***IMS DB***

**IMS DB IMP POINTS**

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

* **Abbrevations**

**IMS -** Information Management System

**DC –** Data Communication

**DBMS –** Database Management System.

**DBD -**  Database Descriptor

**PSB –** Program Specification Block

**PCB –** Program Communication Block

**RPS –** Root Anchor Points

**SSA-** Segment Search Argument

**MFS-**Message Format Service

* Go to **FAI(Fileaid IMS)** and press **SC**(Select Character) before the root to view all data present in IMSDB segments related to primary key specified in root segment.
* START TSO @DAT for binding packages and plans.

If we get bind errors(-805) then check PRINT99

* IMS imp terminology **Segment, Segment Type, Segment Occurrence, Database Record, Dependent Segments, Twin Segments,**
* Data retrieved from IMS DB **Top-to-Bottom, Left-to-Right.**

First, a root segment-occurrence is retrieved. Then, IMS DB digs as far deep as possible, right at the bottom, until it reaches the lowest level Segment-Occurrence. It then retrieves all the twin segment-occurrences at that level. Once, all the Twin Segments have been fetched, IMS moves back up in the hierarchy and retrieves any twins of the Parent. This process continues, till all the Database Records have been retrieved.

* Random-Access to Data in a Hierarchical-Database is possible by supplying a **concatenated-Key**.
* Every Segment can have a **Key-Field(or Sequence Field).**
* A **Concatenated-Key** is formed by combining the Key-Fields of different segment-types e.g. (DEPARTMENT-ID, EMPLOYEE-ID).
* Objectives of DBMS
  1. Reduce data redundancy
  2. Provide data communication facilities
  3. Reduce data maintenance
  4. Provide data integrity and security
  5. Provide indexing capabilities
* **IMS DB - IMS/Database Manager** as the name implies manages the IMS databases. It is used for physical storage creation and management and data retrieval.
* **IMS DC / IMS TM - IMS/Data Communications or IMS/Transaction Manager** handles online transaction processing system.
* The IMS resources are accessed by the application by calling a number of standard IMS functions.

Applications access these functions through **a standard application programming interface (API)** for both the Transaction Manager and Database Manager components. This interface is normally *referred to as* ***Data Language I (DL/I).***

* ***Physical structure of a DL/I data base isn’t specified in an application program***
* ***DL/I use a set of control blocks (DBDs and PSBs) to define a data base’s structure***
* ***Data Base Descriptor (DBD)***
  + *Describes the complete structure of a data base*
  + *A unique DBD for each DL/I data base*
* ***Program Specification Block (PSB)***
  + *Application program’s view of the Database*
  + *PSB Specifies*
    - *Data bases (one or more) a program can access,*
    - *Data elements a program can “see” in those data bases*
    - *The processing a program can do with the data elements*
  + *Application programs that have similar data base processing requirements can share a PSB*
* **One root segment-occurrence e.g. HR plus all the segment-occurrences sub-ordinate to -it for example RAM and RAJ, together constitute one   
  Database-Record.**
* The term Twin segments is applicable to Segment-Occurrences. Two children with same parent segment-occurrence are called twins.
* ***PCB*** *(Program Communication Block) refers to one data base.*
* *Within sensitive segments, the program has access to all fields*
* ***PROCOPT*** *parameter specifies the program’s processing options (generally in PCB)*
* Proc Options available in IMS DB
* *Commonly used Processing Options*
  + ***PROCOPT=G*** *means only read-only access*
  + ***PROCOPT=R*** *means read/replace access*
  + ***PROCOPT=I*** *means insert access allowed*
  + ***PROCOPT=D*** *means Read/Delete access*
  + ***PROCOPT=A*** *means all the above options present*
  + ***For GSAM DBs PROCOPT=LS for output and GS (Get Sequential) for input***
  + ***PROCOPT=L allows a 'load' into the DB. If VSAM DB, it should be empty prior to the load***
* **ACB & ACBGEN**
* ***ACB (Application Control Blocks)****: It is created by merging and expanding PSB’s and DBD’s into an IMS internal format when an application program is scheduled for execution.*
* ***ACBGEN:*** *The process of building ACB is called Block Building and is done by means of ACBGEN.*
* *IMS can build ACB’s either dynamically or it can prebuild them using ACB maintenance utility.*
* ***ACB’s cannot be prebuilt for GSAM DBD’s.***
* ***ACB’s can be prebuild for PSB’s that reference GSAM databases.***
* ***ACB’s save instruction, execution and direct-access wait time and improves performance in application scheduling.***
* *ACB’s are maintained in* ***IMS.ACBLIB*** *library.*
* *Running IMS program in batch*
* ***Batch program does not access IMS directly***
* ***JCL invokes the DL/I ‘batch initialization module’ DFSRRC00 which loads the application program and the required DL/I modules***
* ***The program and DL/I modules execute together***
* ***PCBs must be defined in the Linkage Section***
* *Linkage Section definition of a PCB is called a ‘****PCB Mask’***
* **ENTRY and GO BACK Statements**
* ***Format of the DL/I ENTRY Statement***

**ENTRY ‘DLITCBL’** **USING** PCB-name1

[PCB-name2...]

* ***GO BACK Statement***
  + *When a program ends, it passes control back to the DL/I*
  + *DL/I reallocate resources and closes the data base data sets*

*Use GO BACK and not a STOP RUN statement*

* **The DL/I Call**
* ***CALL statements are used to request DL/I services***
* ***Format of the DL/I call***

*Parameters you code on the CALL statement specify, among other things, the operation you want DL/I to perform*

**CALL ‘CBLTDLI’** USING DLI-function

PCB-mask

segment-io-area

[segment-search-argument(s)]

* ***CBLTDLI => ‘COBOL to DL/I’, is an interface module that is link edited with your program’s object module***

*PLITDLI, ASMTDLI are other options*

* ***The DL/I Function***
  + ***First parameter coded on any DL/I call***
  + ***Four character working storage field containing the function code***

***DL/I function codes***

**01 DLI-FUNCTIONS.**

05 DLI-GU PIC X(4) VALUE ‘GU  ’.

   05 DLI-GHU PIC X(4) VALUE ‘GHU ’.

   05 DLI-GN PIC X(4) VALUE ‘GN  ’.

   05 DLI-GHN PIC X(4) VALUE ‘GHN ’.

   05 DLI-GNP PIC X(4) VALUE ‘GNP ’.

   05 DLI-GHNP PIC X(4) VALUE ‘GHNP’.

   05 DLI-ISRT PIC X(4) VALUE ‘ISRT’.

   05 DLI-DLET PIC X(4) VALUE ‘DLET’.

   05 DLI-REPL PIC X(4) VALUE ‘REPL’.

   05 DLI-CHKP PIC X(4) VALUE ‘CHKP’.

   05 DLI-XRST PIC X(4) VALUE ‘XRST’.

   05 DLI-PCB PIC X(4) VALUE ‘PCB ’.

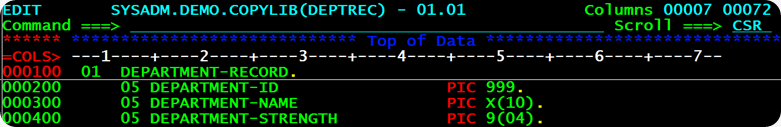
* ***COBOL doesn’t allow coding literals in a CALL statement***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Questions and their Explanation**

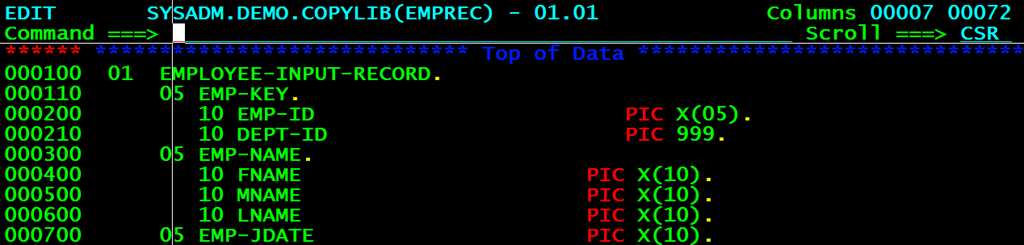
**Q. What is IMS/DB? (http://www.mainframes360.com/2010/11/introduction-to-imsdb.html)**

ANS. IMS (Information Management System) is the Hierarchical-Data Management System from IBM. When hierarchical data-structures are implemented using Flat-Files, it is the responsibility of Application-Programmers to ensure that it works flawlessly. Why storing huge-volumes of data having hierarchical-relationship in Flat-Files is a bad idea? Let's say, a list of Employees and the Departments in which they work – this Data is stored in Flat Files.



The Departments Data is stored in DEPARTMENT File. The record-layout of the DEPARTMENT File is shown in the picture above. Every Department has a DEPARTMENT-ID, that uniquely identifies the Department. Moreover, a Department has a DEPARMENT-NAME and the total number of employees working under that department are recorded in DEPARTMENT-STRENGTH.

Every Department of the company has several Employees working under it. The Employees data is stored in EMPLOYEE File. The snap above shows its record-layout. Each Employee is identified by Unique EMP-KEY, consisting of EMP-ID and DEPT-ID. The Employee's name EMP-NAME is further broken down in FNAME, MNAME and LNAME. The Employee’s Joining Date is stored in EMP-JDATE. This hierarchical relationship between DEPARTMENT File and EMPLOYEE File is depicted in the picture below.

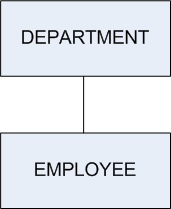


However, there are certain problems with this design.

1. Maintaining Data Consistency - For instance, if the Program adds a new Employee-Record, that has no DEPT-ID, no corresponding department, the DEPARTMENT and EMPLOYEE Files will no longer be synchronized. And the same is true, if a Department-Record is deleted, when active Employees work for that Department. These Employee-Records are orphaned out.

2. COBOL Program and Data Dependency – Because the structure of the Data(File Layout) is hard-coded or embedded in the COBOL Program accessing the data, whenever the data's structure changes(say you add one more field to the DEPARTMENT File), the COBOL Program has to be re-written and re-compiled all over again. **The Program heavily depends on the Data.**

When you use IMS/DB for storing data, difficulties like these don't come up. That's because the hierarchical relationships between data is now taken care of by IMS. IMS would use a single file for storing DEPARTMENT and its EMPLOYEE's data. The structural information is stored in IMS.



**Q. What is DBMS in IMS?**

ANS.A Database Management System is used to define and maintain the structure of the database.

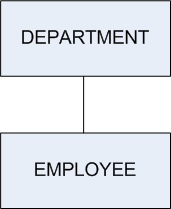
**Q. What is meant by Segment-Type, Segment-Occurrence and Levels in IMS?**

In IMS/DB, when data is stored about any real-world thing, you call it a Segment. For example, if you record information about Employees, Employee is One-Segment. Because you would be storing Departments is another segment. In IMS Jargon, I would call them DEPARTMENT Segment and EMPLOYEE Segment.   
  
I'll introduce two new terms as well – Segment Type and Segment Occurrence.

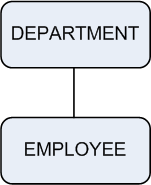
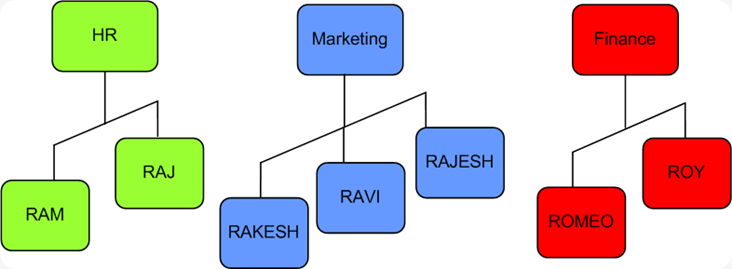
A segment-type is one category or class of data.

For example, DEPARTMENT Segment-Type.

Each instance of a Department is called a **Segment-Occurrence.** For example, you may have 3 segment-occurrences HR, MARKETING and FINANCE of the Segment-Type DEPARTMENT. Bear in mind, there can be only one segment-type of a particular kind, but infinite occurrences of it.   
  
**An IMS Hierarchical-Database can be looked upon as an inverted-tree. The Height of the Tree tells you the number-of-levels in the Tree.**

   
  
This Hierarchical-database has two Levels.**Q. What is meant by Root-Segment and Database Record?**

**ANS.**

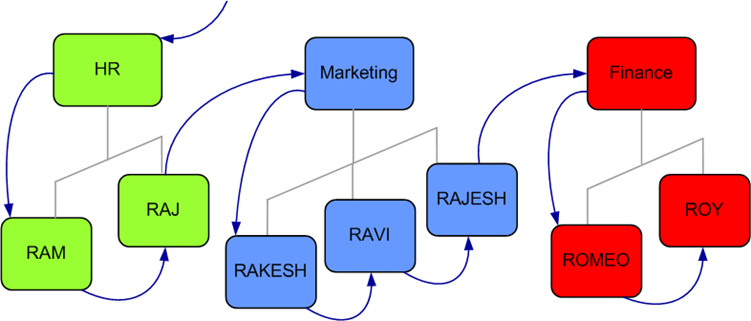
In an IMS Hierarchical-Database, the segment-type at the top of the hierarchy-tree is called the Root-Segment. For example, in the below database, the DEPARTMENT-Segment is called the Root-Segment.   
  
         
Suppose the DEPARTMENT Segment-type has 3 segment-occurrences HR, Marketing and Finance. Each of these Department segment-occurrences have Employee Segment-occurrences subordinate to them. HR Department has 2 sub-ordinate Employee Segment-Occurrences RAM and RAJ. Marketing Department has 3 sub-ordinate Employee-Segment occurrences RAKESH, RAVI and RAJESH. Finance Department has 2 sub-ordinate Employee Segment-occurrences ROMEO and ROY.   
  
     
**One root segment-occurrence e.g. HR plus all the segment-occurrences sub-ordinate to it for example RAM and RAJ, together constitute one   
Database-Record.** Thus, here there are 3 database-records, one for each root segment-occurrence. In all, there are 10 segment-occurrences in this IMS Database.

**Q. What are dependent, parent and Child and twin Segments?**

All the segments in an IMS Hierarchical DB, other than the Root-Segment are called Dependent-Segments. DEPARTMENT Segment is called Parent-Segment and EMPLOYEE Segment is its Child-Segment.   
  
The term Twin segments is applicable to Segment-Occurrences. Two children with same parent segment-occurrence are called twins. For example, RAKESH and RAVI are child of the same parent segment occurrence Marketing Department, so they are called Twin-Segments.

**Q. How data is accessed in an IMS Hierarchical Database?**

Data can be accessed in two-ways from an IMS-Database – **Sequential** and **Random mode** of access. When you access data from IMS DB Sequentially, you read it record-by-record. Within each record, the segments are accessed in the following general pattern:

**Top-to-Bottom, Left-to-Right.**  
First, a root segment-occurrence is retrieved. Then, IMS DB digs as far deep as possible, right at the bottom, until it reaches the lowest level Segment-Occurrence. It then retrieves all the twin segment-occurrences at that level. Once, all the Twin Segments have been fetched, IMS moves back up in the hierarchy and retrieves any twins of the Parent. This process continues, till all the Database Records have been retrieved. So, the sequence in which the Data is fetched is HR-RAM-RAJ, MARKETING-RAKESH-RAVI-RAJESH and FINANCE-ROMEO-ROY.   
  
     
  
**Random-Access to Data in a Hierarchical-Database is possible by supplying a Concatenated-Key.** Every Segment can have a **Key-Field (or Sequence Field).** For example, DEPARTMENT-ID may be a Key-Field of DEPARTMENT Segment. EMPLOYEE-ID would be Key-Field of Employee-Segment. **A Concatenated-Key is formed by combining the Key-Fields of different segment-types e.g. (DEPARTMENT-ID, EMPLOYEE-ID).** This way you can jump or skip directly to any particular Segment-Occurrence under any Database Record.

**Q. What are the objectives of DBMS in IMS?**

**ANS.**

1. Reduce data redundancy
2. Provide data communication facilities
3. Reduce data maintenance
4. Provide data integrity and security
5. Provide indexing capabilities

**Q. What is IMS (DB/DC)?**

ANS.IMS (Information Management System) is IBM’s hierarchical database management system. It has mainly two components: IMS DB and IMS DC (also known as IMS TM)

IMS DB - IMS/Database Manager as the name implies manages the IMS databases. It is used for physical storage creation and management and data retrieval.

IMS DC / IMS TM - IMS/Data Communications or IMS/Transaction Manager handles online transaction processing system.

**Q. What do you mean by Hierarchical Database?**

**ANS.**

1.Follows inverted tree structure

2.Data relationship is predefined by its structure

3.Program accesses the data through predefined paths.

**Q. What is meant by Segment-Type, Segment-Occurrence and Levels in IMS?**

**ANS.**

In IMS/DB, when data is stored about any real-world thing, you call it a Segment. For example, if you record information about Employees, Employee is One-Segment. Because you would be storing Departments is another segment. In IMS Jargon, I would call them DEPARTMENT Segment and EMPLOYEE Segment.

I'll introduce two new terms as well – Segment Type and Segment Occurrence. A segment-type is one category or class of data. For example, DEPARTMENT Segment-Type. Each instance of a Department is called a Segment-Occurrence. For example, you may have 3 segment-occurrences HR, MARKETING and FINANCE of the Segment-Type DEPARTMENT. Bear in mind, there can be only one segment-type of a particular kind, but infinite occurrences of it.

An IMS Hierarchical-Database can be looked upon as an inverted-tree. The Height of the Tree tells you the number-of-levels in the Tree.

This Hierarchical-database has two Levels.

***IMS DB Notes***

*IMS is an IBM program product that provides* ***transaction management*** *and* ***database management*** *functions for large commercial application systems. It was* ***originally introduced in 1968****.*

*There are two major parts to IMS,*

* *a* ***Data Communication manager (DC)****and*
* *a* ***Database Manager (DB).***

***Application programming Languages***

* Applications written to use IMS functions can be written in a number of programming languages.
* Programming languages currently supported are **Assembler, C, COBOL, Pascal, PL/I and REXX.**
* The IMS resources are accessed by the application by calling a number of standard IMS functions.
* Applications access these functions through **a standard application programming interface (API)** for both the Transaction Manager and Database Manager components.
* This interface is normally *referred to as* ***Data Language I (DL/I).***

***Control Blocks***

* ***Physical structure of a DL/I data base isn’t specified in an application program***
* ***DL/I uses a set of control blocks (DBDs and PSBs) to define a data base’s structure***
* ***Data Base Descriptor (DBD)***
  + *Describes the complete structure of a data base*
  + *A unique DBD for each DL/I data base*
* ***Program Specification Block (PSB)***
  + *Application program’s view of the Database*
  + *PSB Specifies*
    - *Data bases (one or more) a program can access,*
    - *Data elements a program can “see” in those data bases*
    - *The processing a program can do with the data elements*
  + *Application programs that have similar data base processing requirements can share a PSB*
* *Data Base Administrator (DBA) has to create DL/I control blocks*
* *DBDGEN and PSBGEN Control Statements*

**Q.SAMPLE DBDGEN (Explained)**

**ANS.**

**STMT SOURCE STATEMENT**

1 **PRINT** NOGEN

2 **DBD** **NAME**=INDBD,**ACCESS**=HIDAM

3 **DATASET** DD1=IN,DEVICE=3380

4 \*\*/   3380 DISK STORAGE

5 \*

6 **SEGM** NAME=INVENSEG, PARENT=0,**POINTER**=TB,BYTES=131

7 **LCHILD** NAME=(INPXPNTR,INPXDBD),**POINTER**=INDX

8 **FIELD** **NAME**=(INVENCOD,**SEQ**),**BYTES**=3,**START**=1,**TYPE**=C

9 FIELD NAME=INVENNAM,BYTES=30,START=4,TYPE=C

10 FIELD NAME=INVENADR,BYTES=30,START=34,TYPE=C

11 FIELD NAME=INVENCIT,BYTES=17,START=64,TYPE=C

12 FIELD NAME=INVENSTA,BYTES=2,START=81,TYPE=C

13 FIELD NAME=INVENZIP,BYTES=9,START=83,TYPE=C

14 FIELD NAME=INVENTEL,BYTES=10,START=92,TYPE=C

15 FIELD NAME=INVENCON,BYTES=30,START=102,TYPE=C

16 \*

17 SEGM NAME=INITMSEG,PARENT=INVENSEG,BYTES=48

18 FIELD NAME=(INITMNUM,SEQ),BYTES=5,START=1,TYPE=C

19 FIELD NAME=INITMDES,BYTES=35,START=6,TYPE=C

20 FIELD NAME=INITMPRC,BYTES=4,START=41,TYPE=P

21 FIELD NAME=INITMCST,BYTES=4,START=45,TYPE=P

22 \*

23 SEGM NAME=INLOCSEG, PARENT=INITMSEG,BYTES=21

24 FIELD NAME=(INLOCLOC,SEQ),BYTES=3,START=1,TYPE=C

25 FIELD NAME=INLOCONH,BYTES=4,START=4,TYPE=P

26 FIELD NAME=INLOCROP,BYTES=4,START=8,TYPE=P

27 FIELD NAME=INLOCONO,BYTES=4,START=12,TYPE=P

28 FIELD NAME=INLOCDAT,BYTES=6,START=16,TYPE=C

29 \*

30 **DBDGEN**

72 \*\*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

73 \*\*/     RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS

74 \*\*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

75 \*\*/\*                            \*NOTE2

76 \*\*/\* DEFINE CLUSTER (NAME(IN) NONINDEXED -

77 \*\*/\*           RECORDSIZE (2041,2041) -

78 \*\*/\*           CONTROLNTERVALSIZE (2048))

79 \*\*/\* \*NOTE2 - SHOULD SPECIFY DSNNAME FOR DD IN

80 \*\*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

162 \*\*/\*\*\*\*\*\*\*\*\*\*\*SEQUENCE FIELD\*\*\*\*\*\*\*\*\*\*\*\*\*

211 \*\*/\*\*\*\*\*\*\*\*\*\*\*SEQUENCE FIELD\*\*\*\*\*\*\*\*\*\*\*\*\*

325   FINISH

326   END

***Explanation***

* + ***First macro – DBD*** *– identifies the data base and specifies the DL/I access method*
  + ***Second macro – DATASET****- identifies the file that would contain the data base*
  + ***Symbolic name (IN)*** *identifies the data set in the JCL at execution time*
  + *Segment types are defined using the* ***SEGM macro***
  + *Segment hierarchical relationships are specified by the* ***PARENT*** *parameter on a SEGM macro*
    - ***PARENT= 0 or absence of PARENT parameter specifies root segment***
  + ***POINTER parameter and LCHILD macro are needed for HIDAM Databases***
  + *Only search fields need be specified in the DB*
  + ***FIELD macro*** *defines a field in the DB*
    - **START** ⇨*position of field within segment*
    - ***NAME***⇨*name of the field*
    - ***LENGTH***⇨*length of the field*
    - ***TYPE***⇨ *data type of the field*

|  |  |
| --- | --- |
| ***FIELD Macro TYPE Codes*** | ***Data Type*** |
| *C* | *Character* |
| *P* | *Packed decimal* |
| *Z* | *Zoned decimal* |
| *X* | *Hexadecimal* |
| *H* | *Half word Binary* |
| *F* | *Full word Binary* |

* + ***SEQ*** *parameter specifies a sequence field*
    - ***segment occurrences are added in sequence by values in these fields***

**Q. SAMPLE PSBGEN**

**ANS.**

**STMT SOURCE STATEMENT**

1 PRINT NOGEN

2 **PCB** **TYPE**=DB,**DBDNAME**=INDBD,PROCOPT=LS

3 **SENSEG** NAME=INVENSEG

4 SENSEG NAME=INITMSEG,PARENT=INVENSEG

5 SENSEG NAME=INLOCSEG,PARENT=INITMSEG

6 **PSBGEN** **PSBNAME**=INLOAD,LANG=COBOL

87 END

***Explanation***

* + ***PCB*** *(Program Communication Block) refers to one data base.*
  + *One PCB macro for each database accessed*
  + ***Segment Level Sensitivity***
    - *A program’s access to parts of the data base identified at the segment level*
    - *Within sensitive segments, the program has access to all fields*
  + ***Field level sensitivity***
    - *When the program accesses that segment, only sensitive fields are presented*
  + ***DBDNAME*** *parameter on the PCB macro specifies the name of the DBD*
  + ***KEYLEN*** *parameter specifies the length of the longest concatenated key the program can process in the data base*
  + ***PROCOPT*** *parameter specifies the program’s processing options*
  + *For each PCB macro, subordinate* ***SENSEG*** *macros identify the sensitive segments in the data base*
  + ***Names specified in the SENSEG macros must be segment names from the DBDGEN*** *for the data base named in the DBDNAME parameter of the PCB macro*
  + ***PSBGEN*** *macro*
    - *Indicates that there are no more statements in the PSBGEN job*
    - ***PSBNAME*** *parameter specifies the name to be given to the output PSB module*

***LANG*** *parameter specifies the language in which the related application program will be written.*

**Q. IMS Processing Options**

**ANS.**

* *Indicates to IMS the type of access allowed for a sensitive segment (SENSEG)*
* *Commonly used Processing Options*
  + ***PROCOPT=G*** *means only read-only access*
  + ***PROCOPT=R*** *means read/replace access*
  + ***PROCOPT=I*** *means insert access allowed*
  + ***PROCOPT=D*** *means Read/Delete access*
  + ***PROCOPT=A*** *means all the above options present*
  + ***For GSAM  DBs PROCOPT=LS for output and GS (Get Sequential) for input***
  + ***PROCOPT=L allows a 'load' into the DB. If VSAM DB, it should be empty prior to the load***
* ***The PROCOPT given for a Sensitive segment would override the one given for the DB***
  + *Example : -*

*PCB   TYPE=DB,NAME=LDB42F,PROCOPT=G, KEYLEN=200*

*SENSEG NAME=SEGL4201, PARENT=0,PROCOPT=A*

* + *WARNING : Indiscriminate use of PROCOPTS can  lead to inexplicable results !*

**ACB & ACBGEN**

* ***ACB(Application Control Blocks)*** *: It is created by merging and expanding PSB’s and DBD’s into an IMS internal format when an application program is scheduled for execution.*
* ***ACBGEN:*** *The process of building ACB is called Block Building and is done by means of ACBGEN.*
* *IMS can build ACB’s either dynamically or it can  prebuild them using ACB maintenance utility.*
* ***ACB’s cannot be prebuilt for GSAM DBD’s.***
* ***ACB’s can be prebuild for PSB’s that reference GSAM databases.***
* ***ACB’s save instruction, execution and direct-access wait time and improves performance in application scheduling.***
* *ACB’s are maintained in* ***IMS.ACBLIB*** *library.*

**Running an application program under DL/I**

* ***Batch program does not access IMS directly***
* ***JCL invokes the DL/I ‘batch initialization module’ DFSRRC00 which loads the application program and the required DL/I modules***
* ***The program and DL/I modules execute together***
* ***Sample JCL :***

//JOBNAME JOB (ACCT),'PGMR NAME',

//             CLASS=J,

//             MSGCLASS=Z,

//             NOTIFY=&SYSUID

//JOBLIB  DD DSN=YOUR.PROGRAM.LOAD.LIBRARY,

//              DISP=SHR

//             DD DSN=YOUR.SYSTEM.RESLIB.LIBRARY,

//              DISP=SHR

//PROC EXEC PROCNAME, SYMBOLIC PARAMETERS

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//PROCNAME PROC

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* THIS PROC LOADS AN IMS VSAM DATABASE

//\* A PROGRAM 'LOAD' IS USED FOR THIS PURPOSE

//\* THE PSB USED FOR LOADING IS LOADPSB

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//LOAD    EXEC PGM=**DFSRRC00**,

//             **PARM**='DLI,LOAD,LOADPSB'

//**DFSRESLB**  DD DSN=YOUR.DFRESLIB.LIBRARY,

//             DISP=SHR

//**IMS**       DD DSN=YOUR.DBD.LIBRARY,

//             DISP=SHR

//          DD DSN=YOUR.PSB.LIBRARY,

//             DISP=SHR

//**IMSLOGR**   DD DSN=YOUR.IMSRLOG.DATASET,

//             DISP=SHR

//**IEFRDER**   DD DSN=YOUR.IEFRDER.DATASET,

//             DISP=OLD

//\* DD NAMES ARE AS SPECIFIED IN THE DATABASE

//**DATA**      DD DSN=VSAMDB.DATA.PART,DISP=SHR

//**INDEX**     DD DSN=VSAMDB.INDEX.PART,DISP=SHR

//**INPUT**     DD DSN=FILE.USED.FOR.LOADING,

//             DISP=SHR

//**DFSVSAMP**  DD DSN=IMSVS.PROCLIB(DFSVSAMP),

//             DISP=SHR

//**CPXMOPTS**  DD DSN=PARMLIB.LIBRARY(LOAD),

//            DISP=SHR

//**CPXMRPTS**  DD SYSOUT=\*

//**SYSOUT**    DD SYSOUT=\*

//**SYSPRINT**  DD SYSOUT=\*

//**SYSUDUMP**  DD SYSOUT=\*

//**IMSERR**    DD SYSOUT=\*

//**IMSPRINT**  DD SYSOUT=\*

Q. **COBOL Basics for Processing a DL/I Data Base**

**ANS.**

**ENTRY and GO BACK Statements**

* ***Format of the DL/I ENTRY Statement***

**ENTRY ‘DLITCBL’** **USING** PCB-name1

[PCB-name2...]

* *Application program is invoked under the control of the batch initialization module*
* ***DLITCBL => ‘DL/I to COBOL’ is the entry point to the program***
* *DL/I supplies the address of each PCB defined in the program’s PSB*
* ***PCBs must be defined in the Linkage Section***
* *Linkage Section definition of a PCB is called a ‘****PCB Mask’***
* *Addressability to PCBs established by listing the PCB Masks on the ENTRY statement*
* ***PCB masks should be listed on the ENTRY statement in the same sequence as they appear in your program’s PSBGEN***
* ***GO BACK Statement***
  + *When a program ends, it passes control back to the DL/I*
  + *DL/I reallocates resources and closes the data base data sets*

*Use GO BACK and not a STOP RUN statement*

**The DL/I Call**

* ***CALL statements are used to request DL/I services***
* ***Format of the DL/I call***

*Parameters you code on the CALL statement specify, among other things, the operation you want DL/I to perform*

**CALL ‘CBLTDLI’** USING DLI-function

PCB-mask

segment-io-area

[segment-search-argument(s)]

* ***CBLTDLI => ‘COBOL to DL/I’, is an interface module that is link edited with your program’s object module***

*PLITDLI, ASMTDLI are other options*

* ***The DL/I Function***
  + ***First parameter coded on any DL/I call***
  + ***Four character working storage field containing the function code***

***DL/I function codes***

**01 DLI-FUNCTIONS.**

   05 DLI-GU PIC X(4) VALUE ‘GU  ’.

   05 DLI-GHU PIC X(4) VALUE ‘GHU ’.

   05 DLI-GN PIC X(4) VALUE ‘GN  ’.

   05 DLI-GHN PIC X(4) VALUE ‘GHN ’.

   05 DLI-GNP PIC X(4) VALUE ‘GNP ’.

   05 DLI-GHNP PIC X(4) VALUE ‘GHNP’.

   05 DLI-ISRT PIC X(4) VALUE ‘ISRT’.

   05 DLI-DLET PIC X(4) VALUE ‘DLET’.

   05 DLI-REPL PIC X(4) VALUE ‘REPL’.

   05 DLI-CHKP PIC X(4) VALUE ‘CHKP’.

   05 DLI-XRST PIC X(4) VALUE ‘XRST’.

   05 DLI-PCB PIC X(4) VALUE ‘PCB ’.

* ***COBOL doesn’t allow coding literals in a CALL statement***

***Get functions***

* *First six 05-level items above*
* *Used to retrieve segments from a DL/I data base*
* ***GU***⇨ *‘get unique’ function causes DL/I to retrieve a specific segment occurrence based on field values that you specify*
* ***GN***⇨ *‘get next’ function used to retrieve segment occurrences in sequence*
* ***GNP***⇨ *‘get next within parent’ function lets you retrieve segment occurrences in sequence, but only subordinate to an established parent segment*
* *The three get function codes that contain an H are ‘****get hold functions’*** *and are used to specify an intent to update a segment after you retrieve it*
* ***GHU*** *or the ‘get hold unique’ function corresponds to GU*
* ***GHN*** *or the ‘get hold next’ function corresponds to GN*
* ***GHNP*** *or the ‘get hold next within parent’ function corresponds to GNP*

***Update functions***

* *Used to change data in the data base*
* *ISRT or the ‘insert’ function is used to add a new segment occurrence to a data base–  whether it be change an existing data base or to load a new one*
* *DLET or the ‘delete’ function is used to remove a segment from a data base*
* *REPL or the ‘replace’ function is used to replace a segment occurrence* ***Other functions***
* *Functions* ***CHKP*** *(the ‘checkpoint’ function) and XRST (the ‘restart’ function) are used in programs to take advantage of IMS’s recovery and restart features*
* *Function* ***PCB*** *is used in CICS programs*
* *Function* ***SYNC*** *is used for releasing resources that IMS has locked for the program (applicable only in a BMP)*
* *Function* ***INIT*** *allows an application to receive status codes regarding deadlock and data availability (from DB PCBs)*
* ***PCB mask***
  + *Second parameter on the DL/I call*
  + *The name of the PCB mask defined in the program’s Linkage Section*
  + *ENTRY statement establishes a correspondence between PCB masks in the Linkage Section and the PCBs within the program’s PSB*
  + ***After each DL/I call, DL/I stores a status code in the PCB mask, which the programmer can use to determine whether the call succeeded or failed***
* ***Segment I/O Area***
  + *Third parameter on the DL/I call*
  + ***Name of the working storage field into which DL/I will return retrieved data or from which it will get data for an update operation***
* ***Segment search argument***
  + *Optional parameter on the DL/I call*
  + *Identifies the segment occurrence you wish to access*
  + *Multiple SSAs on a single DL/I call*
  + *Two kinds of SSAs– unqualified and qualified*
  + ***An unqualified SSA***
    - *Supplies the name of the next segment type that you want to operate on*
    - *If you issue a GN call with an unqualified SSA, DL/I will return the next occurrence of the segment type you specify*
  + ***A qualified SSA***
    - *Combines a segment name with additional information that specifies the segment occurrence to be processed*
    - *A GU call with a qualified SSA might request a particular occurrence of a named segment type by providing a key value*

**Q. The PCB Mask**

**ANS.**

For each data base your program accesses, DL/I maintains an area of storage called the program communication block (PCB)

Masks are defined for those areas of storage in the Linkage Section of your program

01 INVENTORY-PCB-MASK.

   05 **IPCB-DBD-NAME PIC X(8).**

   05 **IPCB-SEGMENT-LEVEL PIC XX.**

   05 **IPCB-STATUS-CODE PIC XX.**

   05 **IPCB-PROC-OPTIONS PIC X(4).**

   05 FILLER PIC S9(5) COMP.

   05 **IPCB-SEGMENT-NAME PIC X(8).**

   05 **IPCB-KEY-LENGTH PIC S9(5) COMP.**

   05 **IPCB-NUMB-SENS-SEGS PIC S9(5) COMP.**

   05 **IPCB-KEY PIC X(11).**

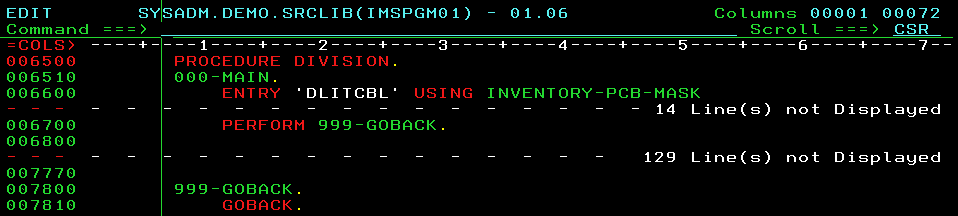
* ***Data base name***
  + *The name of the data base being processed*
* ***Segment level***
  + *Specifies the current segment level in the data base*
  + *After a successful call, DL/I stores the level of the segment just processed in this field*
* ***Status code***
  + *Contains the DL/I status code*
  + *When* ***DL/I successfully complete the processing you request in a call, it indicates that to your program by moving spaces to the status code field in the PCB***
  + ***If a call is unsuccessful or raises some condition that isn’t normal, DL/I moves some non-blank value to the status code field***
  + *It is good programming practice to evaluate the status code after you issue a DL/I call*
* ***Processing options***
  + *Indicates the processing a program is allowed to do on the data base*
* ***Segment name feedback area***
  + *The name of the segment is stored by DL/I in this field after each DL/I call.*
* ***Key length feedback area***
  + *The field DL/I uses to report the length of the concatenated key of the lowest level segment processed during the previous call*
  + *Used with the key feedback area*
* ***Number of sensitive segments***
  + *Contains the number of SENSEG macros subordinate to the PCB macro for this data base*
* ***Key feedback area***
  + *Varies in length from one PCB to another*
  + *As long as the longest possible concatenated key that can be used with the program’s view of the data base*
  + *After a data base operation, DL/I returns the concatenated key of the lowest level segment processed in this field, and it returns the key’s length in the key length feedback area*

**Q. What's the Skeleton of a COBOL-IMS Program?**

**ANS.**

To write a COBOL-IMS Program and win the battle, you must be armed with two heavy-duty tools – **ENTRY and GOBACK Statements** in COBOL**. In the IMS Environment, it’s the IMS/DB Software that is the supreme Commander-in-Chief. The IMS/DB Software will call your COBOL Program.** Your COBOL Program is a sub-program to the IMS/DB Software. The control is transferred to the COBOL Program. **ENTRY statement indicates the ENTRY-Point into the COBOL-Program, the position from where the COBOL Program begins to run. The COBOL Program runs, processes data, and generates Output Results.** As soon as the GOBACK Statement is encountered, the COBOL Program completes, the control is returned (goes) BACK to the IMS/DB Software, and the IMS/DB Software again takes over.

This is what any COBOL-IMS Program would have – the ENTRY and GOBACK Statements.



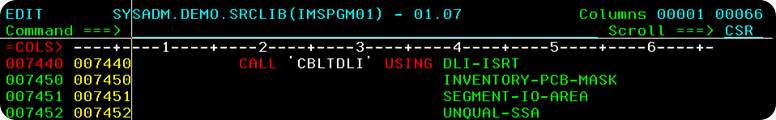
**First, IMS/DB Software loads the DBD (database structure information) and the PCB Control-Blocks into Memory. It then sends down the address of the DBD and the PCB Control Blocks to your COBOL Program**, and then runs your COBOL Program. These PCB Control-Blocks reside outside the COBOL-Program. Then how does the COBOL Program get access to them, see what's inside these Control Blocks? What you do is, you provide a definition of the PCB in the LINKAGE SECTION of the COBOL Program. The LINKAGE SECTION Definition of the PCB is called PCB-Mask. Via These masks in the COBOL Program, you can access the actual PCB's in Storage. You should list these masks on the ENTRY Statement.

The above picture shows, how I've coded the ENTRY Statement. It specifies 'DLITCBL' as its entry point. And the USING Clause lists the names of the masks in the LINKAGE SECTION.

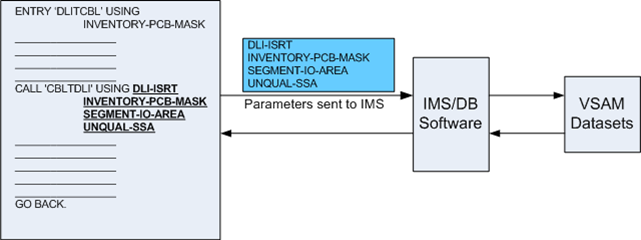
**Q. How do you issue an IMS/DB CALL?**

**ANS.**

Whenever you would like to perform any operation in IMS, say read the data, update the data etc. you issue a CALL to the IMS/DB Software. This is how you code IMS/DB CALLs.



You issue an IMS/DB CALL, by coding CALL 'CBLTDLI'. You supply additional parameters on the CALL. I shall broadly describe them here –



1)**The DLI function**

It is the first Input-Parameter to the IMS/DB Software. The DLI-Function indicates to IMS, the operation to be performed on the data, and is a 4-Bytes Code. To add new data to the Database 'ISRT'(Insert) DLI-Function is used. To update data 'REPL'(Replace) DLI-Function, to erase data, the 'DLET' (Delete) DLI-Function and so on, are used.

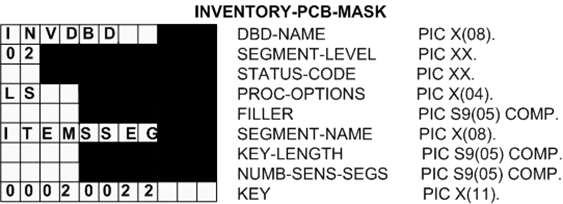
image

2)**The PCB-Mask**

The second parameter on the IMS/DB CALL is the PCB-Mask. PCB-Mask is a common shared communication-area between IMS/DB Software and the COBOL Program. IMS/DB Software returns-back(communicates) important information to the COBOL Program through the PCB Mask.

After an IMS/DB CALL Executes, how do you know, what's the result of the IMS/DB CALL? Was it successful, or did it fail? After every IMS CALL, the IMS/DB Software sets a 2-Digit Status code, to indicate the success or failure of the operation. The Status Code SPACES indicates the IMS CALL was successful. A non-Blank status code like 'GE', indicates that the IMS CALL failed due to some reason. IMS/DB Software stores the two-digit status code in the PCB-Mask.

Here's the picture of the PCB-Mask sent-back to the COBOL Program, after the IMS/DB CALL Executes. As you can see IMS/DB has stored important details in the PCB-Mask. The Status-Code Field is Blank(Spaces), this means the CALL was successful.



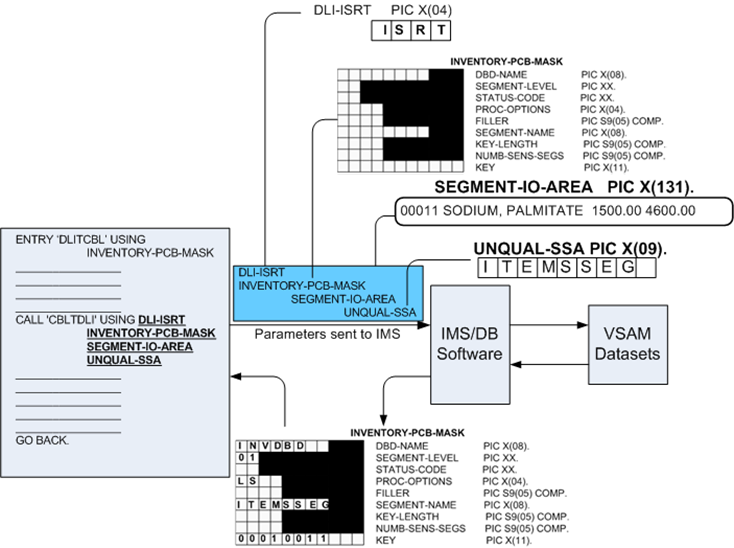
**Initially before the IMS/DB CALL, the PCB-Mask is Empty. You send the PCB-Mask to the IMS/DB Software. After the IMS/DB CALL executes, IMS/DB Software stores details such as Status-Code etc. in the PCB-Mask**.

3)**The Segment Input-Output Area**

After execution of the IMS/DB CALL, the data-result retrieved from IMS-Database needs to received and stored in some COBOL working-storage field, for processing. While inserting, the data to added to IMS-Database needs to be first stored in some COBOL-Variable, from where it will be sent to IMS. The Segment I-O Area is this Input-Output COBOL-Variable, used for either receiving the Data, when reading from the IMS Database or to send the data, when writing to IMS Database.

I have coded an IMS/DB CALL to ISRT(Insert) Items-Data into the IMS-Database.

Look, how I have stored the Item-data (00011 SODIUM PALMITATE ...) that I want to insert into the Database in the COBOL-Variable SEGMENT-IO-AREA(a 131-Byte Field).



**Note: Here observe Key = 000100011**

**This is the concatenated key from the root segment to present segment.**

**So root key=0001 present level=01 and TEMSSEG key = 00011**

4)**Optional SSA (Segment-search Arguments).**

Optionally, you may supply additional Segment Search Arguments(SSA's).

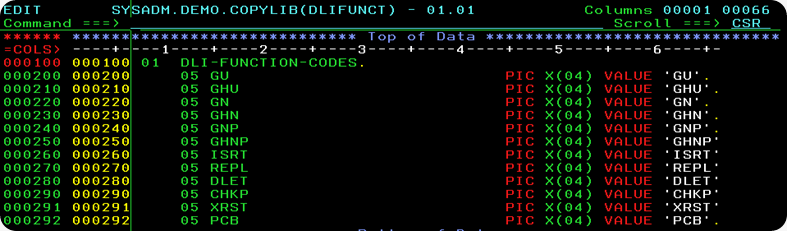
As you know, an IMS/DB Database is made of several segments e.g. VENDRSEG, ITEMSSEG and STLOCSEG, and every segment has several instances. **While fetching data from the IMS database, suppose you would like to filter-out the data. You would like to retrieve only Vendors Data(VENDRSEG). Or you want to fetch only ITEMSSEG(Items) Data. SSA's are used to search or filter data. While inserting, SSA's tell what data are you adding - Vendor Data, Items Data or Stock-Location Data.**

SSA is in a way similar to the **WHERE Clause of SQL**. In the above picture, see how I have specified the SSA('ITEMSSEG ') to indicate, that it is Items-Data, that I am inserting.

**Q. What are the different DLI Functions that you can perform on an IMS Database?**

ANS.

Every IMS/DB CALL performs an operation or function on the Database. The DLI Function is the first parameter you code on an IMS/DB CALL. Here is a list of the important DLI Functions, you would use in time to come.



Let me first describe the GU (Get Unique) and GN(Get Next) Functions. The

GU (Get Unique) Function is used for Random-Retrieval of Data. Say for example, you want to know what is the price for particular Item Code 00012. You can directly jump to this Item Segment Occurrence by doing a **GU. It is like the Standard COBOL READ Statement for Random access.**

The GN (Get Next) function is used for the Sequential-Retrieval of Data. Say for example, you would like to traverse through the entire database. You can browse the segment occurrences in sequential(Hierarchical) order one-by-one by executing **GN CALLs. It is like the Standard COBOL READ Statement for Sequential Access.**

The GNP (Get Next in Parent) function is used to retrieve segment-occurrences sequentially, but under a pre-established Parent**. First, you fix the Parent Segment Occurrence. Then you can retrieve all the segment-occurrences under it one-by-one by executing GNP CALLs**.

The other three function-codes contain 'H' or Hold. The GHU (Get Hold Unique), the GHN (Get Hold Next) and GHNP (Get Hold Next in Parent) functions are the counterparts of GN, GU and GNP CALLs, which are used to express intent of update. Suppose you are going to update a segment occurrence. First you should retrieve it using a GHN, GHU or GHNP CALL. When you retrieve segment-occurrences with the intent of updating them, you use the Hold CALLs.

The ISRT(Insert) Function, REPL(Replace) Function and DLET(Delete) Function are used to add new Segment Occurrence, update a Segment-Occurrence, or deleting Segment-Occurrence from the Database.

**Q. Why code the PCB Mask? Also, how to code the PCB Mask in COBOL?**

**ANS.**

The second parameter that you supply on the IMS/DB CALL, is the PCB-Mask. The PCB-Mask is the Linkage-Section Definition of the PCB Control-Block. The PCB-Mask has a number of uses.

1. Let's say, a Program accesses three different databases db-1 db-2 and db-3. The Program's PSB would therefore contain three PCB-Macro Statements: pcb-1, pcb-2 and pcb-3, one for each database. Suppose you write an GN (Get Next) CALL Like this -

CALL 'CBLTDLI' USING DLI-GN

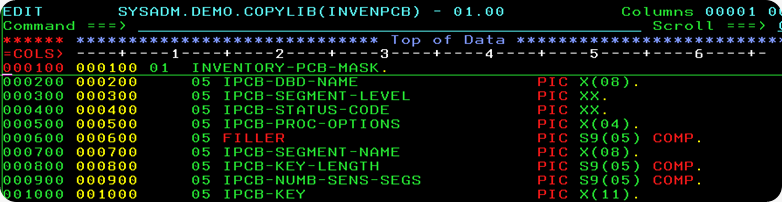
                     PCB-2

                     SEGMENT-IO-AREA

Since, you have coded PCB-2 on the GN (Get Next) CALL, it implies that the next segment-occurrence in Sequential(Hierarchical) order would be fetched from db-2. Thus, the PCB that you supply on the IMS/DB Call, tells you the database on which the CALL is executed.

2. There are other uses of the PCB-Mask: it is the link between your COBOL Program and the IMS/DB Software. The IMS/DB Software stores a 2-digit status code in the PCB-Mask, to indicate the success or failure of the IMS/DB CALL.

Take a look below, on how to write the COBOL Code for the PCB-Mask. Remember, this goes into the LINKAGE SECTION. It’s a good idea to maintain a separate Mainframe File(Copy-Book) for the PCB-Mask. I have created a Copy-book INVENPCB – that contains the COBOL Code for the Inventory Database PCB. I shall use this PCB Mask, during all IMS/DB CALLs that I execute on the Inventory Database.



Let me reiterate the fact, that after executing the IMS/DB CALL, IMS/DB Software stores important information in the PCB-Mask. These details are:

**Database Name:** The first field in the PCB-Mask is the name of the Database, that is being processed. It is an 8-Byte Alphanumeric Field. Imagine, executing the GN (Get Next) CALL on the Inventory Database. After execution of the CALL, the Field IPCB-DBD-NAME will contain the value 'INVDBD' – the name of the database(DBD).

**Segment-Level:** The next field in the PCB-Mask is the hierarchical Level-no of the Segment Occurrence Processed. This is a 2-Byte Alphanumeric Field. After execution of a CALL, stores the Level-Number of the Segment Occurrence processed. For example, if a VENDRSEG (Vendor Data) occurrence is processed, 00 is stored, if a ITEMSSEG(Items Data) occurrence is processed, 01 is stored, and if a STLOCSEG(Stock Data) occurrence is processed, 02 is stored in IPCB-SEGMENT-LEVEL Field.

**Status Code:** The third field in the PCB-Mask is the Two-Digit Status Code. After a successful call, IMS Software stores the Status-Code in this field. If the CALL is successful, IMS stores SPACES in the Status Code Field.

**Processing Options:** The fourth field in the PCB-Mask is Processing-Options, a 4-Byte Alphanumeric Field. This field indicates the privileges the program has, what processing is it allowed to do on the Database. The Field Value is populated from the Program's PSB. The next field in the PCB-Mask is a 4-Byte Filler. Defined as S9(05) COMP, it is reserved for IMS Software.

**Segment Name:** The next field in the PCB-Mask is the Segment-Name, an 8-Byte Alpha-numeric Field. After a successful CALL, IMS Stores the name of the segment-type last processed in this field. For example, upon executing GN CALL, Items Data is retrieved, then 'ITEMSSEG' value will be stored in IPCB-SEGMENT-NAME.

**Key-Length:** The next field is the Key-Length, a 4-Byte Field. After a successful ISRT or

Get CALL, IMS Software stores the Concatenated-Key of the Segment-Occurrence processed in the Key Feedback-Area. The Length of this Key, is placed in this 4-Byte Field.

**Key-Feedback Area: The** Key-Feedback Area is a variable-length Field. The Programmer must define the Key-Feedback field large enough to hold any Concatenated-Key, of the Segment processed. **The Concatenated-Key of the Segment-Occurrence, is the concatenation of the Key-Value of all the Segment-Occurrences, along the Hierarchical path, starting from the Root-Segment leading to the Segment-Processed**

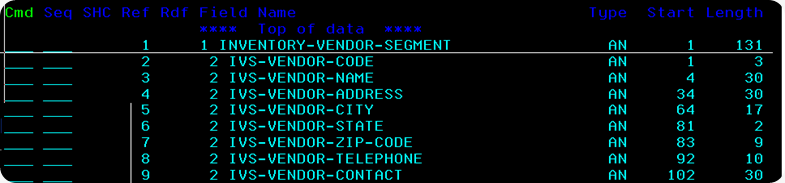
**Q. What's the importance of Segment Input-Output Area in IMS? What should be the size of the Input-Output Area?**

**ANS**

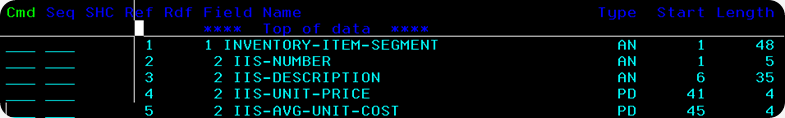
The third Parameter that you must supply on the IMS/DB CALL, is the Segment Input-Output Area. It is the COBOL-Variable or Data-Warehouse, used to hold and store the Segment-data received from IMS, or for keeping data that needs to be sent and inserted into IMS.

The size of this COBOL Variable should be large enough to accommodate the Segment-Data. How to judge(decide) the size of Input-Output Area?

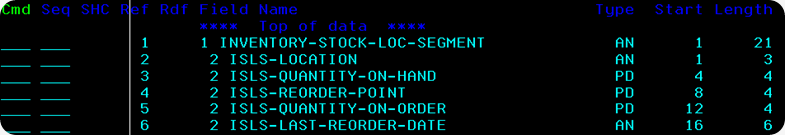
The Inventory-Database is made of three Segments – VENDRSEG (Vendor Data), ITEMSSEG (Item Data) and STLOCSEG (Stock Location Data). Here's a snapshot, of the COBOL-Definition of the VENDRSEG (Vendor Segment). It is 131-Bytes large. The data for one Vendor Segment-Occurrence is of Length 131-Bytes.



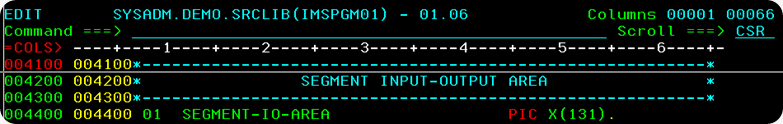
The COBOL-Definition of the ITEMSSEG (Item Segment) is shown in the Picture below. The data for one Item Segment-Occurrence is of Length 48-Bytes.



The COBOL-Definition of the STLOCSEG (Stock Location Segment) is shown in the Picture below. The data for one Stock-Location Segment-Occurrence is of Length21-Bytes.



By comparison, the Largest Segment is the Vendor-Segment. Generally, **the thumb-rule followed is, the size of the Segment I-O Area should be equal to the size of the largest Segment (in this case it is the VENDRSEG(Vendor) Segment).** Look how, I have coded the SEGMENT-IO-AREA in my COBOL-Program as a 131-Byte Alphanumeric Field.



**Q. Sample IMS – Cobol program?**

**ANS**

Here is a small program which performs GU call on a root and retrieves the root segments and displays some info in the root segment

**IDENTIFICATION DIVISION.**

PROGRAM-ID. CAAVCM04.

DATE-COMPILED.

**ENVIRONMENT DIVISION.**

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-370.

OBJECT-COMPUTER. IBM-370.

INPUT-OUTPUT SECTION.

**DATA DIVISION.**

FILE SECTION.

WORKING-STORAGE SECTION.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

**DL/I CALL FUNCTIONS**

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

01 WS-GU PIC X(04) VALUE 'GU '.

01 WS-GN PIC X(04) VALUE 'GN '.

01 WS-GNP PIC X(04) VALUE 'GNP '.

01 WS-GHU PIC X(04) VALUE 'GHU '.

01 WS-GHN PIC X(04) VALUE 'GHN '.

01 WS-GHNP PIC X(04) VALUE 'GHNP'.

01 WS-REPL PIC X(04) VALUE 'REPL'.

01 WS-ISRT PIC X(04) VALUE 'ISRT'.

01 WS-INIT PIC X(04) VALUE 'INIT'.

01 WS-ROLB PIC X(04) VALUE 'ROLB'.

01 WS-DLET PIC X(04) VALUE 'DLET'.

01 WS-CHKP PIC X(04) VALUE 'CHKP'.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

**SEGMENT LAYOUTS - USED AS IOAREAS IN CALLS**

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

**01 CA01-SEGMENT.**

10 **CA01-CS0694** PICTURE S9(4) COMPUTATIONAL VALUE +179.

10 CA01-KYCA01.

15 CA01-WS0001.

20 CA01-FI0178 PICTURE X(10) VALUE SPACE.

20 CA01-CS0189 PICTURE X(20) VALUE SPACE.

20 filler picture x(2000)

**01 CADB-AIB.**

02 CADB-AIBID PIC X(8) VALUE

02 CADB-AIBLEN PIC 9(9) COMP.

02 CADB-AIBRSFUNC PIC X(8).

02 CADB-AIBRSNM1-PCB-NAME PIC X(8).

02 CADB-AIBRSNM2 PIC X(8).

02 CADB-AIBRESV1 PIC X(8).

02 CADB-AIBOALEN PIC 9(9) COMP.

02 CADB-AIBOAUSE PIC 9(9) COMP.

02 CADB-AIBRESV2 PIC X(12).

02 CADB-AIBRETRN PIC 9(9) COMP.

02 CADB-AIBREASN PIC 9(9) COMP.

02 CADB-AIBERRXT PIC 9(9) COMP.

02 CADB-AIBRESA1-PCB-ADDR POINTER.

**LINKAGE SECTION.**

**01 CADB-PCB.**

02 CADB-PCB-DBDNAME PIC X(08).

02 CADB-PCB-SEGMENT-LEVEL PIC X(02).

02 CADB-PCB-STATUS-CODE PIC X(02).

02 CADB-PCB-PROCOPT PIC X(04).

02 CADB-PCB-RESERVED PIC S9(05)

02 CADB-PCB-SEGMENT-NAME PIC X(08).

02 CADB-PCB-KEY-LEN PIC S9(05)

02 CADB-PCB-NUMBER-OF-SENSEGS PIC S9(05)

**02 CADB-PCB-KFBAREA.**

**03 CADB-PCB-KEY OCCURS 0 TO 128**

**DEPENDING ON CADB-PCB-KEY-LEN PIC X**.

**PROCEDURE DIVISION USING CADB-PCB.**

**A-MAINLINE SECTION.**

DISPLAY 'SAMPLE COBOL IMS CODE'

DISPLAY 'CODE BEGINS'

MOVE 'CADBP01' TO CADB-AIBRSNM1-PCB-NAM

MOVE LENGTH OF CADB-AIB TO CADB-AIBLEN

MOVE LENGTH OF CA01-SEGMENT TO CADB-AIBOALEN

MOVE 'keyvalue' TO CA01-Q-BANK-NUM

MOVE 'keyvalue'' TO CA01-Q-CUST-NUM

DISPLAY 'CA01-SSA-ROOT....'

DISPLAY CA01-SSA-ROOT

CALL 'CEETDLI' USING WS-GU

CADB-AIB

CA01-SEGMENT

CA01-SSA-ROOT

MOVE CADB-AIBRETRN TO WS-AIBRETRN

MOVE CADB-AIBREASN TO WS-AIBREASN

DISPLAY 'CADB-AIBRETRN..' WS-AIBRETRN

DISPLAY 'CADB-AIBREASN..' WS-AIBREASN

DISPLAY 'CADB-PCB-STATUS-CODE..' CADB-PCB-STATUS-COD

DISPLAY 'CA01-FI0178...' CA01-FI0178

DISPLAY 'CA01-CS0189...' CA01-CS0189

DISPLAY 'CA01-LN3174...' CA01-LN3174

IF WS-AIBRETRN = spaces

DISPLAY 'found'

ELSE

DISPLAY 'not found'

END-IF

**CALL 'CEETDLI' USING WS-ROLB**

**CADB-PCB.**

DISPLAY 'CODE ENDS'

GOBACK

These are the important things to make note of

CALL 'CEETDLI' USING WS-GU

CADB-AIB

CA01-SEGMENT

CA01-SSA-ROOT

1.parameter 1 what type of call are u going to provide

2.parameter 2 your aibparameters if the cadb-AIBRETRN is 2304 the the segment which you are searching is not found.

3.the receivable segement and its layout

4.SSA root is the place where u define the segement and its key this should be same format as above.

CEETDLI DLI call is same as CBLTDLI call

**Q. What is BMP programs?**

**ANS.**

**Batch message processing programs (BMPs)**

Programs that run as batch jobs but access databases that they share with online transactions. BMPs can access message queues like MPPs for batch processing and can also access operating system files. You start BMPs using JCL. There are two types of BMPs:

**Transaction-oriented BMP**

A BMP that accesses the message queue for input. It can also access databases and operating system files. Output can be sent to databases, operating system files, or a message queue. Only one input file can be associated with the message queue.

**Batch-oriented BMP**

A BMP that does not access a message queue for input. Databases and operating system files are available for input and output processing, and message queues are available for output.

**Q. What are IMS Batch Programs?**

**ANS**

**Batch programs**

Programs that can access private databases and operating system files directly. Batch jobs do not access message queues or databases shared with an online system. In IMS documentation, these are referred to as DL/I batch jobs.

**Q. What is BTS?**

**ANS.**

**BTS**

Batch Terminal Simulator. This is an IBM® product that enables you to run IMS database and data communication programs in a z/OS® or batch environment. BTS provides a comprehensive way to check program logic, IMS program interfaces, teleprocessing activity, 3270 format control blocks, and database activity.

**Q. What is Fast Path?**

**ANS.**

**Fast path**

A type of IMS processing that involves expedited handling for certain transactions and supports special databases designed for large volumes of data with a high availability rate. There are two types of fast path databases:

**DEDB**

Data Entry Database. This is a type of IMS fast path database that contains large volumes of data with a high rate of availability. Subset pointers help manage long chains of segment occurrences. One segment type is stored near the root segment and the occurrences are in chronological order.

**MSDB**

Main Storage Database. A type of IMS fast path database that uses fixed-length root segments that reside in virtual storage for quick access. The segments can be related to a specific terminal or be defined so all terminals can access the data.

**Q. What is GSAM?**

**ANS.**

**GSAM**

Generalized sequential access method- GSAM enables MVS™ batch programs and BMPs to access a sequential OS/VS data set as a database. The database is a root-only database, and the entire root segment represents a record. **Unlike sequential OS/VS data sets, you can checkpoint and restart GSAM files just like DL/I databases**.