DEPENDING ON THE PROBLEM
        INDICATED, IT MAY BE BEST TO
        RETURN AN ERROR MESSAGE TO THE
        TERMINAL USER.
```

COMPLETE SET OF PROGRAM EXAMPLES IN COBOL

This complete set of programs contains the following:

- Data Base Creation (Load) Program
- A Program to Create Data for Load Program
- Batch (Update) Processing Program
- Data Base Reorganization (Dump) Program
- Message (Update) Processing Program
- Data Base Descriptions
- Program Specification Block Generation

All of these programs employing COBOL show the IMS/360 capabilities as simply as possible. The programs exercise the Data Language/I facilities -- the CALL statements:

GU
GN
GNP
ISRT
REPL
DLET

In the message processing program, password and terminal security, along with several priorities, is exercised.

Data Base Creation (Load) Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID, 'HIBLSN01'
AUTHOR.
REMARKS.  THIS PROGRAM IS A TEST LOAD.
         TWO SINGLE DATA SET GROUPS AND TWO MULTIPLE DATA SET
         GROUPS CAN BE LOADED.  A SINGLE CONTROL CARD IS
         REQUIRED.  FORMAT:  COL. 1-2 INDICATES LOADING OF
         SINGLE DSG DATA BASES IF NON-BLANK;  COL. 3-4
         INDICATES LOADING OF MULTIPLE DSG DATA-BASES IF NON-
         BLANK.  ONE INPUT-DATA-TAPE IS REQUIRED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360.
OBJECT-COMPUTER. IBM-360.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CTLCRD ASSIGN TO 'SYSIN' UTILITY.
    SELECT INTAPE ASSIGN TO 'TAPEIN' UTILITY.
    SELECT DISKI ASSIGN TO 'DISKI' UTILITY.
DATA DIVISION.
FILE SECTION.
FD CTLCRD
    LABEL RECORDS STANDARD
    BLOCK CONTAINS 80 CHARACTERS
    RECORDING MODE F
    DATA RECORD  CARD-IN.
01  CARD-IN.
    02  CASE1.
        03  C11 PICTURE X.
        03  C12 PICTURE X.
    02  CASE2.
        03  C21 PICTURE X.
        03  C22 PICTURE X.
    02  DUMP-CASE1.
        03  DC11    PICTURE X.
        03  DC12    PICTURE X.
    02  DUMP-CASE2.
        03  DC21    PICTURE X.
        03  DC22    PICTURE X.
    02  FILLER  PICTURE  X(72).
FD  INTAPE
    LABEL RECORDS OMITTED
    BLOCK CONTAINS 8 RECORDS
    RECORD CONTAINS 440 CHARACTERS
    RECORDING MODE F
    DATA RECORD TAPE-IN.
01  TAPE-IN.
    02  PARENT.
        03  KEY1    PICTURE 9(6).
        03  FILLER1 PICTURE X(84).
        03  LEVEL2.
            04  KEY2.   PICTURE 9(6).
            04  FILLER2 PICTURE X(85).
            04  LEVEL3.
                05  KEY3    PICTURE 9(6).
                05  FILLER3 PICTURE X(253).
FD  DISKI
    LABEL RECORDS STANDARD
    BLOCK CONTAINS 80 CHARACTERS
       RECORDING MODE F
    DATA RECORD IS INTREC.
01  INTREC.
```

```
    02  DC011   PICTURE XX.
    02  DC022   PICTURE XX.
    02  FILLER  PICTURE X(76).
WORKING-STORAGE SECTION.
01  CALL-FUNC   PICTURE X(4)    VALUE  'LOAD'
01  PSBNAME     PICTURE X(8)    VALUE  'HIBLSN01'.
01  SSA1.
    02  SSA1-NAME   PICTURE X(8)    VALUE  'PARENT  '.
    02  SSA1-BEGIN  PICTURE X       VALUE  '('.
    02  SSA1-KEY    PICTURE X(10)   VALUE  'KEY1     ='.
    02  SSA1-VALUE  PICTURE 9(6).
    02  SSA1-END    PICTURE X       VALUE  ')'.
01  SSA21.
    02  SSA21-NAME  PICTURE X(8)    VALUE  'LEVEL21 '.
    02  SSA21-BEGIN PICTURE X       VALUE  '('.
    02  SSA21-KEY   PICTURE X(10)   VALUE  'KEY21    ='.
    02  SSA21-VALUE PICTURE 9(6).
    02  SSA21-END   PICTURE X       VALUE  ')'.
01  SSA31.
    02  SSA31-NAME  PICTURE X(8)    VALUE  'LEVEL31 '.
    02  SSA31-BEGIN PICTURE X       VALUE  '('.
    02  SSA31-KEY   PICTURE X(10)   VALUE  'KEY31    ='.
    02  SSA31-VALUE PICTURE 9(6).
    02  SSA31-END   PICTURE X       VALUE  ')'.
01  SSA22.
    02  SSA22-NAME  PICTURE X(8)    VALUE  'LEVEL22 '.
    02  SSA22-BEGIN PICTURE X       VALUE  '('.
    02  SSA22-KEY   PICTURE X(10)   VALUE  'KEY22    ='.
    02  SSA22-VALUE PICTURE 9(6).
    02  SSA22-END   PICTURE X       VALUE  ')'.
01  SSA32.
    02  SSA32-NAME  PICTURE X(8)    VALUE  'LEVEL32 '.
    02  SSA32-BEGIN PICTURE X       VALUE  '('.
    02  SSA32-KEY   PICTURE X(10)   VALUE  'KEY32    ='.
    02  SSA32-VALUE PICTURE 9(6).
    02  SSA32-END   PICTURE X       VALUE  ')'.
01  DISPLAY-PCB.
    02 DBN  PICTURE X(8).
    02 SL   PICTURE XX.
    02 SC   PICTURE XX.
       02 PO   PICTURE X(4).
    02 JCB  PICTURE S9(5) COMPUTATIONAL.
    02 SNF  PICTURE X(8).
    02 LOFK PICTURE S9(5) COMPUTATIONAL.
    02 NOSS PICTURE S9(5) COMPUTATIONAL.
    02 FKA.
        03  PK  PICTURE X(6).
        03  L2K PICTURE X(6).
        03  L3K PICTURE X(6).
        03  L22K    PICTURE X(6).
        03  L32K    PICTURE X(6).

01  SEGMENT-SWITCHES.
    02  P   PICTURE X.
    02  L21 PICTURE X.
    02  L31 PICTURE X.
    02  L22 PICTURE X.
    02  L32 PICTURE X.
01  ACTIVE-PCB.
    02  CS11    PICTURE X.
    02  CS12    PICTURE X.
    02  CS21    PICTURE X.
    02  CS22    PICTURE X.
LINKAGE SECTION.
01  PCBCASE11.
```

```
    02  DBD-NAME1   PICTURE X(8).
    02  SEG-LEVEL1  PICTURE XX.
    02  STATUS-CODES1   PICTURE XX.
    02  PROC-OPTIONS1   PICTURE X(4).
    02  DLI-JCB-ADDR1   PICTURE S9(5)   COMPUTATIONAL.
    02  SEGMENT-NAME-FEEDBACK1  PICTURE X(8).
    02  LENGTH-OF-FEEDBACK-KEY1 PICTURE S9(5)   COMPUTATIONAL.
    02  NUMBER-OF-SENSITIVE-SEGS1   PICTURE S9(5) COMPUTATIONAL.
    02  KEY-FEEDBACK-AREA1   PICTURE X(30).

01  PCBCASE12.
    02  DBD-NAME2   PICTURE X(8).
    02  SEG-LEVEL2  PICTURE XX.
    02  STATUS-CODES2   PICTURE XX.
    02  PROC-OPTIONS2   PICTURE X(4).
    02  DLI-JCB-ADDR2   PICTURE S9(5)   COMPUTATIONAL.
    02  SEGMENT-NAME-FEEDBACK2  PICTURE X(8).
    02  LENGTH-OF-FEEDBACK-KEY2 PICTURE S9(5)   COMPUTATIONAL.
    02  NUMBER-OF-SENSITIVE-SEGS2   PICTURE S9(5) COMPUTATIONAL.
    02  KEY-FEEDBACK-AREA2  PICTURE X(30).

01  PCBCASE21.
    02  DB-NAME3    PICTURE X(8).
    02  SEG-LEVEL3  PICTURE XX.
    02  STATUS-CODES3   PICTURE XX.
    02  PROC-OPTIONS3   PICTURE X(4).
    02  DLI-JCB-ADDR3   PICTURE S9(5)   COMPUTATIONAL.
    02  SEGMENT-NAME-FEEDBACK3  PICTURE X(8).
    02  LENGTH-OF-FEEDBACK-KEY3 PICTURE S9(5)   COMPUTATIONAL.
    02  NUMBER-OF-SENSITIVE-SEGS3   PICTURE S9(5) COMPUTATIONAL.
    02  KEY-FEEDBACK-AREA3  PICTURE X(30).

01  PCBCASE22.
    02  DB-NAME4    PICTURE X(8).
    02  SEG-LEVEL4  PICTURE XX.
    02  STATUS-CODES4   PICTURE XX.
    02  PROC-OPTIONS4   PICTURE X(4).
    02  DLI-JCB-ADDR4   PICTURE S9(5)   COMPUTATIONAL.
    02  SEGMENT-NAME-FEEDBACK4  PICTURE X(8).
    02  LENGTH-OF-FEEDBACK-KEY4 PICTURE S9(5)  COMPUTATIONAL.
    02  NUMBER-OF-SENSITIVE-SEGS4   PICTURE S9(5) COMPUTATIONAL.
    02  KEY-FEEDBACK-AREA4   PICTURE X(30).

PROCEDURE DIVISION.
BEGIN.
```

```
    ENTER LINKAGE.
        ENTRY 'DLITCBL' USING
        PCBCASE11,
        PCBCASE12,
        PCBCASE21,
        PCBCASE22.
    ENTER COBOL.
    DISPLAY 'HIBLSN01 STARTED'.
CARD-MESSAGE.
    OPEN INPUT CTLCRD.
    READ CTLCRD AT END GO TO EOJ.
    OPEN OUTPUT DISKI.
    MOVE DUMP-CASE1 TO DC011.
    MOVE DUMP-CASE2 TO DC022.
    WRITE INTREC.
    CLOSE DISKI.
    MOVE  'ISRT' TO  CALL-FUNC.
    IF CASE1 NOT = ' '  GO TO OPEN-TAPE.
    IF CASE2 NOT = ' '  GO TO OPEN-TAPE.
    GO TO EOJ.
OPEN-TAPE.
    OPEN INPUT INTAPE.
READ-TAPE.
    READ INTAPE AT END GO TO TAPE-END.
CK-CS11.
    IF C11 NOT = ' ' GO TO PC11.
CK-CS12.
    IF C12 NOT = ' ' GO TO PC12.
CK-CS21.
    IF C21 NOT = ' ' GO TO PC21.
CK-CS22.
    IF C22 NOT = ' ' GO TO PC22.
    GO TO READ-TAPE.
TAPE-END.
    CLOSE INTAPE.
EOJ.
    CLOSE CTLCRD.
        DISPLAY  'SUCCESSFUL END OF HIBLSN01'.
    ENTER LINKAGE.
    RETURN.
    ENTER COBOL.
PC11.
    MOVE KEY1 TO SSA1-VALUE.
    MOVE ' ' TO SSA1-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE11,
            PARENT,
            SSA1-NAME.
        ENTER COBOL.
    MOVE '(' TO SSA1-BEGIN.
    DISPLAY SSA1.
    DISPLAY PARENT. MOVE PCBCASE11 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY  'NO INSERT' GO TO READ-TAPE.
    MOVE KEY2 TO SSA21-VALUE.
    MOVE ' ' TO SSA21-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE11,
            LEVEL2,
            SSA1,
            SSA21-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA21-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA21.
    MOVE PCBCASE11 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    MOVE  KEY3 TO SSA31-VALUE.
    MOVE ' ' TO SSA31-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE11,
            LEVEL3,
            SSA1,
            SSA21,
            SSA31-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA31-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA21.
    DISPLAY SSA31.
    MOVE PCBCASE11 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    ADD 1 TO KEY2.
    MOVE KEY2 TO SSA22-VALUE.
    MOVE ' ' TO SSA22-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE11,
            LEVEL2,
            SSA1,
            SSA22-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA22-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA22.
    MOVE PCBCASE11 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    MOVE KEY3 TO SSA32-VALUE.
    MOVE ' ' TO SSA32-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE11,
            LEVEL3,
            SSA1,
            SSA22,
            SSA32-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA32-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA22.
    DISPLAY SSA32.
    MOVE PCBCASE11 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    GO TO CK-CS12.
PC12.
    MOVE KEY1 TO SSA1-VALUE.
    MOVE ' ' TO SSA1-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE12,
            PARENT,
            SSA1-NAME.
    ENTER COBOL.
```

```
MOVE '(' TO SSA1-BEGIN.
DISPLAY SSA1.
DISPLAY PARENT. MOVE PCBCASE12 TO DISPLAY-PCB.
PERFORM DISP.
IF SC NOT = ' ' DISPLAY  'NO INSERT' GO TO READ-TAPE.
MOVE KEY2 TO SSA21-VALUE.
MOVE ' ' TO SSA21-BEGIN.
ENTER LINKAGE.
    CALL 'CBLTDLI' USING CALL-FUNC,
        PCBCASE12,
        LEVEL2,
        SSA1,
        SSA21-NAME.
ENTER COBOL.
MOVE '(' TO SSA21-BEGIN.
DISPLAY SSA1.
DISPLAY SSA21.
MOVE PCBCASE12 TO DISPLAY-PCB.

PERFORM DISP.
IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
MOVE KEY3 TO SSA31-VALUE.
MOVE ' ' TO SSA31-BEGIN.
ENTER LINKAGE.
    CALL 'CBLTDLI' USING CALL-FUNC,
        PCBCASE12,
        LEVEL3,
        SSA1,
        SSA21,
        SSA31-NAME.
ENTER COBOL.
MOVE '(' TO SSA31-BEGIN.
DISPLAY SSA1.
DISPLAY SSA21.
DISPLAY SSA31.
MOVE PCBCASE12 TO DISPLAY-PCB.
PERFORM DISP.
IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
ADD 1 TO KEY2.
MOVE KEY2 TO SSA22-VALUE.
MOVE ' ' TO SSA22-BEGIN.
ENTER LINKAGE.
    CALL 'CBLTDLI' USING CALL-FUNC,
        PCBCASE12,
        LEVEL2,
        SSA1,
        SSA22-NAME.
ENTER COBOL.
MOVE '(' TO SSA22-BEGIN.
DISPLAY SSA1.
DISPLAY SSA22.
MOVE PCBCASE12 TO DISPLAY-PCB.
PERFORM DISP.
IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
MOVE KEY3 TO SSA32-VALUE.
MOVE ' ' TO SSA32-BEGIN.
ENTER LINKAGE.
    CALL 'CBLTDLI' USING CALL-FUNC,
        PCBCASE12,
        LEVEL3,
        SSA1,
        SSA22,
        SSA32-NAME.
ENTER COBOL.
MOVE '(' TO SSA32-BEGIN.
        DISPLAY SSA1.
        DISPLAY SSA22.
        DISPLAY SSA32.
        MOVE PCBCASE12 TO DISPLAY-PCB.
        PERFORM DISP.
        IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
        GO TO CK-CS21.
    PC21.
        MOVE KEY1 TO SSA1-VALUE.
        MOVE ' ' TO SSA1-BEGIN.
        ENTER LINKAGE.
            CALL 'CBLTDLI' USING CALL-FUNC,
                PCBCASE21,
                PARENT,
                SSA1-NAME.
        ENTER COBOL.
        MOVE '(' TO SSA1-BEGIN.
        DISPLAY SSA1.
        DISPLAY PARENT. MOVE PCBCASE21 TO DISPLAY-PCB.
        PERFORM DISP.
        IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
        MOVE KEY2 TO SSA21-VALUE.
        MOVE ' ' TO SSA21-BEGIN.
        ENTER LINKAGE.
            CALL 'CBLTDLI' USING CALL-FUNC,
                PCBCASE21,
                LEVEL2,
                SSA1,
                SSA21-NAME.
        ENTER COBOL.
        MOVE '(' TO SSA31-BEGIN.
        DISPLAY SSA1.
        DISPLAY SSA21.
        MOVE PCBCASE21 TO DISPLAY-PCB.
        PERFORM DISP.
        IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
        MOVE KEY3 TO SSA31-VALUE.
        MOVE ' ' TO SSA31-BEGIN.
        ENTER LINKAGE.
            CALL 'CBLTDLI' USING CALL-FUNC,
                PCBCASE21,
                LEVEL3,
                SSA1,
                SSA21,
                SSA31-NAME.
        ENTER COBOL.
        MOVE '(' TO SSA31-BEGIN.
        DISPLAY SSA1.
        DISPLAY SSA21.
        DISPLAY SSA31.
        MOVE PCBCASE21 TO DISPLAY-PCB.
        PERFORM DISP.
        IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
        ADD 1 TO KEY2.
        MOVE KEY2 TO SSA22-VALUE.
        MOVE ' ' TO SSA22-BEGIN.
        ENTER LINKAGE.
            CALL 'CBLTDLI' USING CALL-FUNC,
                PCBCASE21,
                LEVEL2,
                SSA1,
                SSA22-NAME.
        ENTER COBOL.
        MOVE '(' TO SSA22-BEGIN.
        DISPLAY SSA1.
    DISPLAY SSA22.
    MOVE PCBCASE21 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    MOVE KEY3 TO SSA32-VALUE.
    MOVE ' ' TO SSA32-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE21,
            LEVEL3,
            SSA1,
            SSA22,
            SSA32-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA32-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA22.
    DISPLAY SSA32.
    MOVE PCBCASE21 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    GO TO CK-CS22.
PC22.
    MOVE KEY1 TO SSA1-VALUE.
    MOVE ' ' TO SSA32-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE22,
            PARENT,
            SSA1-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA1-BEGIN.
    DISPLAY SSA1.
    DISPLAY PARENT. MOVE PCBCASE22 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    MOVE KEY2 TO SSA21-VALUE.
    MOVE ' ' TO SSA21-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE22,
            LEVEL2,
            SSA1,
            SSA21-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA21-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA21.
    MOVE PCBCASE22 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    MOVE KEY3 TO SSA31-VALUE.
    MOVE ' ' TO SSA31-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE22,
            LEVEL3,
            SSA1,
            SSA21,
            SSA31-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA31-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA21.
    DISPLAY SSA31.
    MOVE PCBCASE22 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    ADD 1 TO KEY2.
    MOVE KEY2 TO SSA22-VALUE.
    MOVE ' ' TO SSA22-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE22,
            SSA1,
            SSA22-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA22-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA22.
    MOVE PCBCASE22 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    MOVE KEY3 TO SSA32-VALUE.
    MOVE ' ' TO SSA32-BEGIN.
    ENTER LINKAGE.
        CALL 'CBLTDLI' USING CALL-FUNC,
            PCBCASE22,
            LEVEL3,
            SSA1,
            SSA22,
            SSA32-NAME.
    ENTER COBOL.
    MOVE '(' TO SSA32-BEGIN.
    DISPLAY SSA1.
    DISPLAY SSA22.
    DISPLAY SSA32.
    MOVE PCBCASE22 TO DISPLAY-PCB.
    PERFORM DISP.
    IF SC NOT = ' ' DISPLAY 'NO INSERT' GO TO READ-TAPE.
    GO TO READ-TAPE.
DISP.
    DISPLAY 'DATA BASE NAME = '  DBN.
    DISPLAY 'SEGMENT LEVEL = '    SL.
    DISPLAY 'STATUS CODES = '     SC.
    DISPLAY 'PROCESSING OPTIONS = '   PO.
    DISPLAY 'JCB ADDRESS = '      JCB.
    DISPLAY ' SEGMENT NAME FEEDBACK = '  SNF.
    DISPLAY 'LENGTH OF FEEDBACK KEY = ' LOFK.
    DISPLAY 'NUMBER OF SENSITIVE SEGMENTS = ' NOSS
    DISPLAY 'KEY FEEDBACK AREA = ' PL L2K L3K L22K L32K.
    DISPLAY 'PARENT = ' PT  'LEVEL21 = ' L21T 'LEVEL31 = ' L31T.
    DISPLAY 'LEVEL22 = ' L22T 'LEVEL32 = ' L32T.
```

A Program to Create Data for Load Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'CREATE'
AUTHOR.
REMARKS.  THIS PROGRAM CREATES TEST DATA THAT WILL ULTIMATELY BE
          LOADED INTO A PL/I DATA BASE.  A SEQUENTIAL FILE IS
          CREATED.  THE RECORDS ARE 440 BYTES LONG BLOCKED 8.  THE
          FORMAT IS:

          KEY1        FILLER1     KEY2        FILLER2
        | 6 BYTES   | BYTES     | 6 BYTES   | BYTES    |
                     KEY3        FILLER3
                    | 6 BYTES   | 253 BYTES |
          THE NUMBER OF RECORDS CREATED IS INDICATED BY PUNCHING IT
          IN THE FIRST FOUR COLUMNS OF A CONTROL CARD
          (RIGHT-JUSTIFIED).
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360.
OBJECT-COMPUTER. IBM-360.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT TAPEO ASSIGN TO 'TAPE' UTILITY.
    SELECT CTLCRD ASSIGN TO 'SYSIN' UTILITY.
DATA DIVISION.
FILE SECTION.
FD CTLCRD.
    LABEL RECORDS ARE OMITTED
    BLOCK CONTAINS 80 CHARACTERS
    RECORDING MODE F
    DATA RECORD  CARD-IN.
01  CARD-IN.
    02  CTL-NO    PICTURE 9(4).
    02  FILLER    PICTURE X(76).
FD  TAPEO
    LABEL RECORDS STANDARD
    BLOCK CONTAINS 8 RECORDS
    RECORDING MODE F
    DATA RECORD TAPE-OUT.
01  TAPE-OUT
    02  PARENT.
        03  KEY1          PICTURE X(6).
        03  FILLER1       PICTURE X(84).
        03  LEVEL2.
            04  KEY2      PICTURE X(6).
            04  FILLER2   PICTURE X(85).
            04  LEVEL3.
                05  KEY3      PICTURE X(6).
                05  FILLER3   PICTURE X(253).
WORKING-STORAGE  SECTION.
77  COUNT  PICTURE    99999  COMPUTATIONAL-3 VALUE  00000.
77  CONTROL1  PICTURE  9(6).
77  ADD-CONTROL   PICTURE 9(6).
77  SEC-LEVEL     PICTURE 9(6).
PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT CTLCRD.
    OPEN OUTPUT  TAPEO.
READ-CARD.
        READ CTLCRD AT END  GO TO EOJ.
BUILD.
    MOVE CTL-NO TO CONTROL1.
BUILD1.
    MOVE CONTROL1 TO KEY1.
  1 TO ADD-CONTROL
    MOVE CONTROL1 TO ADD-CONTROL.
    MOVE ADD-CONTROL TO KEY2.
    SUBTRACT 5 FROM ADD-CONTROL.
    MOVE ADD-CONTROL TO KEY3.
    MOVE ALL 'P' TO FILLER1.
    MOVE ALL 'S' TO FILLER2.
    MOVE ALL 'T' TO FILLER3.
WRITE-TAPE.
    WRITE TAPE-OUT.
CK-TOTAL-RECORDS.
    ADD 1 TO COUNT.
    IF COUNT = CTL-NO GO TO EOJ.
    ADD 10 TO CONTROL1.
```

```
        GO TO BUILD1.
    EOJ.
        CLOSE CTLCRD.
        CLOSE TAPEO.
        DISPLAY 'TOTAL RECORDS WRITTEN = 'COUNT.
        STOP RUN.
```

Batch (Update) Processing Program

```
00001  IDENTIFICATION DIVISION.
00002  PROGRAM-ID. 'HIBASN03'
00003  AUTHOR.
00004  REMARKS.    THIS PROGRAM ALLOWS THE PROGRAMMER TO RUN IN
00005              THE BATCH ENVIRONMENT.  THE FUNCTION TO BE
00006              PERFORMED IS ENTERED THRU CARDS IN THE FORM
00007              CALL FUNC SEGNAME(QUAL)SEGMENT(QUAL)SEGMENT(QUAL)
00010              ANY REPLACED RECORDS WILL BE FILLED WITH R'S
00011  ENVIRONMENT DIVISION.
       See Chapter 5, Figure 31.
00012  CONFIGURATION SECTION.
00013  SOURCE-COMPUTER. IBM-360.
00014  OBJECT-COMPUTER. IBM-360.
00015  INPUT-OUTPUT SECTION.
00016  FILE-CONTROL.
00017      SELECT CARDS TO ASSIGN TO 'CARDS' UTILITY.
00018  DATA DIVISION.
00019  FILE SECTION.
00020  FD CARDS
00021      LABEL RECORDS OMITTED
00022      BLOCK CONTAINS 80 CHARACTERS
00023      RECORDING MODE  F
00024      DATA RECORD CARD-IN.
00025  01  CARD-IN PICTURE  X(80).
       See Chapter 5, Figure 31, Ref 1.
00026  WORKING-STORAGE SECTION.
00027  01  WORK-AREA.
00028      02  FUNC    PICTURE X(4).
00029      02  FILLER  PICTURE X.
00030      02  SEG1    PICTURE X(8).
00031      02  QUAL1.
00032          03  LF1 PICTURE X.
00033              88  PAR1    VALUE '('.
00034          03  LFD1    PICTURE X(10).
00035          03  VALUE1  PICTURE  X(6).
00036          03  FILLER  PICTURE X.
00037      02  SEG2    PICTURE X(8).
00038      02  QUAL2.
00039          03  LF2 PICTURE  X.
00040              88 PAR2     VALUE '('.
00041          03  FLD2    PICTURE X(10).
00042          03  VALUE2  PICTURE  X(6).
00043          03  FILLER  PICTURE  X.
00044      02  SEG3    PICTURE  X(8).
00045      02  QUAL3.
00046          03 LF3 PICTURE X.
00047              88 PAR3   VALUE '('.
00048          03  FLD3  PICTURE  X(10).
00049          03  VALUE3  PICTURE  X(6).
00050          03  FILLER  PICTURE X.
00051  01  SAVE1   PICTURE X(4).
00052  01  WA-1.
00053      02  CALL-FUNC  PICTURE  X(4)   VALUE  'GU '.
00054  01  SSA1.
       See Chapter 5, Figure 30, Ref 2.
```

```
00055       02  SEG1-NAME   PICTURE X(8).
00056       02  SSA1-QUAL   PICTURE X(18).
00057   01  SSA2.
00058       02  SEG2-NAME   PICTURE X(8).
00059       02  SSA2-QUAL   PICTURE X(18).
00060   01  SSA3.
00061       02  SEG3-NAME   PICTURE X(8).
00062       02  SSA3-QUAL   PICTURE X(18).
00063   01  I-O-AREA.
        See Chapter 5, Figure 31, Ref 3.
00064       02  KEY1 PICTURE X(6).
00065       02  AREA1    PICTURE X(40).
00066       02  AREA2    PICTURE X(34).
00067       02  AREA3    PICTURE X(220).
00068   01  DISP-MESS.
00069       02  MESS     PICTURE X(40).
00070   01  SW  PICTURE  X  VALUE ' '.
00071   01  SSA-SW  PICTURE  9 VALUE 0.
00072   01  FUNC1   PICTURE  XXXX.
00073
00074   LINKAGE SECTION.
        See Chapter 5, Figure 31, Ref 4.
00075   01  PCBCASE11.
00076       02  DBD-NAME1   PICTURE X(8).
00077       02  SEG-LEVEL1  PICTURE XX.
00078       02  STATUS-CODES1   PICTURE XX.
00079       02  PROC-OPTIONS1   PICTURE X(4).
00080       02  DLI-JCB-ADDR1   PICTURE S9(5)   COMPUTATIONAL.
00081       02  SEGMENT-NAME-FEEDBACK1  PICTURE X(8).
00082       02  LENGTH-OF-FEEDBACK-KEY1 PICTURE S9(5) COMPUTATIONAL.
00083       02  NUMBER-OF-SENSITIVE-SEGS1   PICTURE S9(5)
                COMPUTATIONAL.
00084       02  KEY-FEEDBACK-AREA1   PICTURE X(30).

00095   01  PCBCASE1.
00096       02  DB-NAME3    PICTURE X(8).
00097       02  SEG-LEVEL3  PICTURE XX.
00098       02  STATUS-CODES3   PICTURE XX.
00099       02  PROC-OPTIONS3   PICTURE X(4).
00100       02  DLI-JCB-ADDR3   PICTURE S9(5)  COMPUTATIONAL.
00101       02  SEGMENT-NAME-FEEDBACK3  PICTURE X(8).
00102       02  LENGTH-OF-FEEDBACK-KEY3 PICTURE S9(5) COMPUTATIONAL.
00103       02  NUMBER-OF-SENSITIVE-SEGS3   PICTURE S9(5)
                COMPUTATIONAL.
00104       02  KEY-FEEDBACK-AREA3   PICTURE X(30).

00115   PROCEDURE DIVISION.
00116   BEGIN.
        See Chapter 5, Figure 31, Ref 4.
00117       ENTER LINKAGE.
00118           ENTRY  'DLITCBL' USING
00119           PCBCASE11,
00120           PCBCASE21.
00121       ENTER COBOL.
00122   OPEN-CARDS.
00123       OPEN INPUT CARDS.
00124       MOVE '1' TO SW.
00125   READ-CARDS.
00126       READ CARDS AT END GO TO EOJ.
00127       MOVE CARD-IN TO I-O-AREA.
00128       READ CARDS AT END GO TO EOJ.
00129       MOVE CARD-IN TO AREA3.
00130   MOVE-I-O-AREA.
00131       MOVE I-O-AREA TO WORK-AREA.
00132   CHECK-FUNC.
```

```
00133      IF (FUNC = 'ISRT') OR (FUNC =  'GU  ')
00134      OR (FUNC = 'GNP ')
00135       OR (FUNC = 'GN ') OR (FUNC = 'DLET')
00136       OR (FUNC =  'REPL') GO TO SET-UP-SSA. MOVE SPACES TO MESS.
00137      MOVE  'IMPROPER CALL FUNCTION SPECIFIED' TO MESS.
00138      IF SW = '1' DISPLAY MESS GO TO READ-CARDS.
00139      GO TO EOJ.
00140  SET-UP-SSA.
00141      MOVE SPACES TO SSA1, SSA2, SSA3.
00142      IF SEG1  NOT = '          ' MOVE SEG1 to SEG1-NAME
00143      ELSE MOVE   1 TO SSA-SW   GO TO EXIT1.
00144      IF PAR1 MOVE QUALI TO  SSA1-QUAL ELSE
00145      MOVE SPACES TO SSA1-QUAL.
00146      IF  SEG2  NOT='    ' MOVE TO SEG2=NAME
00147      ELSE MOVE   TO SSA-SW  GO TO EXIT1.
00148      IF PAR2 MOVE QUAL2 TO SSA2-QUAL ELSE
00149      MOVE SPACES TO SSA2-QUAL.
00150      IF SEG3 NOT =  '          ' MOVE SEG3 TO SEG3-NAME
00151      ELSE MOVE  3  to SSA-SW   GO TO EXIT1.
00152      IF PAR3 MOVE QUAL3 TO SSA3-QUAL ELSE
00153      MOVE SPACES TO SSA3-QUAL.
00154      MOVE 4 TO SSA-SW.
00155  EXIT1.
00156      IF FUNC = 'ISRT'  MOVE ALL 'I' TO I-O-AREA.
00157      IF (FUNC = 'ISRT') AND (SSA-SW = 2) MOVE VALUE1 to KEY1
00158          MOVE SPACES TO SSA1-QUAL.
00159      IF (FUNC = 'IRST') AND (SSA-SW =3) MOVE VALUE2 TO KEY1
00160          MOVE SPACES TO SSA2-QUAL.
00161      IF (FUNC = 'ISRT') AND (SSA-SW =4) MOVE VALUE3 TO KEY1
00162          MOVE SPACES TO SSA3-QUAL.
00163      IF FUNC = 'REPL' GO TO GHP.
00164      IF FUNC = 'DLET' GO TO GHP.
00165      IF SSA-SW = 1  PERFORM  CALL-NO-SSA.
00166      IF SSA-SW = 2  PERFORM  CALL-ONE-SSA.
00167      IF SSA-SW = 3  PERFORM  CALL-TWO-SSA.
00168      IF SSA-SW = 4  PERFORM  CALL-THREE-SSA.
00169      CK (in line with Exit 1)
00170      IF STATUS-CODES1 = ' ' MOVE SPACES TO MESS
00171      MOVE 'SUCCESSFUL OPERATION' TO MESS
00172      ELSE MOVE SPACES TO MESS
00173      MOVE 'UNSUCCESSFUL OPERATION CHECK STATUS' TO MESS
00174      GO TO RD-CK.
00175  RD-DISP.
00176      DISPLAY ' '.
00177      DISPLAY DISP-MESS.
00178      DISPLAY STATUS-CODES1.
00179      DISPLAY WORK-AREA.
00180      DISPLAY KEY1, AREA1.
00181  EOJ.
00182      CLOSE CARDS.
00183      DISPLAY 'SUCCESSFUL END OF HIMASN01'.
00184      ENTER LINKAGE.
           See Chapter 5, Figure 31, Ref 10.
00185          RETURN.
00186      ENTER COBOL.
00187  GHP.
00188      MOVE FUNC TO FUNC1.
```

```
00189      MOVE 'GHU ' TO FUNC.
00190      IF SSA-SW = 2 PERFORM CALL-ONE-SSA
00191      MOVE FUNC1 TO FUNC PERFORM CK-REPL
00192      PERFORM CALL-NO-SSA.
00193      IF SSA-SW = 3 PERFORM CALL-TWO-SSA
00194      MOVE FUNC1 TO FUNC PERFORM CK-REPL
00195      PERFORM CALL-NO-SSA.

00196      IF SSA-SW = 4 PERFORM CALL-THREE-SSA
00197      MOVE FUNC1 TO FUNC PERFORM CK-REPL
00198      PERFORM CALL-NO-SSA.
00199      GO TO CK.
00200  RD-CK.
00201      IF SW = '1' PERFORM RD-DISP GO TO READ-CARDS.
00202      GO TO EOJ.
00203  CK-REPL.
00204      IF FUNC = 'REPL' MOVE ALL 'R' TO AREA1.
00205
00206  CALL-NO-SSA.
00207      ENTER LINKAGE.
00208      CALL 'CBLTDLI' USING
           See Chapter 5, Figure 31, Ref 8.
00209          FUNC,
00210          PCBCASE11,
00211          I-O-AREA.
00212      ENTER COBOL.
00213  CALL-ONE-SSA.
00214      ENTER LINKAGE.
00215      CALL 'CBLTDLI' USING
           See Chapter 5, Figure 31, Ref 6 & 7.
00216          FUNC,
00217          PCBCASE11,
00218          I-O-AREA,
00219          SSA1.
00220      ENTER COBOL.
00221  CALL-TWO-SSA.
00222      ENTER LINKAGE.
00223      CALL 'CBLTDLI' USING
00224          FUNC,
00225      PCBCASE11,
00226          I-O-AREA,
00227          SSA1,
00228          SSA2.
00229      ENTER COBOL.
00230  CALL-THREE-SSA.
00231      ENTER LINKAGE.
00232          CALL 'CBLTDLI' USING
00233              FUNC,
00234              PCBCASE11,
00235              I-O-AREA,
00236              SSA1,
00237              SSA2,
00238              SSA3.
00239          ENTER COBOL.
00240
```

Card Data for COBOL Batch Program

```
GN

GN

GN
```

```
GN

GNP

GN

GU    PARENT  (KEY1   =000010)
GNP

GNP

GNP

GNP

GU    PARENT  (KEY1   =000015)

GN    LEVEL22

GU    PARENT  (KEY1   =000010)LEVEL22 (KEY22  =000012)LEVEL32
(KEY32    =000006)
GU    PARENT  (KEY1   =000010)LEVEL22 (KEY22  =000015)

ISRT  PARENT  (KEY    =000015)

ISRT  PARENT  (KEY1   =000016)

REPL  PARENT  (KEY1   =000016)

GU    PARENT  (KEY1   =000015)

DLET  PARENT  (KEY1   =000015)
```

Data Base Reorganization (Dump) Program

```
PROGRAM-ID. 'HIBASN01'
IDENTIFICATION DIVISION.
AUTHOR.
REMARKS.
      THIS PROGRAM IS THE SECOND STEP OF A TWO STEP JOB. TWO OF
      THE FOUR TEST DATA CAN BE DUMPED IN ANY COMBINATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360.
OBJECT-COMPUTER.  IBM-360.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
      SELECT CTLCRD ASSIGN TO 'DISKI' UTILITY.
DATA DIVISION.
FILE SECTION.
FD CTLCRD
      LABEL RECORDS OMITTED
      BLOCK CONTAINS 80 CHARACTERS
      RECORDING MODE  F
      DATA RECORD CARD-IN.
01    CARD-IN.
      02  CTL.
          03  C11 PICTURE X.
          03  C12 PICTURE X.
          03  C21 PICTURE X.
          03  C22 PICTURE X.
      02  FILLER  PICTURE X(76).
WORKING-STORAGE SECTION.
77 DUMP1-SW   PICTURE 9.
```

```
77 BRANCH-V   PICTURE 99.
77 PSBNAME    PICTURE X(8)    VALUE 'HIBASN01'.
01 CALL-FUNC  PICTURE X(4)    VALUE  'LOAD'.
01 SSA1.
   02 SSA1-NAME   PICTURE X(8)    VALUE   'KEY1    '.
   02 SSA1-BEGIN  PICTURE X       VALUE   '('.
   02 SSA1-KEY    PICTURE X(10)   VALUE   'KEY1     ='.
   02 SSA1-VALUE  PICTURE 9(6).
   02 SSA1-END    PICTURE X       VALUE   ')'.
01 SSA21.
   02 SSA21-NAME  PICTURE X(8)    VALUE   'LEVEL21 '.
   02 SSA21-BEGIN PICTURE X       VALUE   '('.
   02 SSA21-KEY   PICTURE X(10)   VALUE   'KEY21    ='.
   02 SSA21-VALUE PICTURE 9(6).
   02 SSA21-END   PICTURE X       VALUE   ')'.
01 SSA31.
   02 SSA31-NAME  PICTURE X(8)    VALUE   'LEVEL31 '.
   02 SSA31-BEGIN PICTURE X       VALUE   '('.
   02 SSA31-KEY   PICTURE X(10)   VALUE   'KEY31    ='.
   02 SSA31-END   PICTURE X       VALUE   ')'.
01 SSA22.
   02 SSA22-NAME  PICTURE X(8)    VALUE   'LEVEL22 '.
   02 SSA22-BEGIN PICTURE X       VALUE   '('.
   02 SSA22-KEY   PICTURE X(10)   VALUE   'KEY22    ='.
   02 SSA22-VALUE PICTURE 9(6).
   02 SSA22-END   PICTURE X       VALUE   ')'.
01 SSA32.
   02 SSA32-NAME  PICTURE X(8)    VALUE   'LEVEL32 '.
   02 SSA32-BEGIN PICTURE X       VALUE   '('.
   02 SSA32-KEY   PICTURE X(10)   VALUE   'KEY32    ='.
   02 SSA32-VALUE PICTURE 9(6).
   02 SSA32-END   PICTURE X       VALUE   ')'.
01 USER-SEG   PICTURE X(440).
01 WORK-PCB.
   02 DBD-NAME                    PICTURE X(8).
   02 SEG-LEVEL                   PICTURE 99.
   02 STATUS-CODES                PICTURE XX.
   02 PROC-OPTIONS                PICTURE X(4).
   02 DLI-JCB-ADDR    PICTURE S9(5)   COMPUTATIONAL.
   02 SEG-NAME                    PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEY  PICTURE S9(5) COMPUTATIONAL.
   02 NO-OF-SENSITIVE-SEGS    PICTURE S9(5) COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA       PICTURE X(30).

01 DISPLAY-PCB.
   02 DBN       PICTURE X(8).
   02 SL        PICTURE XX.
   02 SC        PICTURE XX.
   02 PO        PICTURE X(4).
   02 JCB       PICTURE S9(5)   COMPUTATIONAL.
   02 SNF       PICTURE X(8).
   02 LOFK      PICTURE S9(5)   COMPUTATIONAL.
   02 NOSS      PICTURE S9(5)   COMPUTATIONAL.
   02 KFA.
      03   PK   PICTURE X(6).
      03   L2K  PICTURE X(6).
      03   L3K  PICTURE X(6).
      03   L22K     PICTURE X(6).
      03   L32K     PICTURE X(6).

LINKAGE SECTION.
01 PCBCASE11.
   02 DBD-NAME1     PICTURE X(8).
   02 SEG-LEVEL1    PICTURE XX.
   02 STATUS-CODES1     PICTURE XX.
   02 PROC-OPTIONS1   PICTURE X(4).
   02 DLI-JCB-ADDR1   PICTURE S9(5)   COMPUTATIONAL.
   02 SEGMENT-NAME-FEEDBACK1  PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEY1 PICTURE S9(5)  COMPUTATIONAL.
   02 NUMBER-OF-SENSITIVE-SEGS1   PICTURE S9(5) COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA1   PICTURE X(30).

01 PCBCASE12.
   02 DBD-NAME2   PICTURE X(8).
   02 SEG-LEVEL2  PICTURE XX.
   02 STATUS-CODES2   PICTURE XX.
   02 PROC-OPTIONS2   PICTURE X(4).
   02 DLI-JCB-ADDR2   PICTURE S9(5)   COMPUTATIONAL.
   02 SEGMENT-NAME-FEEDBACK2  PICTURE X(8).
   02 NUMBER-OF-SENSITIVE-SEGS2   PICTURE S9(5)   COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA2   PICTURE X(30).

01 PCBCASE21.
   02 DB-NAME3    PICTURE X(8).
   02 SEG-LEVEL3  PICTURE XX.
   02 STATUS-CODES3   PICTURE XX.
   02 PROC-OPTIONS3   PICTURE X(4).
   02 DLI-JCB-ADDR3   PICTURE S9(5)   COMPUTATIONAL.
   02 SEGMENT-NAME-FEEDBACK3  PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEY3 PICTURE S9(5)  COMPUTATIONAL.
   02 NUMBER-OF-SENSITIVE-SEGS3   PICTURE S9(5)   COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA3   PICTURE X(30).

01 PCBCASE22.
   02 DB-NAME4    PICTURE X(8).
   02 SEG-LEVEL4  PICTURE XX.
   02 STATUS-CODES4   PICTURE XX.
   02 PROC-OPTIONS4   PICTURE X(4).
   02 DLI-JCB-ADDR4   PICTURE S9(5)   COMPUTATIONAL.
   02 SEGMENT-NAME-FEEDBACK4  PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEY4 PICTURE S9(5)   COMPUTATIONAL.
   02 NUMBER-OF-SENSITIVE-SEGS4   PICTURE S9(5)   COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA4   PICTURE X(30).

01 PCBDUMP1.
   02 DBD-NAMED1                  PICTURE X(8).
   02 SEG-LEVELD1                   PICTURE XX.
   02 STATUS-CODESD1                PICTURE XX.
   02 PROC-OPTIONSD1                PICTURE X(4).
   02 DLI-JCB-ADDRD1  PICTURE S9(5)   COMPUTATIONAL.
   02 SEG-NAME-FEEDBACKD1           PICTURE X(8).

   02 LENGTH-OF-FEEDBACK-KEYD1      PICTURE S9(5) COMPUTATIONAL.
   02 NO-OF-SENSITIVE-SEGSD1        PICTURE S9(5) COMPUTATIONAL.
   02 KEY-FEEDBACK-AREAD1           PICTURE X(30).


01 PCBDUMP2.
   02 DBD-NAMED2                    PICTURE X(8).
   02 SEG-LEVELD2                   PICTURE XX.
   02 STATUS-CODESD2                PICTURE XX.
   02 PROC-OPTIONSD2                PICTURE X(4).
   02 DLI-JCB-ADDRD2  PICTURE S9(5)   COMPUTATIONAL.
   02 SEG-NAME-FEEDBACKD2           PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEYD2      PICTURE S9(5) COMPUTATIONAL.
   02 NO-OF-SENSITIVE-SEGSD2        PICTURE S9(5) COMPUTATIONAL.
   02 KEY-FEEDBACK-AREAD2           PICTURE X(30).

PROCEDURE DIVISION.
BEGIN.
   ENTER LINKAGE.
      ENTRY 'DLITCBL' USING
          PCBCASE11,
          PCBCASE12,
          PCBCASE21,
          PCBCASE22,
          DUMP1,
          DUMP2.
   ENTER COBOL.
CTL-OPEN.
   OPEN INPUT CTLCRD.
CTL-READ.
   READ CTLCRD AT END GO TO EOJ.
CHECK.
   IF C11 NOT = ' '
   MOVE 1 TO DUMP1-SW
   ALTER GET-ANOTHER TO PROCEED TO GET-CASE11
   ALTER RETURN-TO TO PROCEED TO GO-TO-VECTOR
   MOVE  ' ' TO C11
   GO TO GET-ANOTHER.
   IF C12 = ' ' NEXT SENTENCE
   ELSE IF DUMP1-SW = 1 ALTER GET-ANOTHER TO PROCEED TO
   GET-CASE12
   ALTER RETURN-TO TO PROCEED TO GO-TO-VECTOR1
   MOVE  ' ' TO C12
   GO TO GET-ANOTHER  ELSE ALTER GET-ANOTHER TO PROCEED TO
   GET-CASE12
   ALTER  RETURN-TO TO PROCEED TO GO-TO-VECTOR
   MOVE  ' ' TO C12  MOVE Figure 1 TO DUMP1-SW
   GO TO GET-ANOTHER.
   IF C21 = ' ' NEXT SENTENCE
   ELSE IF DUMP1-SW = 1 ALTER GET-ANOTHER TO PROCEED TO
   GET-CASE21
   ALTER RETURN-TO TO PROCEED TO GO-TO-VECTOR1
   MOVE  ' ' TO C21
   GO TO GET-ANOTHER ELSE
   ALTER GET-ANOTHER TO PROCEED TO GET-CASE21
   ALTER RETURN-TO TO PROCEED TO GO-TO-VECTOR
   MOVE  ' ' TO C21 MOVE 1 TO DUMP1-SW
   GO TO GET-ANOTHER.
   IF C22 = ' ' NEXT SENTENCE
   ELSE IF DUMP1-SW = 1 ALTER GET-ANOTHER TO PROCEED TO
   GET-CASE22
   ALTER RETURN-TO TO PROCEED TO GO-TO-VECTOR1
   MOVE ' ' TO C22
   GO TO GET-ANOTHER ELSE
   ALTER GET-ANOTHER TO PROCEED TO GET-CASE22
   ALTER  RETURN-TO TO PROCEED TO GO-TO-VECTOR
   MOVE  ' ' TO C22
   GO TO GET-ANOTHER.
EOJ.
   IF CTL NOT = '     ' GO TO CHECK.
   CLOSE CTLCRD.
   DISPLAY 'SUCCESSFUL END OF HIBASN01'.
   ENTER LINKAGE.
      RETURN.
   ENTER COBOL.
GET-CASE22.
   MOVE  'GN  ' TO CALL-FUNC.  MOVE SPACES TO USER-SEG.
   ENTER LINKAGE.
          CALL 'CBLTDLI' USING CALL-FUNC,
              PCBCASE22,
              USER-SEG.
   ENTER COBOL.
   DISPLAY USER-SEG.
   MOVE PCBCASE22 TO DISPLAY-PCB. WORK-PCB.
   PERFORM DISP.
   IF SC = 'GB' GO TO EOJ.
   GO TO RETURN-TO.
GET-CASE21.
   MOVE  'GN  ' TO CALL-FUNC.  MOVE SPACES TO USER-SEG.
   ENTER LINKAGE.
          CALL 'CBLTDLI' USING  CALL-FUNC,
              PCBCASE21,
              USER-SEG.
   ENTER COBOL.
   DISPLAY USER-SEG.
   MOVE PCBCASE21 TO DISPLAY-PCB, WORK-PCB.
   PERFORM DISP.
   IF SC = 'GB' GO TO EOJ.
   GO TO RETURN-TO.
GET-CASE11.
   MOVE  'GN  ' TO CALL-FUNC.  MOVE SPACES TO USER-SEG.
   ENTER LINKAGE.
          CALL 'CBLTDLI' USING CALL-FUNC,
              PCBCASE11,
              USER-SEG.
   ENTER COBOL.
   DISPLAY USER-SEG.
   MOVE PCBCASE11 TO DISPLAY-PCB.
   MOVE PCBCASE11 TO WORK-PCB.
   PERFORM DISP.
   IF SC = 'GB' GO TO EOJ.
   GO TO RETURN-TO.
GET-CASE12.
   MOVE  'GN  ' TO CALL-FUNC.  MOVE SPACES TO USER-SEG.
   ENTER LINKAGE.
          CALL 'CBLTDLI' USING CALL-FUNC,
              PCBCASE12,
              USER-SEG.
   ENTER COBOL.
   DISPLAY USER-SEG.
   MOVE PCBCASE12 TO DISPLAY-PCB, WORK-PCB.
   PERFORM DISP.
   IF SC = 'GB' GO TO EOJ.
   GO TO RETURN-TO.
GO-TO-VECTOR1.
   MOVE SEG-LEVEL TO BRANCH-V.
   GO TO LEVEL-ONE  LEVEL-TWO LEVEL-THREE DEPENDING BRANCH-V.
   DISPLAY 'NO LEVEL RETURNED'.
   STOP RUN.
LEVEL-ONE1.
   IF SEG-NAME  NOT = SSA1-NAME
      DISPLAY 'LEVEL1 ERROR'
      STOP RUN.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE  ' ' TO SSA1-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
      PCBDUMP2,
      USER-SEG,
      SSA1-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA1-BEGIN.
   DISPLAY SSA1.
   MOVE PCBDUMP2 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT = '  ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-TWO1.
   IF SEG-NAME = SSA21-NAME NEXT SENTENCE ELSE
   IF SEG-NAME = SSA22-NAME  GO TO LEVEL-TWO-TWO
      ELSE DISPLAY 'LEVEL1 ERROR'
      STOP RUN.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE  PARENT-KEY TO SSA1-VALUE.
   MOVE  ' ' TO SSA21-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
      PCBDUMP2,
      USER-SEG,
      SSA1,
      SSA21-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA21-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA21.
   MOVE PCBDUMP1 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT = '  ' DISPLAY  'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-TWO-TWO1.
   MOVE  'ISRT' TO CALL-FUNC.
   MOVE  PARENT-KEY TO SSA1-VALUE.
   MOVE  ' ' TO SSA22-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP2,
          USER-SEG,
          SSA1,
          SSA22-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA22-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA22.
   MOVE PCBDUMP2 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT = '  ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-THREE1.
   IF SEG-NAME = SSA31-NAME NEXT SENTENCE  ELSE
   IF SEG-NAME = SSA32-NAME GO TO LEVEL-THREE-TWO
   ELSE DISPLAY 'LEVEL3 ERROR'
   STOP RUN.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE PARENT-KEY TO SSA1-VALUE.
   MOVE LEVEL21-KEY TO SSA21-VALUE.
   MOVE  ' ' TO SSA31-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP2,
          USER-SEG,
          SSA1,
          SSA21,
          SSA31-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA31-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA21.
   DISPLAY SSA31.
   MOVE PCBDUMP2 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT = ' ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-THREE-TWO2.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE  PARENT-KEY TO SSA1-VALUE.
   MOVE  LEVEL22-KEY TO SSA22-VALUE.
   MOVE  ' ' TO SSA32-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP2,
          USER-SEG,
          SSA1,
          SSA22,
          SSA32-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA32-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA22.
   DISPLAY SSA32.
   MOVE PCBDUMP2 TO DISPLAY-PCB.
   DISPLAY  USER-SEG.
   PERFORM DISP.
   IF SC NOT =  '  ' DISPLAY  'NOT DUMPED'.
   GO TO GET-ANOTHER.
GO-TO-VECTOR.
   MOVE SEG-LEVEL TO BRANCH-V.
   GO TO LEVEL-ONE  LEVEL-TWO-LEVEL-THREE DEPENDING BRANCH-V.
   DISPLAY 'NO LEVEL RETURNED'.
   STOP RUN.
LEVEL-ONE.
   IF SEG-NAME  NOT  = SSA1-NAME
      DISPLAY 'LEVEL1 ERROR'
      STOP RUN.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE  ' ' TO SSA1-BEGIN.
   ENTER LINKAGE.
```

```
      CALL 'CBLTDLI' USING CALL-FUNC,
      PCBDUMP1,
      USER-SEG,
      SSA1-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA1-BEGIN.
   DISPLAY SSA1.
   MOVE PCBDUMP1 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT = '  ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-TWO.
   IF SEG-NAME  = SSA21-NAME NEXT SENTENCE ELSE
   IF SEG-NAME  = SSA22-NAME  GO TO LEVEL-TWO-TWO
      ELSE  DISPLAY 'LEVEL1 ERROR'
      STOP RUN.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE  PARENT-KEY TO SSA1-VALUE.
   MOVE  ' ' TO SSA21-BEGIN.
   ENTER LINKAGE.
          CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP1,
          USER-SEG,
          SSA1,
          SSA21-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA21-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA21.
   MOVE PCBDUMP1 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT =  '  ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-TWO-TWO.
   MOVE  'ISRT' TO CALL-FUNC.
   MOVE  PARENT-KEY TO SSA1-VALUE.
   MOVE  ' ' TO SSA22-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP1,
          USER-SEG,
          SSA1,
          SSA22-NAME.
   ENTER COBOL.
   MOVE  '(' TO SSA22-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA22.
   MOVE PCBDUMP1 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT =  '  ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-THREE.
   IF SEG-NAME = SSA31-NAME NEXT SENTENCE ELSE
   IF SEG-NAME = SSA32-NAME GO TO LEVEL-THREE-TWO
   ELSE DISPLAY 'LEVEL3 ERROR'
   STOP RUN.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE PARENT-KEY TO SSA1-VALUE.
   MOVE LEVEL21-KEY TO SSA21-VALUE.
   MOVE ' ' TO SSA31-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP1,
          USER-SEG,
          SSA1,
          SSA21,
          SSA31-NAME.
   ENTER COBOL.
   MOVE '(' TO SSA31-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA21.
   DISPLAY SSA31.
   MOVE PCBDUMP1 TO DISPLAY-PCB.
   PERFORM DISP.
   IF SC NOT = '  ' DISPLAY 'NOT DUMPED'.
   GO TO GET-ANOTHER.
LEVEL-THREE-TWO.
   MOVE 'ISRT' TO CALL-FUNC.
   MOVE  PARENT-KEY TO SSA1-VALUE.
   MOVE  LEVEL22-KEY TO SSA22-VALUE.
   MOVE  ' ' TO SSA32-BEGIN.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING CALL-FUNC,
          PCBDUMP1,
          USER-SEG,
          SSA1,
          SSA22,
          SSA32-NAME.
   ENTER COBOL.
   MOVE '(' TO SSA32-BEGIN.
   DISPLAY SSA1.
   DISPLAY SSA22.
   DISPLAY SSA32.
   MOVE PCBDUMP1 TO DISPLAY-PCB.
   DISPLAY  USER-SEG.
   PERFORM DISP.
   IF SC NOT =  '   ' DISPLAY  'NOT DUMPED'.
   GO TO GET-ANOTHER.
DISP.
   DISPLAY 'DATA BASE NAME = '    DBN.
   DISPLAY 'SEGMENT LEVEL = '     SL.
   DISPLAY 'STATUS CODES = '      SC.
   DISPLAY 'PROCESSING OPTIONS = '  PO.
   DISPLAY 'JCB ADDRESS = '       JCB.
   DISPLAY 'SEGMENT NAME FEEDBACK = ' SNF.
   DISPLAY 'LENGTH OF FEEDBACK KEY = ' LOFK.
   DISPLAY 'NUMBER OF SENSITIVE SEGMENTS = ' NOSS.
   DISPLAY 'KEY FEEDBACK AREA = ' PK L2K L3K L22K L32K.
   DISPLAY 'PARENT = ' PT  'LEVEL21 =  ' L21T 'LEVEL31 =' L31T.
   DISPLAY 'LEVEL22 = ' L22T 'LEVEL32 = ' L32T.
GET-ANOTHER.
   GO TO BEGIN.
RETURN-TO.
   GO TO BEGIN.
```

<u>Message (Update) Processing Program</u>

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  'HIMASN01'
AUTHOR.
REMARKS.  THIS PROGRAM ALLOWS THE PROGRAMMER TO RUN AN ON-LINE
          ENVIRONMENT.  THE FUNCTION TO BE PERFORMED IS ENTERED THROUGH
          THE TERMINAL IN THE FORM TRANSACTION CODE FUNCTION
          SEGMENT(QUAL)SEGMENT (QUAL)SEGMENT(QUAL).  ANY REPLACED
          RECORDS WILL BE FILLED WITH R'S.  ANY INSERTED RECORDS WILL
          BE FILLED WITH I'S.  GET HOLD FUNCTIONS ARE GENERATED WITHIN
          THIS PROGRAM WHEN FUNCTION EQUALS DELETE OR REPLACE.
          FUNCTION SHOULD BE ENTERED AS PROPER FOR CHARACTER CODE.
ENVIRONMENT DIVISION.
```

```
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360.
OBJECT-COMPUTER.  IBM-360.
INPUT-OUTPUT SECTION.
DATA DIVISION
WORKING-STORAGE SECTION.
See Chapter 5, Figure 33, Ref 1.
01 WORK-AREA.
   02 FUNC    PICTURE X(4).
   02 FILLER  PICTURE X.
   02 SEG1    PICTURE X(8).
   02 QUAL1.
      03  LF1 PICTURE X.
          88  PAR1    VALUE  '('.
      03  FLD1    PICTURE  X(10).

      03  VALUE1  PICTURE X(6).
      03  FILLER  PICTURE X.
   02 SEG2    PICTURE X(8).
   02 QUAL2.
      03  LF2 PICTURE X.
          88  PAR2    VALUE  '('.
      03  FLD2    PICTURE X(10).
      03  VALUE2  PICTURE X(6).
      03  FILLER  PICTURE X.
   02 SEG3    PICTURE X(8).
   02 QUAL3.
      03  LF3 PICTURE X.
          88  PAR3    VALUE  '('.
      03  FLD3    PICTURE X(10).
      03  VALUE3  PICTURE X(6).
      03  FILLER  PICTURE X.
01 MESSAGE.
   02 DATE-TIME.
      03  FILLER  PICTURE X(29) VALUE 'THIS OPERATION ENTERED:
          DA  -  'DATE'.
      03  DATE1   PICTURE 9(5).
      03  FILLER  PICTURE X VALUE ' '.
      03  FILLER  PICTURE X(5) VALUE 'TIME'.
      03  FILLER  PICTURE X VALUE ' '.
      03  TIME1   PICTURE 9(7).
   02 OPERATION.
      03  FILLER  PICTURE X(30) VALUE  'THE REQUESTED OPERATION
          WAS:   '.
      03  WORK-ONE    PICTURE X(83).
   02 EFFECTED.
      03  D-E PICTURE X(23) VALUE 'THE DATA EFFECTED WAS:  '.
      03  I-01    PICTURE X(46).
01 T-IO.
   02 FILLER  PICTURE X(4).
   02 TCODE   PICTURE XXX.
   02 DLMT    PICTURE X.
      88  BLNK    VALUE  ' '.
   02 DTA     PICTURE X(120).
01 OUT-IO.
   02 CNT     PICTURE S99 COMPUTATIONAL VALUE  +89.
   02 FILLER  PICTURE S99 COMPUTATIONAL VALUE  +00.
   02 DUM.
      03  FILLER PICTURE X VALUE 'N'.
      03  DATAO PICTURE X(84).
01 SAVE1  PICTURE X(4).
01 WA-1.
   02 CALL-FUNC   PICTURE X(4)    VALUE 'GU  '.
01 SSA1.
See Chapter 5, Figure 33, Ref 2.
```

```
   02 SEG1-NAME   PICTURE X(8).
   02 SSA1-QUAL   PICTURE X(18).
01 SSA2.
   02 SEG2-NAME   PICTURE X(8).
   02 SSA2-QUAL   PICTURE X(18).
01 SSA3.
   02 SEG3-NAME   PICTURE X(8).
   02 SSA3-QUAL   PICTURE X(18).
01 I-O-AREA.
See Chapter 5, Figure 33, Ref 3.
   02 KEY1    PICTURE X(6).
   02 AREA1   PICTURE X(40).
   02 AREA2   PICTURE X(34).
   02 AREA3   PICTURE X(220).
01 DISP-MESS.
   02 MESS    PICTURE X(40).
01 SW PICTURE  X  VALUE  ' '.
01 SSA-SW PICTURE  9 VALUE  0.
01 FUNC1  PICTURE XXXX.
LINKAGE SECTION.
See Chapter 5, Figure 33, Ref 4.
01 TERPCB.
   02 IN-TERM PICTURE X(8).
   02 RES     PICTURE XX.
   02 STATUS  PICTURE XX.
   02 IOPREF.
      03  DATET   PICTURE X9(7) COMPUTATIONAL-3.
      03  TIMET   PICTURE S9(7) COMPUTATIONAL-3.
      03  FILLER  PICTURE X(4).
01 PCBCASE11.
   02 DBD-NAME1   PICTURE X(8).
   02 SEG-LEVEL1  PICTURE XX.
   02 STATUS-CODES1   PICTURE XX.
   02 PROC-OPTIONS1   PICTURE X(4).
   02 DLI-JCB-ADDR1   PICTURE S9(5)   COMPUTATIONAL.
   02 SEGMENT-NAME-FEEDBACK1  PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEY1 PICTURE S9(5)   COMPUTATIONAL.
   02 NUMBER-OF-SENSITIVE-SEGS1   PICTURE S9(5) COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA1  PICTURE X(30).

01 PCBCASE21.
   02 DB-NAME3    PICTURE X(8).
   02 SEG-LEVEL3  PICTURE XX.
   02 STATUS-CODES3   PICTURE XX.
   02 PROC-OPTIONS3   PICTURE X(4).
   02 DLI-JCB-ADDR3   PICTURE S9(5)  COMPUTATIONAL.
   02 SEGMENT-NAME-FEEDBACK3  PICTURE X(8).
   02 LENGTH-OF-FEEDBACK-KEY  PICTURE S9(5) COMPUTATIONAL.
   02 NUMBER-OF-SENSITIVE-SEGS3   PICTURE S9(5) COMPUTATIONAL.
   02 KEY-FEEDBACK-AREA3  PICTURE X(30).

PROCEDURE DIVISION.
BEGIN.
   ENTER LINKAGE.
      ENTRY   'DLITCBL' USING
      See Chapter 5, Figure 33, Ref 5.
      TERPCB,
      PCBCASE11,
      PCBCASE21.
   ENTER COBOL.
IN-TERM.
   MOVE SPACES TO DTA.
   PERFORM TERM-CALL.
   MOVE 'GN  ' TO CALL-FUNC.
   IF (STATUS = 'QB') OR (STATUS = 'QC') GO TO EOJ1.
   IF STATUS = 'QD' MOVE 'GU  ' TO CALL-FUNC
   GO TO IN-TERM.
   IF STATUS NOT = '  '  MOVE SPACES TO MESS
   MOVE  'DISASTER' TO MESS PERFORM ISRT-DISP
   GO TO EOJ1.
   IF DLMT =  ' '  MOVE DTA TO WORK-AREA GO TO CHECK-FUNC.
   MOVE SPACES TO MESS.
   MOVE 'IMPROPER TRANS CODE DELIMITER' TO MESS.
   MOVE +40 TO CNT.
   PERFORM ISRT-DISP3.
      MOVE +88 TO CNT.

   MOVE 'GU  ' TO CALL-FUNC.
   GO TO IN-TERM.
CHECK-FUNC.
   IF (FUNC = 'ISRT') OR (FUNC = 'GU  ')
   OR (FUNC = 'GNP ')
   OR (FUNC = 'GN  ') OR (FUNC = 'DLET')
   OR (FUNC = 'REPL') GO TO SET-UP-SSA.  MOVE SPACES TO MESS.
   MOVE  'IMPROPER  CALL FUNCTION SPECIFIED' TO MESS.
   MOVE +45 TO CNT.
   PERFORM ISRT-DISP3.
   MOVE +88 TO CNT.
   MOVE 'GU  ' TO CALL-FUNC.
   GO TO IN-TERM.
SET-UP-SSA.
   MOVE SPACES TO SSA1, SSA2, SSA3.
   IF SEG1  NOT = '        ' MOVE SEG1 TO SEG1-NAME

   ELSE MOVE  1  TO  SSA-SW  GO TO EXIT1.
   IF PAR1 MOVE QUAL1 TO SSA1-QUAL ELSE
   MOVE SPACES TO SSA1-QUAL.
   IF SEG2  NOT = '        ' MOVE SEG2 TO SEG2-NAME
   ELSE MOVE  2  TO  SSA-SW   GO TO EXIT1
   IF PAR2 MOVE QUAL2 TO SSA2-QUAL ELSE
   MOVE SPACES TO SSA2-QUAL.
   IF SEG3  NOT = '        ' MOVE SEG3 TO SEG3-NAME
   ELSE MOVE  3  TO SSA-SW  GO TO EXIT1.
   IF PAR3 MOVE QUAL3 TO SSA3-QUAL ELSE
   MOVE SPACES TO SSA3-QUAL.
   MOVE 4 TO SSA-SW.
EXIT1.
   IF FUNC = 'ISRT'  MOVE ALL 'I' TO I-O-AREA.
   IF (FUNC = 'ISRT') AND (SSA-SW = 2) MOVE VALUE1 TO KEY1
      MOVE SPACES TO SSA1-QUAL.
   IF (FUNC = 'ISRT') AND (SSA-SW = 3)  MOVE VALUE2 TO KEY1
      MOVE SPACES TO SSA2-QUAL.
   IF (FUNC = 'ISRT') AND (SSA-SW = 4)  MOVE VALUE3 TO KEY1
      MOVE SPACES TO SSA3-QUAL.
   IF FUNC = 'REPL' GO TO GHP.
   IF FUNC = 'DLET' GO TO GHP.
   IF SSA-SW = 1  PERFORM  CALL-NO-SSA.
   IF SSA-SW = 2  PERFORM  CALL-ONE-SSA.
   IF SSA-SW = 3  PERFORM  CALL-TWO-SSA.
   IF SSA-SW = 4  PERFORM  CALL-THREE-SSA.
CK.
```

```
   IF STATUS-CODES1 = ' '  MOVE SPACES TO MESS
   MOVE 'SUCCESSFUL OPERATION' TO MESS
   ELSE MOVE SPACES  TO MESS
   MOVE 'UNSUCCESSFUL OPERATION CHECK STATUS' TO MESS
      MOVE +45 TO CNT
   PERFORM ISRT-DISP3
      MOVE SPACES TO MESS
   MOVE STATUS-CODES1 TO MESS
      MOVE +15 TO CNT
   PERFORM ISRT-DISP3
      MOVE +88 TO CNT
   GO TO IN-TERM.
   GO TO ISRT-DISP1
EOJ.
   DISPLAY 'SUCCESSFUL END OF HIMASN01'.
   ENTER LINKAGE.
      RETURN.
      See Chapter 5, Figure 33, Ref 9.
   ENTER COBOL.
GHP.
   MOVE FUNC TO FUNC1.
   MOVE 'GHU' TO FUNC.
   IF SSA-SW = 2 PERFORM CALL-ONE-SSA
   MOVE FUNC1 TO FUNC PERFORM CK-REPL
   PERFORM CALL-NO-SSA.
   IF SSA-SW = 3  PERFORM CALL-TWO-SSA
   MOVE FUNC1 TO FUNC PERFORM CK-REPL
   PERFORM CALL-NO-SSA.
   IF SSA-SW = 4 PERFORM CALL-THREE-SSA
   MOVE FUNC1 TO FUNC  PERFORM CK-REPL
   PERFORM CALL-NO-SSA.
   GO TO CK.
CK-REPL.
   IF FUNC = 'REPL' MOVE ALL 'R' TO AREA1.
CALL-NO-SSA.
   ENTER LINKAGE.
   CALL 'CBLTDLI' USING
   See Chapter 5, Figure 33, Ref 6.
      FUNC,
      PCBCASE11,
      I-O-AREA,
   ENTER COBOL.
CALL-ONE-SSA.
   ENTER LINKAGE.
   CALL 'CBLTDLI' USING
   See Chapter 5, Figure 33, Ref 7.
      FUNC,
      PCBCASE11,
      I-O-AREA,
      SSA1,
   ENTER COBOL.
CALL-TWO-SSA.
   ENTER LINKAGE.
   CALL 'CBLTDLI' USING
      FUNC,
      PCBCASE11,
      I-O-AREA,
      SSA1,
      SSA2.
   ENTER COBOL.
CALL-THREE-SSA.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING
          FUNC,
          PCBCASE11,
          I-O-AREA,
          SSA1,
          SSA2,
          SSA3.
   ENTER COBOL.
TERM-CALL.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING
      CALL-FUNC,
      TERPCB,
      T-IO.
   ENTER COBOL.
ISRT-DISP2.
   MOVE CALL-FUNC TO SAVE1.  MOVE 'ISRT' TO CALL-FUNC.
   ENTER LINKAGE.
      CALL 'CBLTDLI' USING
      See Chapter 5, Figure 33, Ref 8.
      CALL-FUNC,
      TERPCB,
      OUT-IO.
   ENTER COBOL.
   MOVE  SAVE1 TO CALL-FUNC.
ISRT-DISP3.
   MOVE SPACES TO DATAO.
   MOVE MESS TO DATAO.
   PERFORM  ISRT-DISP2.
ISRT-DISP.
   PERFORM ISRT-DISP3.
   MOVE SPACES TO DATAO.
   MOVE TERPCB TO DATAO.
   PERFORM ISRT-DISP2.
ISRT-DISP1.
   PERFORM ISRT-DISP3.
   MOVE SPACES TO DATAO.
   MOVE DATET TO DATE1.
   MOVE TIMET TO TIME1.
   MOVE DATE-TIME TO DATAO.
   PERFORM ISRT-DISP2.
      MOVE +88 TO CNT.
   MOVE SPACES TO DATAO.
   MOVE WORK-AREA TO WORK-ONE.
   MOVE OPERATION TO DATAO.
   PERFORM ISRT-DISP2.
   MOVE SPACES TO DATAO.
   MOVE I-O-AREA TO I-O1.
   MOVE EFFECTED TO DATAO.
   MOVE +74 TO CNT.
   PERFORM ISRT-DISP2.
      MOVE +88 TO CNT.
   GO TO IN-TERM.
EOJ1.
   MOVE  SPACES  TO MESS.
   MOVE  'END OF TRANS CODE (DLI)  (IMS) (ICS)'  TO MESS.
   MOVE +45 TO CNT.
   PERFORM ISRT-DISP3.
   ENTER LINKAGE.
      RETURN.
   ENTER COBOL.
```

Data Base Description (DBD)

The following DBD has a data base segment logical hierarchical relationship like Figure 45, but using hierarchical sequential organization. The DMAN control card has DD1 equal to DUMP1.

```
DBD    NAME=DS21SN01,ACCESS=SEQ
DMAN   DD1=DUMP1,DEV1=2311,DD2=DUMP1OF
SEGM   NAME=PARENT,PARENT=0,BYTES=90,FREQ=500
FLDK   NAME=KEY1,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER1,TYPE=C,BYTES=84,START=7
SEGM   NAME=LEVEL21,PARENT=PARENT,BYTES=91,FREQ=1
FLDK   NAME=KEY21,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER21,TYPE=C,BYTES=85,START=7
SEGM   NAME=LEVEL31,PARENT=LEVEL21,BYTES=259,FREQ=1
FLDK   NAME=KEY31,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER31,TYPE=C,BYTES=253,START=7
SEGM   NAME=LEVEL22,PARENT=PARENT,BYTES=91,FREQ=1
FLDK   NAME=KEY22,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER22,TYPE=C,BYTES=85,START=7
SEGM   NAME=LEVEL32,PARENT=LEVEL22,BYTES=259,FREQ=1
FLDK   NAME=KEY32,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER32,TYPE=C,BYTES=253,START=7
DBDGEN
FINISH
END
```

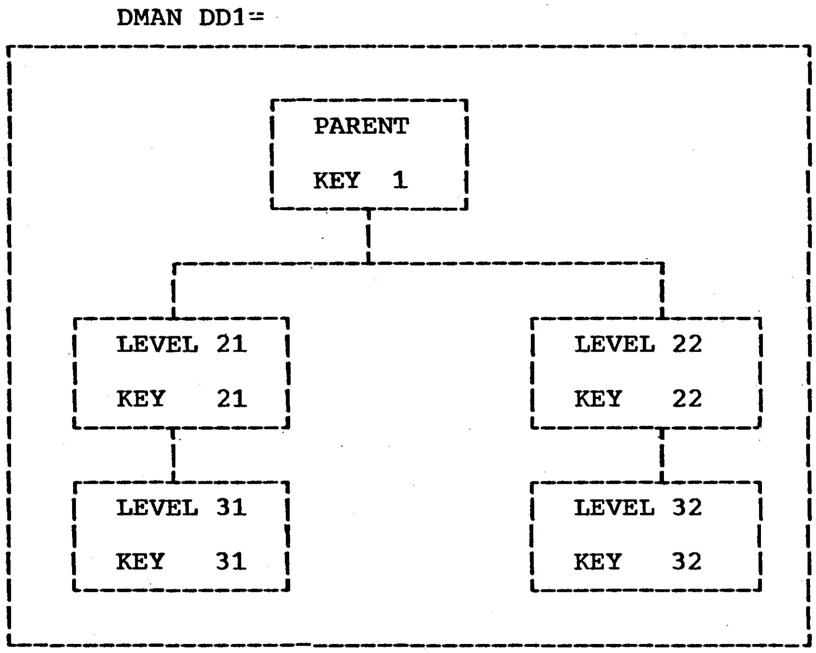


Figure 45. Single data set group data base

The following DBD has a data base segment logical hierarchical relationship like Figure 45, but uses hierarchial indexed sequential organization. The DMAN control card has DD1 equal to CASE11.

```
DBD    NAME=DI21SN01,ACCESS=INDX
DMAN   DD1=CASE11,DEV1=2311,DLIOF=CASE11OF
SEGM   NAME=PARENT,PARENT=0,BYTES=90,FREQ=500
FLDK   NAME=KEY1,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER1,TYPE=C,BYTES=84,START=7
SEGM   NAME=LEVEL21,PARENT=PARENT,BYTES=91,FREQ=1
FLDK   NAME=KEY21,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER21,TYPE=C,BYTES=85,START=7
SEGM   NAME-LEVEL31,PARENT=LEVEL21,BYTES=259,FREQ=1
FLDK   NAME=KEY31,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER31,TYPE=C,BYTES=253,START=7
SEGM   NAME=LEVEL22,PARENT=PARENT,BYTES=91,FREQ=1
FLDK   NAME=KEY22,TYPE=C,BYTES=6,START=1
```

```
FLD    NAME=FILLER22,TYPE=C,BYTES=85,START=7
SEGM   NAME=LEVEL32,PARENT=LEVEL22,BYTES=259,FREQ=1
FLDK   NAME=KEY32,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER32,TYPE=C,BYTES=253,START=7
DBDGEN
FINISH
END

DBD    NAME=DI12SN01,ACCESS=INDX
DMAN   DD1=CASE21,DEV1=2311,DLIOF=CASE21OF
SEGM   NAME=PARENT,PARENT=0,BYTES=90,FREQ=500
FLDK   NAME=KEY1,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER1,TYPE=C,BYTES=84,START=7
DMAN   DD1=DLEV21,DEV1=2311,DLIOF=LEV21OF
SEGM   NAME=LEVEL21,PARENT=PARENT,BYTES=91,FREQ=1
FLDK   NAME=KEY21,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER21,TYPE=C,BYTES=85,START=7
SEGM   NAME=LEVEL31,PARENT=LEVEL21,BYTES=259,FREQ=1
FLDK   NAME=KEY31,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER31,TYPE=C,BYTES=253,START-7
DMAN   DD1=DLEV22,DEV1=2311,DLIOF=LEV22OF
SEGM   NAME=LEVEL22,PARENT=PARENT,BYTES=91,FREQ=1
FLDK   NAME=KEY22,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER22,TYPE=C,BYTES=85,START=7
SEGM   NAME=LEVEL32,PARENT=LEVEL22,BYTES=259,FREQ=2
FLDK   NAME=KEY32,TYPE=C,BYTES=6,START=1
FLD    NAME=FILLER32,TYPE=C,BYTES=253,START=7
DBDGEN
FINISH
END
```

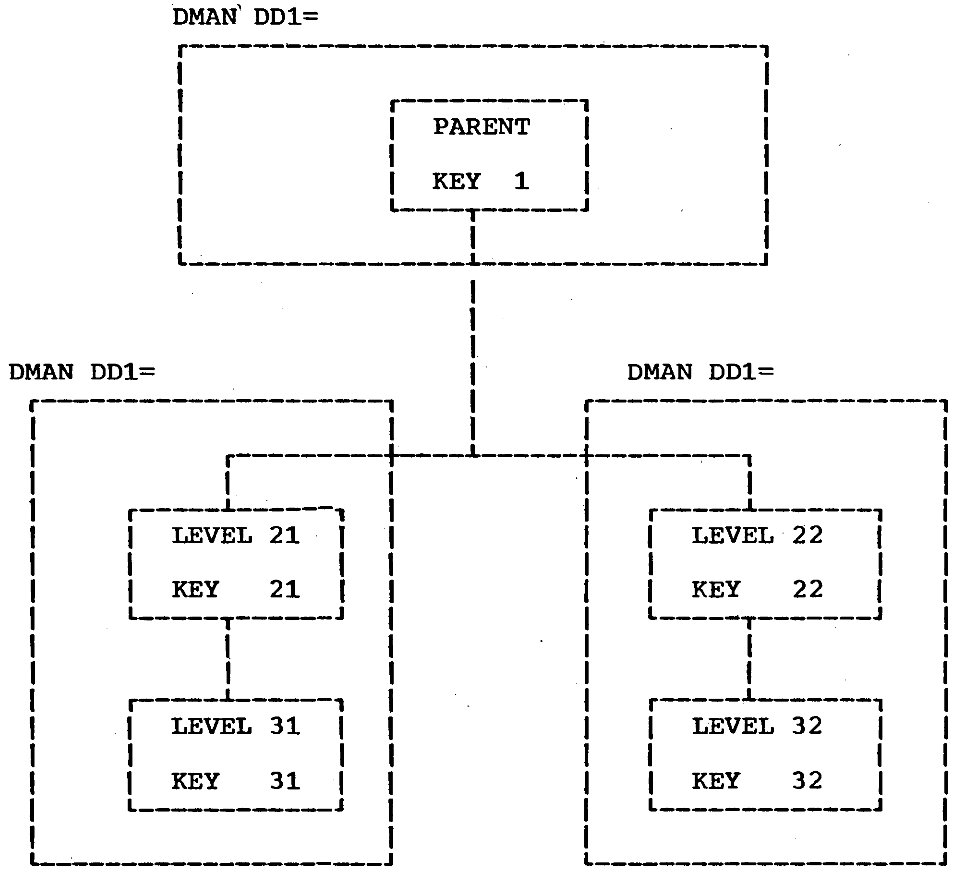


Figure 46. Multiple data set group data base

The following DBD has a data base segment logical hierarchical relationship like Figure 46, but uses hierarchial indexed sequential organization. The DMAN control card has DD1 equal to CASE21, DLEV21, and DLEV22.

PSB Generation Example

The following is one example of a PSB generation for the DBDname equal to DI21SN01. This data base name corresponds to the name of the second example in the DBD Generation example above. This is a PSB for Data Base Load (Creation) program.

```
PCB    TYPE=DB,DBDNAME=DI21SN01,PROCOPT=L,KEYLEN=30
SENSEG PARENT
SENSEG LEVEL21,PARENT
SENSEG LEVEL31,LEVEL21
SENSEG LEVEL22,PARENT
SENSEG LEVEL32,LEVEL22
PSBGEN LANG=COBOL,PSBNAME=HIBLSN01
END
```

PL/I PROGRAM EXAMPLES

PL/I Batch Program

```
          DLITPLI: PROCEDURE (PCBCASE11,PCBCASE21) OPTIONS(MAIN);

 1                     DLITPLI: PROCEDURE (PCBCASE11,PCBCASE21) OPTIONS(MAIN);
                /****************************************/
                /*                                      */
                /*   PROGRAM=HIBAJC01, PL/I             */
                /*     BATCH PROGRAM SIMILAR TO         */
                /*  HIBASN03  (SECTION  A2)             */
                /*                                      */
                /****************************************/
 2              DECLARE   FUNC             CHARACTER(4) INITIAL('    ');
 3              DECLARE   FILLER           CHARACTER(1);
 4              DECLARE   SEG1             CHARACTER(8);
 5              DECLARE   QUAL_LF1         CHARACTER(1) INITIAL('(');
 6              DECLARE   FLD1             CHARACTER(10);
 7              DECLARE   VALUE1           CHARACTER(6);
 8              DECLARE   FILLER1          CHARACTER(1);
 9              DECLARE   SEG2             CHARACTER(8);
10              DECLARE   QUAL_LF2         CHARACTER(1) INITIAL('(');
11              DECLARE   FLD2             CHARACTER(10);
12              DECLARE   VALUE2           CHARACTER(6);
13              DECLARE   FILLER2          CHARACTER(1);
14              DECLARE   SEG3             CHARACTER(8);
15              DECLARE   QUAL_LF3         CHARACTER(1) INITIAL('(');
16              DECLARE   FLD3             CHARACTER(10);
17              DECLARE   VALUE3           CHARACTER(6);
18              DECLARE   FILLER3          CHARACTER(1);
19              DECLARE   SW               CHARACTER(1) INITIAL(' ');
20              DECLARE   FUNC1            CHARACTER(4);
21              DECLARE   CARD_IN          CHARACTER(160);
22              DECLARE   SSA_SW           CHARACTER(1) INITIAL(' ');
23              DECLARE   I_O_AREA         CHARACTER(300) INITIAL(' ');
24              DECLARE   KEY1             CHARACTER(6) INITIAL(' ');
25              DECLARE   AREA1            CHARACTER(40) INITIAL(' ');
26              DECLARE   AREA2            CHARACTER(34) INITIAL(' ');
27              DECLARE   AREA3            CHARACTER(220) INITIAL(' ');
28              DECLARE     SSA            CHARACTER(8);
29              DECLARE   SSA1             CHARACTER(26);
30              DECLARE   SSA1_NAME        CHARACTER(8);
31              DECLARE   SSA1_QUAL        CHARACTER(18);
32              DECLARE   SSA2             CHARACTER(26);
33              DECLARE   SSA2_NAME        CHARACTER(8);
34              DECLARE   SSA2_QUAL        CHARACTER(18);
35              DECLARE   SSA3             CHARACTER(26);
36              DECLARE   SSA3_NAME        CHARACTER(8);
37              DECLARE   SSA3_QUAL        CHARACTER(18);
38              DECLARE   MESS             CHARACTER(40);
39                DECLARE END CHARACTER(30);
40                    DECLARE THREE FIXED BINARY(31) INITIAL (3);
41                    DECLARE FOUR FIXED BINARY(31) INITIAL (4);
42                    DECLARE FIVE FIXED BINARY(31) INITIAL (5);
43                    DECLARE SIX FIXED BINARY(31) INITIAL (6);
44                    DECLARE    SYSPRINT FILE STREAM OUTPUT;
                      /*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
                      /*  EQUIVALANT TO LINKAGE SECTION-COBOL    */
                      /*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
45              DECLARE   1 PCBCASE11,
                            2 DBD_NAME1  CHARACTER(8),
                            2 SEG_LEVEL1 CHARACTER(2),
                            2 STATUS_CODES1 CHARACTER(2),
                            2 PROC_OPTIONS1 CHARACTER(4),
                            2 DLI_JCB_ADDR1 FIXED BINARY(31,0),
                            2 SEGMENT_NAME_FEEDBACK1     CHARACTER(8),
                            2 LENGTH_OF_FEEDBACK_KEY1  FIXED BINARY(31,0),
                            2 NUMBER_OF_SENSITIVE_SEGS1 FIXED BINARY(31,0),
                            2 KEY_FEEDBACK_AREA1    CHARACTER(30);
46              DECLARE   1 PCBCASE21,
                            2 DB_NAME3        CHARACTER(8),
                            2 SEG_LEVEL3      CHARACTER(2),
                            2 STATUS_CODES3   CHARACTER(2),
                            2 PROC_OPTIONS3   CHARACTER(4),
                            2 DLI_JCB_ADDR3 FIXED BINARY(31,0),
                            2 SEGMENT_NAME_FEEDBACK3  CHARACTER(8),
                            2 LENGTH_OF_FEEDBACK_KEY3  FIXED BINARY(31,0),
                            2 NUMBER_OF_SENSITIVE_SEGS3 FIXED BINARY(31,0),
                            2 KEY_FEEDBACK_AREA3    CHARACTER(30);
47                    SW='1';
48              READ_CARDS:  GET FILE (SYSIN) EDIT (CARD_IN) (A(160));
49                          GET STRING (CARD_IN) EDIT (FUNC,FILLER,SEG1,QUAL_LF1,FLD1,
                                VALUE1,FILLER1,SEG2,QUAL_LF2,FLD2,VALUE2,FILLER2,
                                SEG3,QUAL_LF3,FLD3,VALUE3,FILLER3) (A(4),A(1),A(8),
                                A(1),A(10),A(6),A(1),A(8),A(1),A(10),A(6),A(1),A(8),
                                A(1),A(10),A(6),A(1));
                ON ENDFILE (SYSIN) GO TO EOJ;
    CHECK_FUNC: IF FUNC¬='ISRT' THEN IF FUNC¬='GU  ' THEN
                IF FUNC¬='GN  ' THEN IF FUNC¬='GNP ' THEN
                IF FUNC¬='DLET' THEN IF FUNC¬='REPL' THEN GO TO DISP;
          SET_UP_SSA: IF SEG1¬='        ' THEN GO TO A;
                ELSE SSA_SW='1';   GO TO EXIT1;
             A: IF QUAL_LF1='(' THEN GO TO B;
             B: IF SEG2¬='        ' THEN GO TO C;
                ELSE SSA_SW='2';  GO TO EXIT1;
             C: IF QUAL_LF2='(' THEN GO TO D;
             D: IF SEG3¬='        ' THEN GO TO F;
                ELSE SSA_SW='3';   GO TO EXIT1;
             E: IF QUAL_LF3='(' THEN GO TO F;
DLITPLI: PROCEDURE (PCBCASE11,PCBCASE21) OPTIONS(MAIN);


              F: SSA_SW='4';
    EXIT1:      IF FUNC='ISRT' THEN AREA1=(40)'I';
                IF FUNC='ISRT' THEN GO TO CALL_NO_SSA1;
                ELSE GO TO MORE_FUNC;
    CALL_NO_SSA1:  IF SSA_SW='2' THEN DO;
                KEY1=VALUE1;
                SSA=SEG1;
                END;
                IF SSA_SW='3' THEN DO;
                KEY1=VALUE2;
                SSA=SEG2;
                END;
                IF SSA_SW='4' THEN DO;
                KEY1=VALUE3;
                SSA=SEG3;
                END;
                I_O_AREA=KEY1||AREA1||AREA2||AREA3;
            CALL PLITDLI(FOUR,FUNC,PCBCASE11,I_O_AREA,SSA);
                GO TO CK;
    CALL_NO_SSA2:  IF SSA_SW='2' THEN DO;
                KEY1=VALUE1;
                END;
                IF SSA_SW='3' THEN DO;
                KEY1=VALUE2;
                END;
                IF SSA_SW='4' THEN DO;
                KEY1=VALUE3;
                END;
                I_O_AREA=KEY1||AREA1||AREA2||AREA3;
            CALL PLITDLI(THREE,FUNC,PCBCASE11,I_O_AREA);
                GO TO CK;
    MORE_FUNC:  IF FUNC='DLET' THEN GO TO GHP;
           IF FUNC='REPL' THEN GO TO GHP;
           IF SSA_SW='1' THEN GO TO CALL_NO_SSA;
           IF SSA_SW='2' THEN GO TO CALL_ONE_SSA;
           IF SSA_SW='3' THEN GO TO CALL_TWO_SSA;
           IF SSA_SW='4' THEN GO TO CALL_THREE_SSA;
       CK: IF STATUS_CODES1='  ' THEN MESS='SUCCESSFUL OPERATION';
           IF STATUS_CODES1='GA' THEN MESS='SUCCESSFUL OPERATION';
           IF STATUS_CODES1='GK' THEN MESS='SUCCESSFUL OPERATION';
           IF STATUS_CODES1¬='GA' THEN IF STATUS_CODES1¬='GK' THEN
           IF STATUS_CODES1¬='  ' THEN DO;
           MESS='UNSUCCESSFUL OPERATION CHECK STATUS CODE';
           END;
           GO TO RD_CK;
    RD_CK:  IF SW='1' THEN GO TO RD_DISP;
           ELSE GO TO EOJ;
         RD_DISP: PUT SKIP(2);
                  PUT EDIT (MESS) (A(40));
                    PUT SKIP(2);
                  PUT EDIT (STATUS_CODES1) (A(2));
                    PUT SKIP(2);
                  PUT EDIT (FUNC,FILLER,SEG1,QUAL_LF1,FLD1,
                      VALUE1,FILLER1,SEG2,QUAL_LF2,FLD2,VALUE2,FILLER2,
                      SEG3,QUAL_LF3,FLD3,VALUE3,FILLER3) (A(4),A(1),A(8),
                      A(1),A(10),A(6),A(1),A(8),A(1),A(10),A(6),A(1),A(8),
                      A(1),A(10),A(6),A(1));
                    PUT SKIP(2);
    PUT EDIT (KEY1,I_O_AREA) (A(6),X(2),A(300));
                    PUT SKIP(2);
      PUT FILE (SYSPRINT) EDIT
      ('DBD NAME=',DBD_NAME1,'SGMT LEVEL=',SEG_LEVEL1)
      (SKIP(1),A,A,SKIP(1),A,A);
      PUT FILE (SYSPRINT) EDIT
      ('PROC OPTIONS=',PROC_OPTIONS1,'DLI JCB ADDR=',DLI_JCB_ADDR1)
      (SKIP(1),A,A,SKIP(1),A,A);
      PUT FILE (SYSPRINT) EDIT
      ('SGMT NAME FDBK=',SEGMENT_NAME_FEEDBACK1)
      (SKIP(1),A,A);
      PUT FILE (SYSPRINT) EDIT
      ('LGTH OF FDBK=',LENGTH_OF_FEEDBACK_KEY1)
      (SKIP(1),A,A);
      PUT FILE (SYSPRINT) EDIT
      ('NBR OF SEN SGMTS=',NUMBER_OF_SENSITIVE_SEGS1)
      (SKIP(1),A,A);
      PUT FILE (SYSPRINT) EDIT
      ('KEY FDBK AREA=',KEY_FEEDBACK_AREA1)
      (SKIP(1),A,A);
             GO TO READ_CARDS;
    EOJ: END='SUCCESSFUL END OF PLIBATCH JWC';
         DISPLAY (END);
         RETURN;
    GHP: FUNC1=FUNC;
```

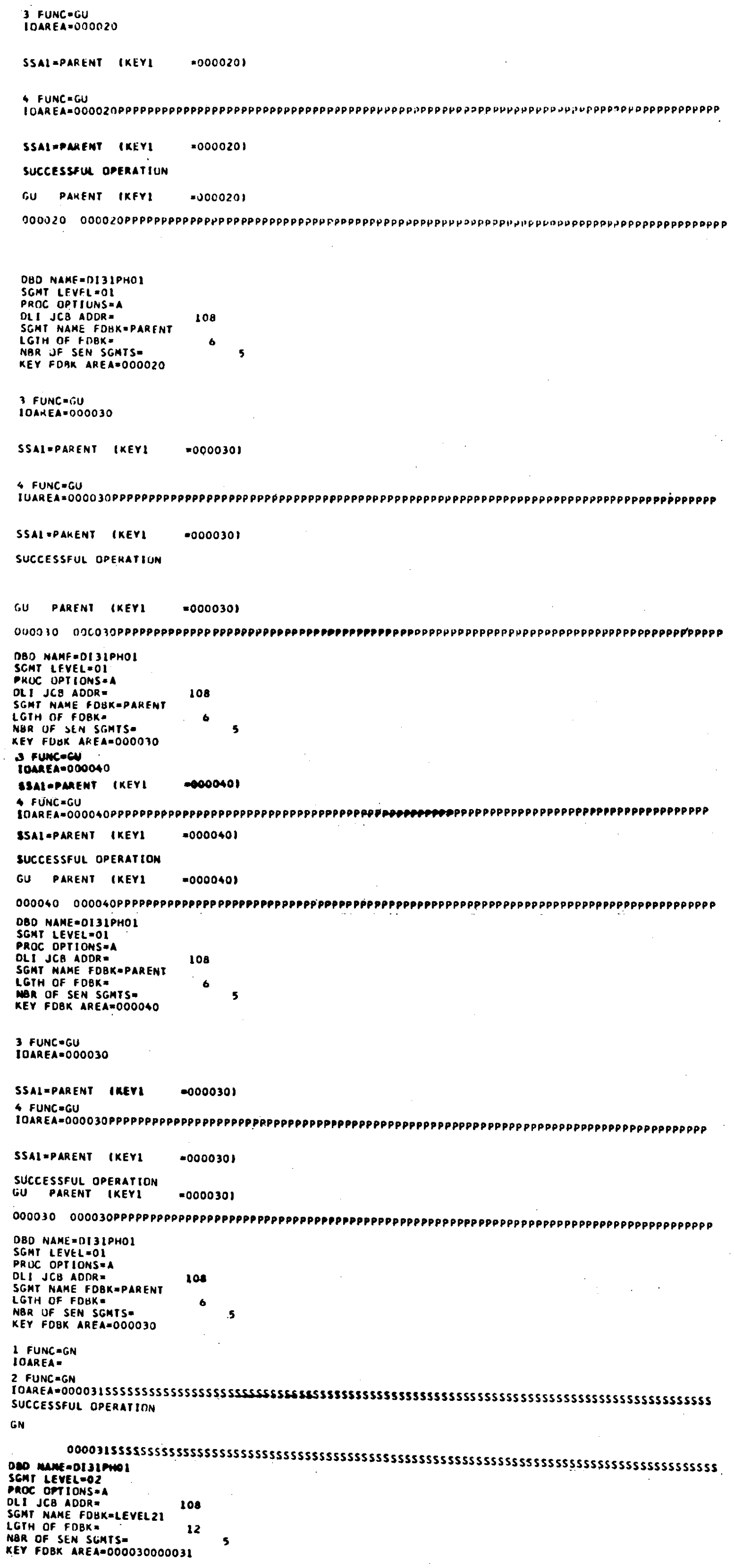
```
164           FUNC='GHU ';
165           IF SSA_SW='2' THEN GO TO CALL_ONE;
167           IF SSA_SW='3' THEN GO TO CALL_TWO;
169           IF SSA_SW='4' THEN GO TO CALL_THREE;
171      CK_REPL: FUNC=FUNC1;
172                  IF FUNC='REPL' THEN DO;
174                  AREA1=(4C)'R';
175                  GO TO CALL_NO_SSA2;
176                  END;
177                  IF FUNC='DLET' THEN DO;
179                  AREA1=(40)'D';
180                  GO TO CALL_NO_SSA2;
181                  END;
182      CALL_NO_SSA:      KEY1=VALUE1;
183                        I_O_AREA=KEY1||AREA1||AREA2||AREA3;
184                      PUT SKIP(2);
185      PUT FILE (SYSPRINT) EDIT ('1 FUNC=',FUNC,'IOAREA=',I_O_AREA)
               (SKIP(1),A,A,SKIP(1),A,A);
186                  CALL PLITDLI(THREE,FUNC,PCBCASE11,I_C_AREA);
187                      PUT SKIP(2);
188      PUT FILE (SYSPRINT) EDIT ('2 FUNC=',FUNC,'IOAREA=',I_O_AREA)
               (SKIP(1),A,A,SKIP(1),A,A);
189                  GO TO CK;
190      CALL_ONE_SSA:     KEY1=VALUE1;
191                        I_O_AREA=KEY1||AREA1||AREA2||AREA3;
192                  SSA1=SEG1||QUAL_LF1||FLD1||VALUE1||FILLER1;
193                      PUT SKIP(2);
194      PUT FILE (SYSPRINT) EDIT ('3 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
            SSA1) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
195                    CALL PLITDLI(FOUR,FUNC,PCBCASE11,I_O_AREA,SSA1);
196                      PUT SKIP(2);
197      PUT FILE (SYSPRINT) EDIT ('4 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
            SSA1) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
198                  GO TO CK;
199      CALL_ONE:         KEY1=VALUE1;
200                        I_O_AREA=KEY1||AREA1||AREA2||AREA3;
201                  SSA1=SEG1||QUAL_LF1||FLD1||VALUE1||FILLER1;
202                      PUT SKIP(2);
203      PUT FILE (SYSPRINT) EDIT ('5 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
            SSA1) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
204                CALL PLITDLI(FOUR,FUNC,PCBCASE11,I_O_AREA,SSA1);
205                      PUT SKIP(2);
206      PUT FILE (SYSPRINT) EDIT ('6 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
            SSA1) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
207                GO TO CK_REPL;
208      CALL_TWO_SSA:     KEY1=VALUE1;
209                        I_O_AREA=KEY1||AREA1||AREA2||AREA3;
210                  SSA1=SEG1||QUAL_LF1||FLD1||VALUE1||FILLER1;
211                  SSA2=SEG2||QUAL_LF2||FLD2||VALUE2||FILLER2;
212                      PUT SKIP(2);
213      PUT FILE (SYSPRINT) EDIT ('7 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA=',
            SSA1,'SSA2=',SSA2) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),SKIP(1),A,A);
214                    CALL PLITDLI(FIVE,FUNC,PCBCASE11,I_O_AREA,SSA1,SSA2);
215                      PUT SKIP(2);
216      PUT FILE (SYSPRINT) EDIT ('8 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA=',
            SSA1,'SSA2=',SSA2) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),SKIP(1),A,A);
217                  GO TO CK;
218      CALL_TWO:         KEY1=VALUE1;
219                         I_O_AREA=KEY1||AREA1||AREA2||AREA3;
220                   SSA1=SEG1||QUAL_LF1||FLD1||VALUE1||FILLER1;
221                   SSA2=SEG2||QUAL_LF2||FLD2||VALUE2||FILLER2;
222                       PUT SKIP(2);
223       PUT FILE (SYSPRINT) EDIT ('9 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA=',
             SSA1,'SSA2=',SSA2) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),SKIP(1),A,A);
224                 CALL PLITDLI(FIVE,FUNC,PCBCASE11,I_O_AREA,SSA1,SSA2);
225                       PUT SKIP(2);
226       PUT FILE (SYSPRINT) EDIT ('0 FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA=',
             SSA1,'SSA2=',SSA2) (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),SKIP(1),A,A);
227                 GO TO CK_REPL;
228       CALL_THREE_SSA:   KEY1=VALUE1;
229                         I_O_AREA=KEY1||AREA1||AREA2||AREA3;
230                   SSA1=SEG1||QUAL_LF1||FLD1||VALUE1||FILLER1;
231                   SSA2=SEG2||QUAL_LF2||FLD2||VALUE2||FILLER2;
232                   SSA3=SEG3||QUAL_LF3||FLD3||VALUE3||FILLER3;
233                       PUT SKIP(2);
234       PUT FILE (SYSPRINT) EDIT ('A FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
             SSA1,'SSA2=',SSA2,'SSA3=',SSA3)
             (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
235                       CALL PLITDLI(SIX,FUNC,PCBCASE11,I_O_AREA,SSA1,SSA2,
                    SSA3);
236                       PUT SKIP(2);
237       PUT FILE (SYSPRINT) EDIT ('B FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
             SSA1,'SSA2=',SSA2,'SSA3=',SSA3)
             (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
238                   GO TO CK;
239       CALL_THREE:                 KEY1=VALUE1;
240                         I_O_AREA=KEY1||AREA1||AREA2||AREA3;
241                   SSA1=SEG1||QUAL_LF1||FLD1||VALUE1||FILLER1;
242                   SSA2=SEG2||QUAL_LF2||FLD2||VALUE2||FILLER2;
243                   SSA3=SEG3||QUAL_LF3||FLD3||VALUE3||FILLER3;
244                       PUT SKIP(2);
245       PUT FILE (SYSPRINT) EDIT ('C FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
             SSA1,'SSA2=',SSA2,'SSA3=',SSA3)
             (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
246                   CALL PLITDLI(SIX,FUNC,PCBCASE11,I_O_AREA,SSA1,SSA2,SSA3);
247                       PUT SKIP(2);
248       PUT FILE (SYSPRINT) EDIT ('D FUNC=',FUNC,'IOAREA=',I_O_AREA,'SSA1=',
             SSA1,'SSA2=',SSA2,'SSA3=',SSA3)
             (SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A,SKIP(1),A,A);
249                 GO TO CK_REPL;
250       DISP: IF SW='1' THEN PUT EDIT ('IMPROPER CALL FUNC SPECIFIED') (A);
252             ELSE GO TO EOJ;
253             GO TO READ_CARDS;
254       END DLITPLI;
```

Data Input for PL/I Batch Program

```
GU   PARENT  (KEY1      =000020)
GU   PARENT  (KEY1      =000030)
GU   PARENT  (KEY1      =000040)
GU   PARENT  (KEY1      =000030)
GN
GN
GN
GN
GN
GU   PARENT  (KEY1      =000050)LEVEL22 (KEY22     =000052)LEVEL32
ISRT PARENT  (KEY1      =000025)
GU   PARENT  (KEY1      =000025)
REPL PARENT  (KEY1      =000025)
GU   PARENT  (KEY1      =000025)
DLET PARENT  (KEY1      =000025)
GU   PARENT  (KEY1      =000025)
```

Result of Data Input - PL/I Batch Program

This is a sample of some of the output results when PL/I batch program is executed.



```
3 FUNC=GU
IOAREA=000020

SSA1=PARENT  (KEY1     =000020)

4 FUNC=GU
IOAREA=000020PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

SSA1=PARENT  (KEY1     =000020)

SUCCESSFUL OPERATION

GU   PARENT  (KEY1     =000020)

000020  000020PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP


DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000020

3 FUNC=GU
IOAREA=000030

SSA1=PARENT  (KEY1     =000030)

4 FUNC=GU
IOAREA=000030PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

SSA1=PARENT  (KEY1     =000030)

SUCCESSFUL OPERATION


GU   PARENT  (KEY1     =000030)

000010  00C030PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

DBD NAMF=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000030
3 FUNC=GU
IOAREA=000040
SSA1=PARENT  (KEY1     =000040)
4 FUNC=GU
IOAREA=000040PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
SSA1=PARENT  (KEY1     =000040)

SUCCESSFUL OPERATION

GU   PARENT  (KEY1     =000040)

000040  000040PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000040

3 FUNC=GU
IOAREA=000030

SSA1=PARENT  (KEY1     =000030)
4 FUNC=GU
IOAREA=000030PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

SSA1=PARENT  (KEY1     =000030)

SUCCESSFUL OPERATION
GU   PARENT  (KEY1     =000030)

000030  000030PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000030

1 FUNC=GN
IOAREA=
2 FUNC=GN
IOAREA=000031SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SUCCESSFUL OPERATION

GN

        000031SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
DBD NAME=DI31PH01
SGMT LEVEL=02
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=LEVEL21
LGTH OF FDBK=           12
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000030000031
```

```
1 FUNC=GN
IOAREA=

2 FUNC=GN                                               TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
IOAREA=000026TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT

TTTTTTTTTTTTTTTTTTTTTTTTTTT

 SUCCESSFUL OPERATION
 GN

         000026TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTTTT


 DBD NAME=DI31PH01
 SGMT LEVEL=03
 PROC OPTIONS=A
 DLI JCB ADDR=            108
 SGMT NAME FDBK=LEVEL31
 LGTH OF FDBK=             18
 NBR OF SEN SGMTS=              5
 KEY FDBK AREA=000030000031000026


 1 FUNC=GN
 IOAREA=

 2 FUNC=GN
 IOAREA=000032SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 SUCCESSFUL OPERATION

 GA

 GN

        000032SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 DBD NAME=DI31PH01
 SGMT LEVEL=02
 PROC OPTIONS=A
 DLI JCB ADDR=            108
 SGMT NAME FDBK=LEVEL22
 LGTH OF FDBK=             12
 NBR OF SEN SGMTS=              5
 KEY FDBK AREA=000030000032000026
 1 FUNC=GN
 IOAREA=
 2 FUNC=GN
IOAREA=000026TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTT

 SUCCESSFUL OPERATION
 GN

         000026TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTTTT


 DBD NAME=DI31PH01
 SGMT LEVEL=03
 PROC OPTIONS=A
 DLI JCB ADDR=            108
 SGMT NAME FDBK=LEVEL32
 LGTH OF FDBK=             18
 NBR OF SEN SGMTS=              5
 KEY FDBK AREA=000030000032000026
 1 FUNC=GN
 IOAREA=

 2 FUNC=GN
 IOAREA=000040PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP

 SUCCESSFUL OPERATION

 GA

 GN

        000040PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP



 DBD NAME=DI31PH01
 SGMT LEVEL=01
 PROC OPTIONS=A
 DLI JCB ADDR=            108
 SGMT NAME FDBK=PARENT
 LGTH OF FDBK=              6
 NBR OF SEN SGMTS=              5
 KEY FDBK AREA=000040000032000026


 A FUNC=GU
 IOAREA=000050


 SSA1=PARENT  (KEY1      =000050)
 SSA2=LEVEL22 (KEY22     =000052)
 SSA3=LEVEL32


 B FUNC=GU
 IOAREA=000046TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTT
 SSA1=PARENT  (KEY1      =000050)
 SSA2=LEVEL22 (KEY22     =000052)
 SSA3=LEVEL32

 SUCCESSFUL OPERATION


 GU    PARENT  (KEY1      =000050)LEVEL22 (KEY22     =000052)LEVEL32

 000050  000046TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 TTTTTTTTTTTTTTTTTTTTTTTTTT

 DBD NAME=DI31PH01
 SGMT LEVEL=03
 PROC OPTIONS=A
 DLI JCB ADDR=            108
 SGMT NAME FDBK=LEVEL32
 LGTH OF FDBK=             18
 NBR OF SEN SGMTS=              5
 KEY FDBK AREA=000050000052000046
 SUCCESSFUL OPERATION

 ISRT PARENT  (KEY1      =000025)

 000025  000025IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
 DBD NAME=DI31PH01
 SGMT LEVEL=03
 PROC OPTIONS=A
 DLI JCB ADDR=            108
 SGMT NAME FDBK=LEVEL32
 LGTH OF FDBK=             18
 NBR OF SEN SGMTS=              5
 KEY FDBK AREA=000050000052000046
```

```
3 FUNC=GU
IOAREA=000025IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII


SSA1=PARENT  (KEY1     =000025)


4 FUNC=GU
IOAREA=000025IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII


SSA1=PARENT  (KEY1     =000025)

SUCCESSFUL OPERATION


GU   PARENT  (KEY1     =000025)

000025  000025IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII

DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000025000052000046

5 FUNC=GHU
IOAREA=000025IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII


SSA1=PARENT  (KEY1     =000025)


6 FUNC=GHU
IOAREA=000025IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII


SSA1=PARENT  (KEY1     =000025)

SUCCESSFUL OPERATION

REPL PARENT  (KEY1     =000025)

000025  000025RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT

LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000025000052000046
3 FUNC=GU
IOAREA=000025RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR


SSA1=PARENT  (KEY1     =000025)


4 FUNC=GU
IOAREA=000025RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR


SSA1=PARENT  (KEY1     =000025)

SUCCESSFUL OPERATION


GU   PARENT  (KEY1     =000025)

000025  000025RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR



DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000025000052000046

5 FUNC=GHU
IOAREA=000025RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR


SSA1=PARENT  (KEY1     =000025)


6 FUNC=GHU
IOAREA=000025RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR


SSA1=PARENT  (KEY1     =000025)

SUCCESSFUL OPERATION


DLET PARENT  (KEY1     =000025)

000025  000025DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD

DBD NAME=DI31PH01
SGMT LEVEL=01
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000025000052000046
3 FUNC=GU
IOAREA=000025DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD


SSA1=PARENT  (KEY1     =000025)


4 FUNC=GU
IOAREA=000025DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD


SSA1=PARENT  (KEY1     =000025)

UNSUCCESSFUL OPERATION CHECK STATUS CODE

GE

GU   PARENT  (KEY1     =000025)

000025  000025DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD



DBD NAME=DI31PH01
SGMT LEVEL=
PROC OPTIONS=A
DLI JCB ADDR=          108
SGMT NAME FDBK=PARENT
LGTH OF FDBK=            6
NBR OF SEN SGMTS=            5
KEY FDBK AREA=000025000052000046
```

PL/I Message Program Example

The following is an example for a PL/I message program which accesses the same data base accessed by the batch PL/I program example and COBOL programs. This program can be executed on either the 2740 terminal or the 2260 Display Station.

```
      DLITPLI: PROCEDURE(TERMINAL,MASTER_TERM,PCBCASE11) OPTIONS(MAIN);


 1              DLITPLI: PROCEDURE(TERMINAL,MASTER_TERM,PCBCASE11) OPTIONS(MAIN);
                /********************************************************************/
                /*  DECLARING PCB'S -- 1 - INPUT PCB, 2 - OUTPUT PCB, 3 - DB PCB    */
                /********************************************************************/
 2              DECLARE 1 TERMINAL,
                          2 NAME        CHARACTER(8),
                          2 FILLER      BIT(16),
                          2 STAT_CODES    CHAR(2),
                          2 PREFIX,
                             3 DATE     FIXED DECIMAL(7,0),
                             3 TIME     FIXED DECIMAL(7,0),
                             3 MSG_NUMBER   FIXED BINARY(31,0);
 3              DCL 1 MASTER_TERM,
                    2 MSNAME       CHAR(8),
                    2 MSFILL       BIT(16),
                    2 MSSTAT       CHAR(2);
 4              DECLARE   1 PCBCASE11,
                             2 DBD_NAME1  CHARACTER(8),
                             2 SEG_LEVEL1 CHARACTER(2),
                             2 STATUS_CODES1 CHARACTER(2),
                             2 PROC_OPTIONS1 CHARACTER(4),
                             2 DLI_JCB_ADDR1 FIXED BINARY(31,0),
                             2 SEGMENT_NAME_FEEDBACK1      CHARACTER(8),
                             2 LENGTH_OF_FEEDBACK_KEY1  FIXED BINARY(31,0),
                             2 NUMBER_OF_SENSITIVE_SEGS1 FIXED BINARY(31,0),
                             2 KEY_FEEDBACK_AREA1     CHARACTER(30);
                /********************************************************************/
                /*                    VARIABLES                                     */
                /********************************************************************/
 5              DCL   GU STATIC     CHAR(4) INITIAL('GU  ');
 6              DCL   GN STATIC     CHAR(4) INITIAL('GN  ');
 7              DCL   ISRT STATIC  CHAR(4) INITIAL('ISRT');
 8              DECLARE   FUNC STATIC      CHARACTER(4) INITIAL('    ');
 9              DECLARE   FUNC1 STATIC     CHARACTER(4) INITIAL(' ');
10              DECLARE   SSA_SW STATIC    CHARACTER(1) INITIAL(' ');
11              DCL I_O_AREA               CHARACTER(300) INITIAL(' ');
12              DECLARE   KEY1 STATIC      CHARACTER(6) INITIAL(' ');
13              DECLARE   SSA  STATIC      CHARACTER(9) INITIAL(' ');
14              DECLARE   SSA1  STATIC     CHARACTER(26) INITIAL(' ');
15              DECLARE   SSA2  STATIC     CHARACTER(26) INITIAL(' ');
16              DECLARE   SSA3  STATIC     CHARACTER(26) INITIAL(' ');
17              DECLARE   MESS  STATIC     CHARACTER(22) INITIAL(' ');
18              DCL (L,S) STATIC  FIXED BINARY(31,0) INITIAL(0);
19              DCL  OUTMSGCODE     CHAR(2) INITIAL(' ');
20              DCL  STRING(1:4)  STATIC    CHAR(48);
21              DCL  Q(1:4)  STATIC         FIXED BINARY;
22              DCL  (SGMT_NO,M,I)  STATIC   FIXED DECIMAL(5);
23              DCL  SEG(1:4)  STATIC       CHAR(8);
24              DCL  QUAL_LF(1:4)  STATIC   CHAR(1);
25              DCL  FLD(1:4)  STATIC       CHAR(8);
26              DCL  RO(1:4)  STATIC        CHAR(2);
27              DCL  VALUE(1:4)  STATIC     CHAR(6);
28              DCL  QUAL_RF(1:4)  STATIC   CHAR(1);
29              DCL THREE STATIC FIXED BINARY(31) INITIAL (3);
30              DCL FOUR  STATIC FIXED BINARY(31) INITIAL (4);
31              DCL FIVE  STATIC FIXED BINARY(31) INITIAL (5);
32              DCL SIX   STATIC FIXED BINARY(31) INITIAL (6);
                /********************************************************************/
                /*           INPUT/OUTPUT STATIC STRUCTURES                         */
                /********************************************************************/
33              DCL 1 INPUT_MSG STATIC,
                     2  LL     FIXED BINARY(31,0) INITIAL(0),
                     2  Z7     BIT(16) INITIAL((16)'0'B),
                     2  IN_TEXT    CHAR(80) INITIAL(' ');
34              DCL 1 OUTPUT_MSG STATIC,
                     2  LL1     FIXED BINARY(31,0),
                     2  Z3     BIT(8) INITIAL((8)'0'B),
                     2  Z4     BIT(8),
                     2  TEXT_OUT       CHAR(80) INITIAL(' ');
35              DCL 1 OUTPUT_ANS STATIC,
                     2  LL2      FIXED BINARY(31,0),
                     2  Z5      BIT(8) INITIAL((8)'0'B),
                     2  Z6      BIT(8),
                     2  TEXT_OUT,
                       3  FIRST CHAR(15) INITIAL('CALL WAS: FUNC='),
                       3  CALL_FUNC CHAR(4) INITIAL(' '),
                       3  SEC CHAR(7) INITIAL('  SSA1='),
                       3  SSA_DATA1  CHAR(27) INITIAL(' ');
36              DCL 1 OUTPUT_ANSC STATIC,
                     2  LL6      FIXED BINARY(31,0),
                     2  Z15     BIT(8) INITIAL((8)'0'B),
                     2  Z16     BIT(8),
                     2  TEXT_OUT,
                       3  FIRST CHAR(15) INITIAL('CALL WAS: FUNC='),
                       3  CALL_FUNC1 CHAR(5) INITIAL(' ');
37              DCL 1 OUTPUT_ANS1 STATIC,
                     2  LL3     FIXED BINARY(31,0),
                     2  Z7      BIT(8) INITIAL((8)'0'B),
                     2  Z8      BIT(8),
                     2  TEXT_OUT,
                3  THIRD CHAR(26) INITIAL('                          SSA2='),
                3  SSA_DATA2  CHAR(27) INITIAL(' ');
38         DCL 1 OUTPUT_ANS2 STATIC,
              2  LL4     FIXED BINARY(31,0),
              2  Z9      BIT(8) INITIAL((8)'0'B),
              2  Z10     BIT(8),
              2  TEXT_OUT,
                3  FOURTH CHAR(26) INITIAL('                         SSA3='),
                3  SSA_DATA3    CHAR(27) INITIAL(' ');
39         DCL 1  OUTPUT_ANS3 STATIC,
              2  LL5 FIXED BINARY(31,0),
              2  Z11     BIT(8) INITIAL((8)'0'B),
              2  Z12     BIT(8),
              2  OUT_TEXT      CHAR(320) INITIAL(' ');
40         DCL 1 ERR STATIC,
              2  C4     FIXED BINARY(31,0),
              2  Z13     BIT(8) INITIAL((8)'0'B),
              2  Z14     BIT(8),
              2  STAT    CHAR(12) INITIAL('STATUS CODE='),
              2  STAT_CODE     CHAR(2) INITIAL(' '),
              2  CR      CHAR(1) INITIAL('▲');          11,5,9 multipunch
41         DCL 1 CALL_FUNC_ERR  STATIC,
             2  C5  FIXED BINARY(31,0),
             2  Z17     BIT(8) INITIAL((8)'0'B),
             2  Z18  BIT(8),
             2  MESSAGE      CHAR(17) INITIAL('INVALID CALL FUNC='),
             2  FUNCTION     CHAR(5) INITIAL(' ');
           /*****************************************************************/
           /*                    FIXED MESSAGES                               */
           /*****************************************************************/
42         DCL MSG0 CHAR(49) STATIC
           INITIAL('      STATE THE FOLLOWING FOR OBTAINING DATA FROM ');
43         DCL MSG1 STATIC CHAR(21) INITIAL('DATA BASE DI31PH01 BY');
44         DCL MSG1A STATIC CHAR(35)
             INITIAL('      FILLING IN THE UNDERLINES AND ');
45         DCL MSG2 STATIC CHAR(25) INITIAL('ADDING A NEW LINE SYMBOL.');
46         DCL MSG3 STATIC CHAR(51)
             INITIAL(' TRAN  FUNC SGMT-NAME SGMT-FLD-NAME R/O  COMP-VALUE');
47         DCL MSG4 STATIC CHAR(48)
             INITIAL('        ____ ________ (________       __    ______)');
48         DCL MSG5 STATIC CHAR(48)
             INITIAL('             ________ (________       __    ______)');
49         DCL MSGT STATIC CHAR(4) INITIAL('TUBE');
50         DCL MSGA STATIC CHAR(53)
            INITIAL('      ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE ');
51         DCL MSGA1 STATIC CHAR(9) INITIAL('DI31PH01.');
           /*****************************************************************/
           /*                 WRITE INITIAL =  WI                             */
           /*****************************************************************/
52         DCL  WI STATIC               BIT(8) INITIAL((8)'0'B);
           /*****************************************************************/

           /*     WRITE AT LINE ADDRESS =  WALAN , N= LINE NUMBER             */
           /*****************************************************************/
53         DCL  WALA1 STATIC            BIT(8) INITIAL('00000001'B);
54         DCL  WALA2 STATIC            BIT(8) INITIAL('00000010'B);
55         DCL  WALA3 STATIC            BIT(8) INITIAL('00000011'B);
56         DCL  WALA4 STATIC            BIT(8) INITIAL('00000100'B);
57         DCL  WALA5 STATIC            BIT(8) INITIAL('00000101'B);
58         DCL  WALA6 STATIC            BIT(8) INITIAL('00000110'B);
59         DCL  WALA7 STATIC            BIT(8) INITIAL('00000111'B);
60         DCL  WALA8 STATIC            BIT(8) INITIAL('00001000'B);
61         DCL  WALA9 STATIC            BIT(8) INITIAL('00001001'B);
62         DCL  WALA10 STATIC           BIT(8) INITIAL('00001010'B);
63         DCL  WALA11 STATIC           BIT(8) INITIAL('00001011'B);
64         DCL  WALA12 STATIC           BIT(8) INITIAL('00001100'B);
           /****************************************************************/
           /*  ERASE SCREEN, START AT LINE ADDRESS = ESSLAN, N= LINE NUMBER  */
           /****************************************************************/
65         DCL  ESSLA1 STATIC           BIT(8) INITIAL('00010001'B);
66         DCL  ESSLA2 STATIC           BIT(8) INITIAL('00010010'B);
67         DCL  ESSLA3 STATIC           BIT(8) INITIAL('00010011'B);
68         DCL  ESSLA4 STATIC           BIT(8) INITIAL('00010100'B);
69         DCL  ESSLA5 STATIC           BIT(8) INITIAL('00010101'B);
70         DCL  ESSLA6 STATIC           BIT(8) INITIAL('00010110'B);
71         DCL  ESSLA7 STATIC           BIT(8) INITIAL('00010111'B);
72         DCL  ESSLA8 STATIC           BIT(8) INITIAL('00011001'B);
73         DCL  ESSLA9 STATIC           BIT(8) INITIAL('00011001'B);
74         DCL  ESSLA10  STATIC         BIT(8) INITIAL('00011010'B);
75         DCL  ESSLA11 STATIC          BIT(8) INITIAL('00011011'B);
76         DCL  ESSLA12 STATIC          BIT(8) INITIAL('00011100'B);
           /****************************************************************/
           /*                   WRITE ERASE  =  WE                           */
           /****************************************************************/
77         DCL  WE     STATIC           BIT(8) INITIAL('00100000'B);
           /****************************************************************/
           /*                   START MANUAL INPUT SYMBOL                    */
           /****************************************************************/
78         DCL START_MI STATIC CHAR(1) INITIAL('▲');        12,2,8 multipunch
           /****************************************************************/
           /*         NEW LINE SYMBOL (NL),  SAME AS (CR)                    */
           /****************************************************************/
79         DCL NL STATIC CHAR(1) INITIAL('▲');              11,5,9 multipunch
80         DCL  (LGTH_KEY,NBR_SENSGMT,DLIJCB)   CHAR(14);
           /****************************************************************/
           /*                                                                */
           /*                       PROGRAM                                  */
           /*                                                                */
           /*  CAN BE EXECUTED ON A 2740 TERMINAL OR A 2260 DISPLAY STATION  */
/*             OUTPUT FURM TO 2260 TUBE OR 2740 TERMINAL                    */
/***************************************************************************/
B:
STRING=' ';
Q=0;
SGMT_NO=0;  M=0;  I=0;
SEG=' ';  QUAL_LF=' ';  FLD=' ';  RO=' ';  VALUE=' ';  QUAL_RF=' ';
   CALL PLITDLI(THREE,GU,TERMINAL,INPUT_MSG);
   IF STAT_CODES='QD' THEN GO TO FINAL;
   ELSE IF STAT_CODES¬='  ' THEN GO TO FINAL;
   ELSE IF STAT_CODES='  ' THEN GO TO A;
A:  I= INDEX(IN_TEXT,' ');
  IF I=LL - 4 THEN GO TO E;
  FUNC= SUBSTR(IN_TEXT,7,4);
  IF (FUNC='GN  ')|(FUNC='GU  ')|(FUNC='ISRT')|(FUNC='REPL')
     |(FUNC='DLET')|(FUNC='GNP ')|(FUNC='GHU ')|(FUNC='GHN ')
     |(FUNC='GHNP') THEN GO TO C;
  ELSE GO TO INCORR;
E:  LL1=75;
   Z4=WE;
   OUTPUT_MSG.TEXT_OUT=MSG0||MSG1||NL;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
   LL1=65;
   Z4=WALA2;
   OUTPUT_MSG.TEXT_OUT=MSG1A||MSG2||NL;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
   LL1=56;
   Z4=WALA3;
   OUTPUT_MSG.TEXT_OUT=MSG3||NL;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
LL1=53;
   Z4=WALA4;
   OUTPUT_MSG.TEXT_OUT=MSG4||NL;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
   LL1=53;
   Z4=WALA5;
   OUTPUT_MSG.TEXT_OUT=MSG5||NL;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
  LL1=53;
   Z4=WALA6;
   OUTPUT_MSG.TEXT_OUT=MSG5||NL;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
   LL1=9;
   Z4=WALA4;
   OUTPUT_MSG.TEXT_OUT=START_MI||MSGI;
   CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
   GO TO B;
/***************************************************************************/
/*                GET DATA INPUT FROM DISPLAY ON TUBE OR 2740               */
/*          AND EDIT FOR DATA BASE ACCESS                                   */
/***************************************************************************/
C:  DO M=1 BY 1 TO 3;

   Q(M)=LL - 4;
   STRING(M)= SUBSTR(IN_TEXT,1,LL - 4);
   CALL PLITDLI(THREE,GN,TERMINAL,INPUT_MSG);
IF (STAT_CODES='QD')|(STAT_CODES='QC')|(STAT_CODES¬='  ')
   THEN GO TO F;
END C;
F:  STRING(1)=' '||STRING(1);
Q(1)=Q(1) + 1;
  IF Q(1)>11 THEN GO TO AB;
   ELSE DO;
     SSA_SW='1';
   IF (FUNC¬='ISRT')
    |(FUNC¬='REPL')|(FUNC¬='DLET') THEN GO TO CALL_NO_SSA;
     GO TO EXIT1;
   END;
AB:  SEG(1)= SUBSTR(STRING(1),13,8);
   IF Q(1)>20  THEN GO TO AC;
   ELSE DO;
   SSA_SW='2';
   SSA1=SEG(1);
   IF (FUNC¬='ISRT')|(FUNC¬='DLET')|(FUNC¬='REPL')
     THEN GO TO CALL_ONE_SSA;
   GO TO EXIT1;     END;
AC:  SSA_SW='2';
   QUAL_LF(1)= SUBSTR(STRING(1),22,1);
   FLD(1)= SUBSTR(STRING(1),23,8);
   RO(1)= SUBSTR(STRING(1),37,2);
   VALUE(1)= SUBSTR(STRING(1),42,6);
   QUAL_RF(1)= SUBSTR(STRING(1),48,1);
SSA1=SEG(1)||QUAL_LF(1)||FLD(1)||RO(1)||VALUE(1)||QUAL_RF(1);
IF Q(2)>0 THEN GO TO AD;

IF (Q(2)=0)&((FUNC='ISRT')|(FUNC='DLET')|(FUNC='REPL')) THEN
GO TO EXIT1;
ELSE GO TO CALL_ONE_SSA;
AD:  SEG(2)= SUBSTR(STRING(2),13,8);
  IF Q(2)>20 THEN GO TO AE;
  ELSE DO;
    SSA_SW='3';
SSA1=SEG(1)||QUAL_LF(1)||FLD(1)||RO(1)||VALUE(1)||QUAL_RF(1);
SSA2=SEG(2);
```

```
    IF (FUNC¬='ISRT')|(FUNC¬='DLET')|(FUNC¬='REPL')
      THEN GO TO CALL_TWO_SSA;
    GO TO EXIT1;     END;
AE:  SSA_SW='3';
    QUAL_LF(2)= SUBSTR(STRING(2),22,1);
    FLD(2)= SUBSTR(STRING(2),23,9);
    RO(2)= SUBSTR(STRING(2),37,2);
    VALUE(2)= SUBSTR(STRING(2),42,6);
    QUAL_RF(2)= SUBSTR(STRING(2),48,1);
SSA1=SEG(1)||QUAL_LF(1)||FLD(1)||RO(1)||VALUE(1)||QUAL_RF(1);
SSA2=SEG(2)||QUAL_LF(2)||FLD(2)||RO(2)||VALUE(2)||QUAL_RF(2);
  IF Q(3)>0 THEN GO TO AF;

     IF (Q(3)=0)&((FUNC='ISRT')|(FUNC='DLET')|(FUNC='REPL')) THEN
     GO TO EXIT1:
  ELSE GO TO CALL_TWO_SSA;
AF:  SEG(3)= SUBSTR(STRING(3),13,8);
    IF Q(3)>20 THEN GO TO AG;
    ELSE DO;
  SSA_SW='4';
SSA1=SEG(1)||QUAL_LF(1)||FLD(1)||RO(1)||VALUE(1)||QUAL_RF(1);
SSA2=SEG(2)||QUAL_LF(2)||FLD(2)||RO(2)||VALUE(2)||QUAL_RF(2);
  SSA3=SEG(3);
    IF (FUNC¬='ISRT')|(FUNC¬='DLET')|(FUNC¬='REPL')
    THEN GO TO CALL_THREE_SSA;
    GO TO EXIT1;     END;
AG:  SSA_SW='4';
    QUAL_LF(3)= SUBSTR(STRING(3),22,1);
    FLD(3)= SUBSTR(STRING(3),23,8);
    RO(3)= SUBSTR(STRING(3),37,2);
    VALUE(3)= SUBSTR(STRING(3),42,6);
    QUAL_RF(3)= SUBSTR(STRING(3),48,1);
SSA1=SEG(1)||QUAL_LF(1)||FLD(1)||RO(1)||VALUE(1)||QUAL_RF(1);
SSA2=SEG(2)||QUAL_LF(2)||FLD(2)||RO(2)||VALUE(2)||QUAL_RF(2);
SSA3=SEG(3)||QUAL_LF(3)||FLD(3)||RO(3)||VALUE(3)||QUAL_RF(3);
    IF (FUNC¬='ISRT')|(FUNC¬='DLET')|(FUNC¬='REPL')
    THEN GO TO CALL_THREE_SSA;
    ELSE GO TO EXIT1;
/*****************************************************************/
/*     FOR FUNC IF  ISRT,  DLET,  REPL                           */
/*****************************************************************/
EXIT1:   IF FUNC='ISRT' THEN I_O_AREA=(40)'I';
    IF FUNC='ISRT' THEN GO TO CALL_NO_SSA1;
    ELSE GO TO MORE_FUNC;
CALL_NO_SSA1:  IF SSA_SW='2' THEN DO;
            SSA=SEG(1);
            KEY1=VALUE(1);
    QUAL_LF(1)=' ';
    FLD(1)=' ';
    RO(1)=' ';
    VALUE(1)=' ';
    QUAL_RF(1)=' ';
            END;
            IF SSA_SW='3' THEN DO;
            SSA=SEG(2);
            KEY1=VALUE(2);
    QUAL_LF(2)=' ';
    FLD(2)=' ';
    RO(2)=' ';
    VALUE(2)=' ';
    QUAL_RF(2)=' ';
            END;
            IF SSA_SW='4' THEN DO;
            SSA=SEG(3);
            KEY1=VALUE(3);
    QUAL_LF(3)=' ';
    FLD(3)=' ';
    RO(3)=' ';
    VALUE(3)=' ';
    QUAL_RF(3)=' ';
            END;
    I_O_AREA=KEY1||I_O_AREA:
        CALL PLITDLI(FOUR,FUNC,PCBCASE11,I_O_AREA,SSA);
            GO TO CK;
/*****************************************************************/
/*              FOR FUNC IF DLET,  REPL                          */
/*****************************************************************/
MORE_FUNC:  IF FUNC='DLET' THEN GO TO GHP;
       IF FUNC='REPL' THEN GO TO GHP;
/*****************************************************************/
/*   CALLS TO DATABASE DEPENDING ON NUMBER OF SSA'S              */
/*****************************************************************/
CALL_NO_SSA:
    KEY1=VALUE(1);
            CALL PLITDLI(THREE,FUNC,PCBCASE11,I_O_AREA);
            GO TO CK;
CALL_ONE_SSA:
    KEY1=VALUE(1);
              CALL PLITDLI(FOUR,FUNC,PCBCASE11,I_O_AREA,SSA1);
        IF FUNC='GHU ' THEN GO TO CK_REPL;
        ELSE GO TO CK;
CALL_TWO_SSA:
    KEY1=VALUE(1);
              CALL PLITDLI(FIVE,FUNC,PCBCASE11,I_O_AREA,SSA1,SSA2);
        IF FUNC='GHU ' THEN GO TO CK_REPL;
        ELSE GO TO CK;
CALL_THREE_SSA:
    KEY1=VALUE(1);
               CALL PLITDLI(SIX,FUNC,PCBCASE11,I_O_AREA,SSA1,SSA2,
          SSA3);
        IF FUNC='GHU ' THEN GO TO CK_REPL;
        ELSE GO TO CK;
/*********************************************************************/
/* DO A  GHU  FIRST FOR A DLET & REPL CALL                           */
/*********************************************************************/
GHP: FUNC1=FUNC;
     FUNC='GHU ';
    IF SSA_SW='2' THEN GO TO CALL_ONE_SSA;
    IF SSA_SW='3' THEN GO TO CALL_TWO_SSA;
    IF SSA_SW='4' THEN GO TO CALL_THREE_SSA;
CK_REPL: FUNC=FUNC1;
            IF FUNC='REPL' THEN DO;
         I_O_AREA=(40)'R';
            GO TO CALL_NO_SSA2;
            END;
            IF FUNC='DLET' THEN DO;
         I_O_AREA=(40)'D';
            GO TO CALL_NO_SSA2;
            END;
CALL_NO_SSA2:  IF SSA_SW='2' THEN DO;
            KEY1=VALUE(1);
    QUAL_LF(1)=' ';
    FLD(1)=' ';
    RO(1)=' ';
    VALUE(1)=' ';
    QUAL_RF(1)=' ';
            END;
            IF SSA_SW='3' THEN DO;
            KEY1=VALUE(2);
    QUAL_LF(2)=' ';
    FLD(2)=' ';
    RO(2)=' ';
    VALUE(2)=' ';
    QUAL_RF(2)=' ';
            END;
            IF SSA_SW='4' THEN DO;
            KEY1=VALUE(3);
    QUAL_LF(3)=' ';
    FLD(3)=' ';
    RO(3)=' ';
    VALUE(3)=' ';
    QUAL_RF(3)=' ';
            END;
      I_O_AREA=KEY1||I_O_AREA;
        CALL PLITDLI(THREE,FUNC,PCBCASE11,I_O_AREA);
            GO TO CK;
 /*********************************************************************/
 /*   PUT DATABASE CALL BACK TO 2740 TERMINAL OR 2260 TUBE            */
 /*********************************************************************/
 CK: IF (STATUS_CODES1='  ')|(STATUS_CODES1='GA')
     |(STATUS_CODES1='GK') THEN
     MESS='SUCCESSFUL   OPERATION';
  ELSE MESS='UNSUCCESSFUL OPERATION';
 LL1=67;
 Z4=WE;
 OUTPUT_MSG.TEXT_OUT=MSGA||MSGA1;
 SUBSTR(OUTPUT_MSG.TEXT_OUT,LL1 - 4,1)=NL;
 CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
 IF SSA_SW='1' THEN DO;
 LL6=23;
 Z16=WALA2;
 CALL_FUNC1=FUNC;
 CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANSO);
  GO TO EE;
 END;
 IF SSA_SW='2' THEN DO;
 LL2= LENGTH(SSA1) + 31;
 Z6=WALA2;
 CALL_FUNC=FUNC;
 SSA_DATA1=SSA1||NL;
 CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS);
  GO TO EE;
 END;
 IF SSA_SW='3' THEN DO;
 LL2=57;
 Z6=WALA2;
 CALL_FUNC=FUNC;
 SSA_DATA1=SSA1||NL;
 CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS);
 LL3= LENGTH(SSA2) + 31;
 Z8=WALA3;
 SSA_DATA2=SSA2||NL;
 CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS1);
  GO TO EE;
 END;
 IF SSA_SW='4' THEN DO;
 LL2=57;
 Z6=WALA2;
 CALL_FUNC=FUNC;
 SSA_DATA1=SSA1||NL;
 CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS);
 LL3=57;
 Z8=WALA3;
 SSA_DATA2=SSA2||NL;
```

```
CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS1);
LL4= LENGTH(SSA3) + 31;
Z10=WALA4;
SSA_DATA3=SSA3||NL;
CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS2);
 GO TO EE;
END;
EE:    LL1=79;
  IF STAT_CODES='  ' THEN OUTMSGCODE='XX';
    ELSE OUTMSGCODE=STAT_CODES;
Z4=WALA5;
OUTPUT_MSG.TEXT_OUT='RETURNED DATA:  '||MESS||', FUNC='||FUNC
  ||', STATUS CODE='||OUTMSGCODE||','||' IOAREA='||NL;
CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
IF MESS='UNSUCCESSFUL OPERATION' THEN GO TO DD;
  L= LENGTH(I_O_AREA);
  Z12=WALA6;
  S=1;
     IF L>80 THEN DO;
Y1:   OUT_TEXT= SUBSTR(I_O_AREA,S,80);
  SUBSTR(OUT_TEXT,80,1)=NL;
  LL5=84;
  CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS3);
  Z12=WI;
  L=L - 79;
  S=S + 79;
    IF L>80 THEN GO TO Y1;
  END;
  IF L=0 THEN GO TO X1;
LL5=L + 5;
OUT_TEXT= SUBSTR(I_O_AREA,S,L);
SUBSTR(OUT_TEXT,L + 1,1)=NL;
  CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_ANS3);
X1:  I_O_AREA=' ';
/**********************************************************************/
/*  DISPLAY  DATA BASE PCB'S                                          */
/**********************************************************************/
DD:   LL1=72;
Z4=WALA10;
OUTPUT_MSG.TEXT_OUT=' DBD NAME='||DBD_NAME1||', SGMT LEVEL='
  ||SEG_LEVEL1||', DB STAT CODES='||STATUS_CODES1||', PROC OPT='
  ||PROC_OPTIONS1||','||NL;
CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
LL1=83;
Z4=WALA11;
DLIJCB=DLI_JCB_ADDR1;
LGTH_KEY=LENGTH_OF_FEEDBACK_KEY1;
OUTPUT_MSG.TEXT_OUT=' DLIJCB ADDR='||DLIJCB||', SGMT NAME FDBK='
  ||SEGMENT_NAME_FEEDBACK1||',LGTH FDBK='||LGTH_KEY||','||NL;
CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
LL1=79;
Z4=WALA12;
NBR_SENSGMT=NUMBER_OF_SENSITIVE_SEGS1;
OUTPUT_MSG.TEXT_OUT=' NBR SENSGMTS='||NBR_SENSGMT
  ||', KEYFDBK AREA='||KEY_FEEDBACK_AREA1||','||NL;
CALL PLITDLI(THREE,ISRT,TERMINAL,OUTPUT_MSG);
GO TO B;
/**********************************************************************/
/*        INCORRECT CALL FUNCTION OUTPUT MESSAGE                      */
/**********************************************************************/
INCORR:  C5=27;
Z18=WE;
FUNCTION=FUNC||NL;
CALL PLITDLI(THREE,ISRT,TERMINAL,CALL_FUNC_ERR);
GO TO FINAL;
/**********************************************************************/
/*           MESSAGE STATUS CODE OUTPUT MESSAGE                       */
/**********************************************************************/
ERRO:  STAT_CODE=STAT_CODES;
    C4=19;
    Z14=WI:
    CALL PLITDLI(THREE,ISRT,TERMINAL,ERR);
FINAL:  END DLITPLI;
```

Output of PL/I Message Program

Entry to 2740 or 2260:

```
TUBE
```

Result is matrix on which to state parameter:

```
     STATE THE FOLLOWING FOR OBTAINING DATA FROM DATA BASE DI31PH01 BY
     FILLING IN THE UNDERLINES AND ADDING A NEW LINE SYMBOL.
 TRAN  FUNC SGMT-NAME SGMT-FLD-NAME R/O  COMP-VALUE
¢TUBE ____ ________ (________       __   ______)
                    ________ (________       __   ______)
                    ________ (________       __   ______)
```

Note: On 2260 the TUBE appears on line 4 of 2260. The ¢ is the Start MI symbol.

Entry to 2740 or 2260:

```
¢TUBE  GU   PARENT   (KEY1          =   000020)
            LEVEL21  (KEY21         =   000021)
            LEVEL31
```

Result in Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=GU     SSA1=PARENT  (KEY1      =000020)
                      SSA2=LEVEL21 (KEY21     =000021)
                      SSA3=LEVEL31
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=GU  , STATUS CODE=XX, IOAREA=
000016TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTT
 DBD NAME=DI31PH01, SGMT LEVEL=03, DB STAT CODES=  , PROC OPT=A   ,
 DLIJCB ADDR=           108, SGMT NAME FDBK=LEVEL31 ,LGTH FDBK=          18,
 NBR SENSGMTS=             5, KEYFDBK AREA=000020000021000016          ,
```

Entry to 2740 and 2260:

```
TUBE  GN  PARENT
```

Result of Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=GN    SSA1=PARENT
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=GN  , STATUS CODE=XX, IOAREA=
000020PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
PPPPPPPPPPP
DBD NAME=DI31PH01, SGMT LEVEL=01, DB STAT CODES=  , PROC OPT=A   ,
DLIJCB ADDR=           108, SGMT NAME FDBK=PARENT  ,LGTH FDBK=          6,
NBR SENSGMTS=             5, KEYFDBK AREA=000020000021000016          ,
```

Entry to 2740 and 2260:

```
TUBE  GU  PARENT    (KEY1        =  000040)
```

Result of Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=GU    SSA1=PARENT  (KEY1     =000040)
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=GU  , STATUS CODE=XX, IOAREA=
000040PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
PPPPPPPPPPP
DBD NAME=DI31PH01, SGMT LEVEL=01, DB STAT CODES=  , PROC OPT=A   ,
DLIJCB ADDR=           108, SGMT NAME FDBK=PARENT  ,LGTH FDBK=          6,
NBR SENSGMTS=             5, KEYFDBK AREA=000040000021000016          ,
```

Entry to 2260:

```
     STATE THE FOLLOWING FOR OBTAINING DATA FROM DATA BASE DI31PH01 BY
     FILLING IN THE UNDERLINES AND ADDING A NEW LINE SYMBOL.
 TRAN  FUNC SGMT-NAME SGMT-FLD-NAME R/O  COMP-VALUE
¢TUBE  ISRT PARENT    (KEY1           =   000045)
            _________(_______        __  _______)
            _________(_______        __  _______)
```

Result in Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=ISRT  SSA1=PARENT  (KEY1     =000045)
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=ISRT, STATUS CODE=XX, IOAREA=
000045IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
DBD NAME=DI31PH01, SGMT LEVEL=01, DB STAT CODES=  , PROC OPT=A   ,
DLIJCB ADDR=              108, SGMT NAME FDBK=PARENT  ,LGTH FDBK=            6,
NBR SENSGMTS=                5, KEYFDBK AREA=000040000021000016            ,
```

Entry to 2260:

```
     STATE THE FOLLOWING FOR OBTAINING DATA FROM DATA BASE DI31PH01 BY
     FILLING IN THE UNDERLINES AND ADDING A NEW LINE SYMBOL.
 TRAN  FUNC SGMT-NAME SGMT-FLD-NAME R/O  COMP-VALUE
¢TUBE  GU   PARENT    (KEY1           =   000045)
            _________(_______        __  _______)
            _________(_______        __  _______)
```

Result in Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=GU    SSA1=PARENT  (KEY1     =000045)
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=GU  , STATUS CODE=XX, IOAREA=
000045IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
DBD NAME=DI31PH01, SGMT LEVEL=01, DB STAT CODES=  , PROC OPT=A   ,
DLIJCB ADDR=              108, SGMT NAME FDBK=PARENT  ,LGTH FDBK=            6,
NBR SENSGMTS=                5, KEYFDBK AREA=000045000021000016            ,
```

Entry to 2260:

```
     STATE THE FOLLOWING FOR OBTAINING DATA FROM DATA BASE DI31PH01 BY
     FILLING IN THE UNDERLINES AND ADDING A NEW LINE SYMBOL.
 TRAN  FUNC SGMT-NAME SGMT-FLD-NAME R/O  COMP-VALUE
¢TUBE  REPL PARENT    (KEY1           =   000045)
            _________(_______        __  _______)
            _________(_______        __  _______)
```

Result in Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=REPL  SSA1=PARENT  (KEY1     =000045)
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=REPL, STATUS CODE=XX, IOAREA=
000045RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
DBD NAME=DI31PH01, SGMT LEVEL=01, DB STAT CODES=  , PROC OPT=A   ,
DLIJCB ADDR=              108, SGMT NAME FDBK=PARENT  ,LGTH FDBK=            6,
NBR SENSGMTS=                5, KEYFDBK AREA=000045000021000016            ,
```

Entry to 2260:

```
     STATE THE FOLLOWING FOR OBTAINING DATA FROM DATA BASE DI31PH01 BY
     FILLING IN THE UNDERLINES AND ADDING A NEW LINE SYMBOL.
 TRAN  FUNC SGMT-NAME SGMT-FLD-NAME R/O  COMP-VALUE
¢TUBE  DLET PARENT    (KEY1           =   000045)
            _________(_______        __  _______)
            _________(_______        __  _______)
```

Result in Call:

```
     ANSWER TO REQUEST(CALL) FOR DATA FROM DATA BASE DI31PH01.
CALL WAS: FUNC=DLET  SSA1=PARENT  (KEY1     =000045)
RETURNED DATA:  SUCCESSFUL   OPERATION, FUNC=DLET, STATUS CODE=XX, IOAREA=
000045DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DBD NAME=DI31PH01, SGMT LEVEL=01, DB STAT CODES=  , PROC OPT=A   ,
DLIJCB ADDR=              108, SGMT NAME FDBK=PARENT  ,LGTH FDBK=            6,
NBR SENSGMTS=                5, KEYFDBK AREA=000045000021000016            ,
```

Data Base Description (DBD) Generation Example for PL/I Message Program

The data base used is as described in Figure 46, except that DBNAME is DIPH01.

PSBGEN Example for PL/I Message Program

```
STMT   SOURCE STATEMENT

    1           PCB    TYPE=TP,LTERM=MASTER
    2           PCB     TYPE=DB,DBDNAME=DI31PH01,PROCOPT=A,KEYLEN=30
    3           SENSEG PARENT
    4           SENSEG LEVEL21,PARENT
    5           SENSEG LEVEL31,LEVEL21
    6           SENSEG LEVEL22,PARENT
    7           SENSEG LEVEL32,LEVEL22
    8           PSBGEN     LANG=PL/I,PSBNAME=HIMAJC03
    9                  *,*************************************************
   10                  *,*                                               *
   11+          PUNCH '          SETSSI  00000001
    +                 '
   12                  *,*                                               *
   13                  *,*************************************************
   14                  *,*************************************************
   15                  *,*                                               *
   17                  *,*                                               *
   18                  *,*************************************************
   19+HIMAJC03 CSECT
   20+PSBTOP   EQU    *
   21+         DC     F'0' RESERVED
   22+         DC     A(ENDTPPCB-PSBTOP) PST ADDRESS
   23+         DC     X'0' RESERVED
   24+         DC     BL1'00010000' CODE BYTE
   25+         DC     AL2(ENDTPPCB-PSBTOP) PSB SIZE
   26+         DC     H'4' TP OFFSET TO LAST TPPCB
   27+         DC     H'8' DB OFFSET TO FIRST DBPCB
   28+         DC     F'0' I/O PCB ADDRESS
   29+         DC     A(PCB1-PSBTOP) PCB ADDRESS
   30+         DC     X'80',AL3(PCB2) LAST PCB ADDRESS
   32+         DS     0F
   33+PCB1     EQU    *
   34+*********************************************************************
   35+*********************************************************************
   36+*
   37+*        DOPE VECTORS FOR TERMINAL PCB
   38+*
   39+         DC     A(TNAME1-*),H'8',H'8' CHAR(8)  TERMINAL NAME DV.
   40+         DC     A(CNT1-*),H'16',H'16' BIT(16)  RESERVED
   41+         DC     A(STAT1-*),H'2',H'2' CHAR(2) STATUS CODE DOPE VECTOR.
   42+         DC     A(PREF11-*) DEC(7) PREFIX - DATE DOPE VECTOR
   43+         DC     A(PREF21-*) DEC(7) PREFIX - TIME DOPE VECTOR
   44+         DC     A(PREF31-*) FIXED BIN(31) PREFIX - MSG NBR DOPE VECTOR
   45+*********************************************************************
   46+*********************************************************************
   47+TNAME1   DC     CL8'MASTER' LOGICAL TERMINAL NAME
   48+CNT1     DC     BL2'0' RESERVED
   49+STAT1    DC     CL2' ' STATUS CODE
   50+PREF11   DC     F'0' PREFIX - DATE 0CYYDDDC
   51+PREF21   DC     F'0' PREFIX - TIME HHMMSSTC
   52+PREF31   DC     F'0' PREFIX - MESSAGE NUMBER
   53+OPTL1    DC     F'0' LAST TTR
   54+OPTN1    DC     F'0' NEXT TTR
   55+SMBPT1   DC     F'0' CNT/SMB PTR
   56+*********************************************************************
   57+*********************************************************************
   58+ENDTPPCB EQU    *
   59+PCB2     EQU    *
   61+*********************************************************************
   62+*********************************************************************
   63+*
   64+*        DOPE VECTORS FOR DATA BASE PCB
   65+*
   66+         DC     A(DNAME2-*),H'8',H'8' CHAR(8)  DB NAME DOPE VECTOR.
   67+         DC     A(LEVFD2-*),H'2',H'2' CHAR(2)  LEVEL FEEDBACK DV.
   68+         DC     A(STACD2-*),H'2',H'2' CHAR(2)  STATUS CODE DV.
   69+         DC     A(PROCO2-*),H'4',H'4' CHAR(4)  PROCESSING OPTIONS DV.
   70+         DC     A(JCBAD2-*) FIXED BIN(31)  JCB ADDRESS DOPE VECTOR.
   71+         DC     A(SEGFD2-*),H'8',H'8' CHAR(8)  SEG FEEDBACK DV.
   72+         DC     A(KEYLN2-*) FIXED BIN(31) LEVEL FEEDBACK DOPE VECTOR.
   73+         DC     A(NOSS2-*) FIXED BIN(31) NO SEN SEG DOPE VECTOR.
   74+         DC     A(KEYFD2-*),H'30',H'30' CHAR(N) N=MAX CONCATENATED KEY
   75+*********************************************************************
   76+*********************************************************************
   77+         DS     0F
   78+DNAME2   DC     CL8'DI31PH01' DBD NAME
   79+LEVFD2   DC     H'60' LEVEL FEEDBACK, IF PL/I DV SIZE AT LOAD.
   80+STACD2   DC     CL2' ' STATUS CODES
   81+PROCO2   DC     CL4'A' PROCESSING OPTIONS
   82+JCBAD2   DC     F'0' JCB ADDRESS
   83+SEGFD2   DC     CL8' ' SEGMENT NAME FEEDBACK
   84+KEYLN2   DC     F'30' KEY FEEDBACK LENGTH MAXINUM
   85+NOSS2    DC     F'5' NO OF SENSITIVE SEGMENTS
   86+KEYFD2   DS     CL30' ' KEY FEEDBACK AREA
   87+SN21     DC     CL8'PARENT' SEGMENT NAME
   88+SN22     DC     CL8'LEVEL21' SEGMENT NAME
   89+SN23     DC     CL8'LEVEL31' SEGMENT NAME
   90+SN24     DC     CL8'LEVEL22' SEGMENT NAME
   91+SN25     DC     CL8'LEVEL32' SEGMENT NAME
   92+*********************************************************************
   93+*********************************************************************
   94           END
```

PSBGEN for PL/I Batch Program

```
STMT   SOURCE STATEMENT

   1           PCB     TYPE=DB,DBDNAME=DI31PH01,PROCOPT=A,KEYLEN=30
   2           SENSEG PARENT
   3           SENSEG LEVEL21,PARENT
   4           SENSEG LEVEL31,LEVEL21
   5           SENSEG LEVEL22,PARENT
   6           SENSEG LEVEL32,LEVEL22
   7           PCB    TYPE=DB,DBDNAME=DI31PH02,PROCOPT=A,KEYLEN=30
   8           SENSEG PARENT
   9           SENSEG LEVEL21,PARENT
  10           SENSEG LEVEL31,LEVEL21
  11           SENSEG LEVEL22,PARENT
  12           SENSEG LEVEL32,LEVEL22
  13           PSBGEN LANG=PL/I,PSBNAME=HIBAJC01
  14                 *,***************************************************
  15                 *,*                                                  *
  16+          PUNCH '        SETSSI  00000000                               X
    +                '
  17                 *,*                                                  *
  18                 *,***************************************************
  19                 *,***************************************************
  20                 *,*                                                  *
```

MESSAGE PROCESSING PROGRAM SIMULATION EXAMPLE

The following is an example of a typical COBOL program that might be written to test a message program in a Type 3 processing region. For further details see Chapter 6, "Message Processing Region Simulation".

Simulation Module A

(See Figure 40)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'CAB'.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    INOUT-PCB
        02  IO-TERMINAL     PICTURE X(8).
        02  IO-RSERVE       PICTURE XX.
        02  IO-STATUS       PICTURE XX.
        02  I-PREFIX        PICTURE X(12).
 LINKAGE SECTION.
 01  DB-PCB.
       02  DATA-BAS-DESC    PICTURE X(71).
 PROCEDURE DIVISION.
       ENTER LINKAGE.
          ENTRY 'DLITCBL' USING DB-PCB.
       ENTER COBOL.
       ENTER LINKAGE.
          CALL 'TEST' USING INOUT-PCB, DB-PCB.
       ENTER COBOL.
       STOP RUN.
```

Message Processing Program - Figure 40

Section of Message Processing program being tested shows entry point and call to Message Input and Output (Message Simulator Interface B):

```
START-OUT.
    ENTER LINKAGE.
     ENTRY 'TEST' USING TERMINAL INOUT-PCB,DB-PCB.
    ENTER COBOL.
    ENTER LINKAGE.
     CALL 'GEORGEI' USING GET-UNIQUE,INOUT-PCB, LINE-INPUT.
    ENTER COBOL.
```

Simulation Module B

(See Figure 40)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'IMSTEST'.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MESSAGE-FILE  ASSIGN TO 'TESTIN'  UTILITY.
    SELECT TEST-OUTPUT-FILE  ASSIGN  TO 'TESTOUT' UTILITY.
DATA DIVISION.
FILE SECTION.
FD  MESSAGE-FILE
    RECORDING MODE IS V
    DATA RECORD IS INPUT-MESSAGE.
    01  INPUT-MESSAGE                 PICTURE IS X(143).
FD  TEST-OUTPUT-FILE
    BLOCK CONTAINS 10 RECORDS
    DATA RECORD IS PRINT-LINE.
    01  PRINT-LINE                    PICTURE IS X(133).
WORKING-STORAGE  SECTION.
    77  OPEN-SWITCH  PICTURE X   VALUE ' '.
    77  END-SWITCH   PICTURE X   VALUE ' '.
    77  MESSAGE-SIZE-WORK  PICTURE S9(4) VALUE 0
                     USAGE COMPUTATIONAL.
    77  BAD-FUNCTION-CODE  PICTURE XX      VALUE '0A'.
    77  NO-DATA-CODE.      PICTURE XX      VALUE '0C'.
    77  REC-SWT    PICTURE X  VALUE ' '.
    77  MESS-OUT  PICTURE X  VALUE ' '.
    77  C-329     PICTURE   S9(6)    VALUE  329
                  USAGE COMPUTATIONAL.
    01  MESSAGE-IN-WORK-AREA.
        02   HEADER-DATA-IN.
             03   MESSAGE-COUNT                  PICTURE  9(4).
             03   MESSAGE-TYPE                   PICTURE  X.
             03   TERMINAL-NAME                  PICTURE  X(8).
        02   MESSAGE-TEXT.
             03  FILLER PICTURE X OCCURS 130 TIMES
                 DEPENDING ON MESSAGE-SIZE-WORK.
    01. TEST-OUTPUT-HEADER.
        02  FILLER PICTURE X(18) VALUE
            '   MESSAGE TYPE = '.
        02 FILLER.
           03 IN-OR-OUT-MESSAGE                  PICTURE X.
           03 HEAD-OR-BODY                       PICTURE X.
        02 FILLER  PICTURE X(18)  -              VALUE
            '. MESSAGE COUNT = '.
        02 OUTPUT-COUNT                          PICTURE  9999.
        02 FILLER  PICTURE X(13)                 VALUE
            '. TERMINAL = '.
        02 OUTPUT-TERMINAL                       PICTURE  X(8).
        02   FILLER  PICTURE  XX  VALUE SPACES.
        02   OUT-FUN  PICTURE  XXXX.
    01  TEST-OUTPUT-TEXT.
        02 TEST-OUTPUT-CHAR  OCCURS 130 TIMES
                                               PICTURE X.
LINKAGE SECTION.
    01  INOUT-PCB.
        02   IO-TERMINAL          PICTURE X(8).
        02   IO-RESERVE           PICTURE XX.
        02   IO-STATUS            PICTURE XX.
        02  I-PREFIX   PICTURE X(12).
    01  FUNCTION   PICTURE  XXXX.
    01  IO-AREAS-RECORD.
        02  RCC PICTURE S9(4) USAGE COMPUTATIONAL.
         02  RCC-ZEROS   PICTURE  XX.
        02  TEXT.
            03  FILLER PICTURE X OCCURS 130 TIMES
                DEPENDING ON MESSAGE-SIZE-WORK.
PROCEDURE DIVISION.
    ENTER LINKAGE.
    ENTRY 'GEORGEI' USING FUNCTION, INOUT-PCB, IO-AREAS-RECORD.
    ENTER COBOL.
OPEN-FILES.
    IF OPEN-SWITCH = '1' GO TO PROCESS-X.
        MOVE 0 TO TALLY.
    OPEN INPUT MESSAGE-FILE
         OUTPUT TEST-OUTPUT-FILE.
    MOVE '1' TO OPEN-SWITCH.
PROCESS-X.
    IF FUNCTION = 'GU  ' GO TO GET-HEADER.
    IF FUNCTION = 'GN  ' GO TO GET-BODY.
    IF FUNCTION = 'ISRT' GO TO WRITE-REPLY.
    MOVE BAD-FUNCTION-CODE  TO IO-STATUS.
RETURN-TO-APPLICATION.
    ENTER LINKAGE.
    RETURN.
    ENTER COBOL.
FORMAT-INPUT-MESSAGE.
    MOVE 'I' TO IN-OR-OUT-MESSAGE.
    MOVE MESSAGE-TYPE TO HEAD-OR-BODY.
    MOVE MESSAGE-COUNT TO OUTPUT-COUNT.
    MOVE TERMINAL-NAME TO OUTPUT-TERMINAL.
    MOVE MESSAGE-TEXT  TO TEST-OUTPUT-TEXT.
SET-UP-FOR-USER.
    MOVE MESSAGE-COUNT TO RCC.
    MOVE LOW-VALUES  TO RCC-ZEROS.
    MOVE TERMINAL-NAME TO IO-TERMINAL.
    MOVE MESSAGE-TEXT TO     TEXT.
     MOVE '  ' TO IO-STATUS.
READ-MESSAGE-FILE.
    IF  END-SWITCH = '1' GO TO FINISH-UP.
        MOVE 130 TO MESSAGE-SIZE-WORK.
    READ MESSAGE-FILE INTO MESSAGE-IN-WORK-AREA
                AT END MOVE '1' TO END-SWITCH
                GO TO READ-MESSAGE-FILE.
    COMPUTE  MESSAGE-SIZE-WORK = MESSAGE-COUNT - 4.
    PERFORM  FORMAT-INPUT-MESSAGE.
    PERFORM  WRITE-TEST-OUTPUT-FILE.
WRITE-TEST-OUTPUT-FILE.
        MOVE  FUNCTION  TO  OUT-FUN.
    WRITE PRINT-LINE  FROM  TEST-OUTPUT-HEADER.
    WRITE PRINT-LINE  FROM  TEST-OUTPUT-TEXT.
GET-HEADER.
    IF  REC-SWT NOT = 'H'
            PERFORM  READ-MESSAGE-FILE
            GO TO REC-GOT.
    COMPUTE  MESSAGE-SIZE-WORK = MESSAGE-COUNT - 4.
    PERFORM  FORMAT-INPUT-MESSAGE.
    PERFORM  WRITE-TEST-OUTPUT-FILE.
REC-GOT.
    IF MESSAGE-TYPE NOT = TD 'H' GO TO GET-HEADER.
    PERFORM  SET-UP-FOR-USER. MOVE ' ' TO REC-SWT.
    GO  TO RETURN-TO-APPLICATION.
GET-BODY.
    PERFORM  READ-MESSAGE-FILE.
     IF MESSAGE-TYPE = 'H' NEXT SENTENCE ELSE
            MOVE 'H' TO  REC-SWT
            MOVE   'OD'          TO IO-STATUS
            GO TO RETURN-TO-APPLICATION.
    PERFORM  SET-UP-FOR-USER.
    GO TO RETURN-TO-APPLICATION.
WRITE-REPLY.
    MOVE IO-TERMINAL TO OUTPUT-TERMINAL.
        COMPUTE   MESSAGE-SIZE-WORK  =  RCC - 4.
    MOVE  RCC TO  OUTPUT-COUNT.
    MOVE  'O' TO  IN-OR-OUT-MESSAGE.
    MOVE ' ' TO HEAD-OR-BODY.
    MOVE  TEXT TO   TEST-OUTPUT-TEXT.
    MOVE  MESS-OUT  TO  IO-STATUS.
    PERFORM  WRITE-TEST-OUTPUT-FILE.
FINISH-UP.
    IF FUNCTION = 'GU  '  MOVE 'OC' TO IO-STATUS.
    IF FUNCTION = 'GN  '  MOVE 'OD' TO IO-STATUS.
    GO TO RETURN-TO-APPLICATION.
/*
```

INPUT EDITOR EXAMPLE

The following example illustrates the use of the input editor. This example shows the relevant coding both before and after the editing process.

A. Edit table before editing occurs

```
 01  EDIT-TABLE-EXAMPLE.

       02  TABLE-HEADER.
           03  EDITOR-RESERVE  PICTURE S9(5)    COMPUTATIONAL
                           VALUE 0.
           03  LANGUAGE    PICTURE X   VALUE 'C'.
           03  AUDIT-TRAIL-CODE    PICTURE X   VALUE 'N'.
           03  FIELD-FEEDBACK-RESET    PICTURE X   VALUE 'Y'.
           03  EDIT-TABLE-FEEDBACK PICTURE X   VALUE '0'.
           03  TABLE-HEADER-LENGTH PICTURE S999 COMPUTATIONAL
                           VALUE 28.
           03  EDIT-START-POSITION PICTURE S999 COMPUTATIONAL
                           VALUE 5.
           03  NUMB-DELIMITER-ENTRIES  PICTURE S999
                           COMPUTATIONAL   VALUE 1.
           03  LENGTH-OF-A-DELIM-ENT   PICTURE S999
                           COMPUTATIONAL   VALUE 4.
           03  NUMBER-OF-FIELD-ENTRIES PICTURE S999
                           COMPUTATIONAL   VALUE 4.
           03  LENGTH-OF-A-FIELD-ENT   PICTURE S999
                           COMPUTATIONAL   VALUE 26.
           03  NO-OF-VALID-POSTL-FLDS  PICTURE S999
                           COMPUTATIONAL   VALUE 0.
           03  NO-OF-VALID-FLDS-W-KEYS PICTURE S999
                           COMPUTATIONAL   VALUE 0.
           03  NO-OF-INVALID-FLDS-W-KEYS   PICTURE S999
                           COMPUTATIONAL   VALUE 0.

       02  DELIMITER-ENTRY-1.
           03  LENGTH  PICTURE S999    COMPUTATIONAL   VALUE 1.
           03  DELIMITER   PICTURE X   VALUE ';'.
           03  FILLER  PICTURE X.

       02  FIELD-ENTRY-1.
           03  DEL-LEAD-FILL-CHAR  PICTURE X   VALUE 'Y'.
           03  LEAD-FILL-CHAR  PICTURE X   VALUE ' '.
           03  DEL-TRAIL-FILL-CHAR PICTURE X   VALUE 'Y'.
           03  TRAILING-FILL-CHAR  PICTURE X   VALUE ' '.
           03  MIN-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                           VALUE 3.
           03  MAX-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                           VALUE 20.
           03  OUTPUT-START    PICTURE S999    COMPUTATIONAL
                           VALUE 1.
           03  OUTPUT-JUSTIFICATION    PICTURE X   VALUE 'L'.
           03  FIELD-EDIT-STATUS-CODE-1    PICTURE X   VALUE '0'.
           03  VALID-OUTPUT-ACT-CODE   PICTURE X   VALUE 'Y'.
           03  VALID-OUTPUT-FILL-CHAR  PICTURE X   VALUE ' '.
           03  INVALID-OUTPUT-ACT-CODE PICTURE X   VALUE 'F'.
           03  INVALID-OUTPUT-FILL-CHR PICTURE X   VALUE '*'.
           03  LENGTH-FIELD-KEYWORD    PICTURE S999
                           COMPUTATIONAL   VALUE 0.
           03  NUMBER-OF-CHECK-CHARS   PICTURE S999
                           COMPUTATIONAL   VALUE 1.
           03  CHECK-CHARACTER PICTURE X   VALUE 'E'.
           03  FILLER  PICTURE XXXXX.
```

```
02  FIELD-ENTRY-2.
    03  DEL-LEAD-FILL-CHAR  PICTURE X   VALUE 'Y'.
    03  LEAD-FILL-CHAR  PICTURE X   VALUE ' '.
    03  DEL-TRAIL-FILL-CHAR PICTURE X   VALUE 'Y'.
    03  TRAILING-FILL-CHAR  PICTURE X   VALUE ' '.
    03  MIN-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                    VALUE 2.
    03  MAX-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                    VALUE 9.
    03  OUTPUT-START    PICTURE S999    COMPUTATIONAL
                    VALUE 21.
    03  OUTPUT-JUSTIFICATION    PICTURE X   VALUE 'R'.
    03  FIELD-EDIT-STATUS-CODE-2    PICTURE X   VALUE '0'.
    03  VALID-OUTPUT-ACT-CODE   PICTURE X   VALUE 'Y'.
    03  VALID-OUTPUT-FILL-CHR   PICTURE X   VALUE '0'.
    03  INVALID-OUTPUT-ACT-CODE PICTURE X   VALUE 'F'.
    03  INVALID-OUTPUT-FILL-CHR PICTURE X   VALUE '0'.
    03  LENGTH-FIELD-KEYWORD    PICTURE S999
                    COMPUTATIONAL   VALUE 0.
    03  NUMBER-OF-CHECK-CHARS   PICTURE S999
                    COMPUTATIONAL   VALUE 1.
    03  CHECK-CHARACTER PICTURE X   VALUE 'N'.
    03  FILLER  PICTURE XXXXX.

02  FIELD-ENTRY-3
    03  DEL-LEAD-FILL-CHAR  PICTURE X   VALUE 'Y'.
    03  LEAD-FILL-CHAR  PICTURE X   VALUE ' '.
    03  DEL-TRAIL-FILL-CHAR PICTURE X   VALUE 'Y'.
    03  TRAILING-FILL-CHAR  PICTURE X   VALUE ' '.
    03  MIN-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                    VALUE 2.
    03  MAX-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                    VALUE 3.
    03  OUTPUT-START    PICTURE S999    COMPUTATIONAL
                    VALUE 30.
    03  OUTPUT-JUSTIFICATION    PICTURE X   VALUE 'R'.
    03  FIELD-EDIT-STATUS-CODE-3    PICTURE X   VALUE '0'.
    03  VALID-OUTPUT-ACT-CODE   PICTURE X   VALUE 'Y'.
    03  VALID-OUTPUT-FILL-CHR   PICTURE X   VALUE '0'.
    03  INVALID-OUTPUT-ACT-CODE PICTURE X   VALUE 'F'.
    03  INVALID-OUTPUT-FILL-CHR PICTURE X   VALUE '*'.
    03  LENGTH-FIELD-KEYWORD    PICTURE S999
                    COMPUTATIONAL   VALUE 4.
    03  NUMBER-OF-CHECK-CHARS   PICTURE S999
                    COMPUTATIONAL   VALUE 1.
    03  FIELD-KEYWORD   PICTURE XXXX    VALUE 'LOC='.
    03  CHECK-CHARACTERS    PICTURE X   VALUE 'N'.
    03  FILLER  PICTURE X.

02  FIELD-ENTRY-4
    03  DEL-LEAD-FILL-CHAR  PICTURE X   VALUE 'Y'.
    03  LEAD-FILL-CHAR  PICTURE X   VALUE ' '.
    03  DEL-TRAIL-FILL-CHAR PICTURE X   VALUE 'Y'.
    03  TRAILING-FILL-CHAR  PICTURE X   VALUE ' '.
    03  MIN-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                    VALUE 4.
    03  MAX-FIELD-LEN   PICTURE S999    COMPUTATIONAL
                    VALUE 4.
    03  OUTPUT-START    PICTURE S999    COMPUTATIONAL
                    VALUE 33.
    03  OUTPUT-JUSTIFICATION    PICTURE X   VALUE 'R'.
    03  FIELD-EDIT-STATUS-CODE-4    PICTURE X   VALUE '0'.
    03  VALID-OUTPUT-ACT-CODE   PICTURE X   VALUE 'M'.
    03  VALID-OUTPUT-FILL-CHR   PICTURE X   VALUE ' '.
    03  INVALID-OUTPUT-ACTION-CODE  PICTURE X   VALUE 'F'.
```

```
        03  INVALID-OUTPUT-FILL-CHR PICTURE X   VALUE '*'.
        03  LENGTH-FIELD-KEYWORD     PICTURE S999    COMPUTATIONAL
                        VALUE 5.
        03  NUMBER-OF-CHECK-CHARS    PICTURE S999    COMPUTATIONAL
                        VALUE 1.
        03  FIELD-KEYWORD    PICTURE XXXXX   VALUE 'INSP='.
        03  CHECK-CHARACTERS     PICTURE X   VALUE 'N'.
```

B. Input string submitted to the editing process. The COUNT and FILLER are both halfword binary fields. The input string format is identical to the format of an IMS/360 input message from a terminal.

```
|COUNT|FILLER|ER14bbbbPARTbDATA;b127864;LOC=129b;INSP=1351
```

This character string is in the area named INPUT-AREA.

C. The following are the calls used to invoke the input editor.

In COBOL, the call is:

```
ENTER LINKAGE.

CALL 'EDITOR' USING INPUT-AREA, EDIT-TABLE-EXAMPLE
                OUTPUT-AREA.

ENTER COBOL.
```

In PL/I, the call is:

```
CALL EDITOR (INPUT_AREA,EDIT_TABLE_EXAMPLE,OUTPUT_AREA);
```

D. Edit table after editing. The following entries in the edit would be changed during the editing process. Their new values are:

```
01  EDIT-TABLE-EXAMPLE                          VALUES

    02  TABLE-HEADER
        . . . .

    03  EDIT-TABLE-FEEDBACK                        1

    . . . .

    03  NO-OF-VALID-POSTL-FLDS                     2

    . . . .

    03  NO-OF-VALID-FLDS-W-KEYS                    2

    . . . .

    03  FIELD-EDIT-STATUS-CODE-1                   1

    . . . .

    03  FIELD-EDIT-STATUS-CODE-2                   1

    . . . .

    03  FIELD-EDIT-STATUS-CODE-3                   1
```

```
. . . .

03   FIELD-EDIT-STATUS-CODE-4                 1

. . . .
```

E. The output string produced by the input editor. This data will be in the area named OUTPUT-AREA.

```
PARTbDATAbbbbbbbbbbb 000127864 129 1351
|--------20-----------|---9-----|-3-|--4-|
```

INDEX

SH20-0634-1

Information Management S/360 for the IBM S/360 PDM Printed in U.S.A. SH20-0634-1

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601
(USA only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENT FORM

Information Management S/360 for the IBM S/360
Program Description Manual

SH20-0634-1

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold

fold

fold

fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
FOLD ON TWO LINES, STAPLE AND MAIL.

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

Information Management S/360 for the IBM S/360
Program Description Manual

SH20-0634-1

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold

fold

fold

fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, STAPLE AND MAIL.

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM® Technical Newsletter

Base Publ. No. SH20-0634-1

This Newsletter No. SN20-2339

Date March 19, 1971

Previous Newsletter Nos. None

INFORMATION MANAGEMENT SYSTEM/360 FOR THE
IBM SYSTEM/360 PROGRAM DESCRIPTION

PROGRAM NUMBER 5736-CX3

This Technical Newsletter, a part of Version 1, Modification Level 2, of Information Management System/360, provides replacement pages for the subject manual. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are listed below:

Front cover
75 - 76
83 - 84
135 - 136

Changes are indicated by vertical rules in the left margin.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Technical Publications Dept., 1133 Westchester Avenue, White Plains, N. Y. 10604

Printed in U.S.A.