DL/1

OVERVIEW

The historical approach to data processing was to have individual files dedicated to each application. This led to considerable data duplication, and therefore wasted space and additional programming effort to ensure data integrity. To solve problems arising from this traditional use of datasets by application programs, the concept of a "data base" was introduced.

The database approach is to have **one** file - a database - holding interrelated data which may be accessed by many applications. Since many applications will be accessing the same data, a database management system (DBMS) is needed to control accesses and to keep the data secure.

Advantages of using Data Bases

- Application programs are independent from the physical storage and access method.

- The structure of an existing data base can be modified without changing existing application programs.

- Integration and control of common data gives data integrity.

- Data can be concurrently accessed by several users.

- Data redundancy is minimised, so the maintenance of duplicated data is avoided or at least reduced.

- As the DBMS controls the access of an application program to the data base, the access of each program can be limited to the data it needs therefore improving data security.

IBM`s Information Management System, (IMS), provides database management and can be accessed by your program (the HOST program) via Data Language 1 (DL/1), which is an example of a DATA MANIPULATION LANGUAGE. An IMS database uses a **hierarchical** data structure. For this course, we shall be using the Curriculum database (see appendix), for examples and exercises.

The hierarchy shows how one piece of data relates to many other pieces of data (a "one-to-many" relationship). Data is always accessed starting at the highest level of the hierarchy.

e.g. LECTURER number 001 on the November 1994 OFFERING of the German COURSE, is Herr Schmidt. It is possible that LECTURER number 001 on the April 1995 OFFERING of the same COURSE is someone else, such as Frau Hiedelberger. The two records have the same key value, but are uniquely defined by their **hierarchical path**.
In a relational DBMS (for example DB2), each LECTURER would be uniquely defined by one LECTURER number.

Segments

DL/1 allows both sequential and direct access to individual data "segments" via a key.

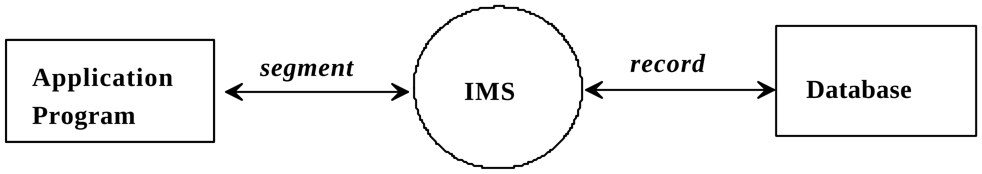
A DL/1 database consists of "records" which in turn consist of "segments". A **segment** is a group of related data "items", each "item" being defined as a "field". For example, in the Curriculum Database, the COURSE "segment" contains "fields" to describe Course Number, Title and Duration.

As mentioned earlier, an IMS database uses a hierarchical data structure. The highest level segment (in our case the COURSE segment) is called the **root** segment, and all other segments depend upon it i.e. A COURSE can exist on its own, but for an OFFERING segment to exist there must first be a COURSE segment to "attach" it to.

A **database record** consists of the root segment and all its dependants :-

e.g. One particular COURSE has no PREREQUISITEs, 2 TOPICS, 3 OFFERINGS with 5 STUDENTS booked onto the first OFFERING, and none on the other two. There is 1 LECTURER already booked for each OFFERING of the COURSE, and each STUDENT has to complete 4 EXERCISEs. There are 34 **segments** on the **database record**.

The unit of transfer between IMS and the program is a segment - only one segment can be retrieved into the work area of a program at a time, so it appears that the program is actually reading segments. In fact, IMS is retrieving **records** from the database, but only the segments required by the program are actually passed across to it, one at a time.



Child A segment which is dependent on a higher level segment e.g. TOPIC is the child of COURSE.

Parent A segment with dependants. e.g. COURSE is the parent of PREREQUISITE.

Field Individual items of data within a segment. DL/1 can identify these by their description in the DBD (Data Base Description).

Key Field The identifying field within the segment which is used to access the segment directly.

Concatenated Key A concatenated key contains a number of key fields which have been put together in order to uniquely identify a segment of data. e.g. LECTURER number 012 taking COURSE number 03 OFFERed on 1st January 1990 would have a concatenated key of 03900101012. This would uniquely identify it.

Hierarchical Sequence

Position within the database is maintained through the use of pointers. Each segment is made up of two parts, embedded pointers and data, the pointers being used to maintain position, whilst the data is returned to the program.

All segments within the database are stored in a predefined hierarchical sequence : top to bottom, front to back, left to right. Therefore, sequential DL/1 calls access the segment occurrences in that order. IMS maintains position within the database after each call, and moves forward through the database in this manner. If a new position is needed, a direct access call may be made to reposition within the database.

The types of calls used, and the order in which they are made can have dramatic effects on efficiency, it is therefore very important to understand about positioning, and this subject will be covered in more detail later.

Within the database there may be:

From 1 to n database records.
From 1 to n segments within a record.
Up to 255 segment types.
Up to 15 hierarchical levels.
Each database record must have a root segment.
The maximum segment size is 4096 characters.

The value 'n' above refers to a large system-defined limit which is not usually of concern to the application programmer.

EXERCISE 1

1. What is the parent segment of STUDENT ?

2. What are the child segments of OFFERING ?

3. What would be the concatenated key for the retrieval of the first EXERCISE segment corresponding to STUDENT 000000123 doing the COURSE whose number is 03 on 21st November 1994 ?

4. A particular course is to be held twice, the first time for a class of three students and the second time for a class of four. Each student is to complete one exercise, each class has one lecturer, students do not need any other courses as prerequisites, and the number of topics covered is three.
  a) How many segments would there be within this record ?
  b) If the segments were to be accessed sequentially, by drawing a diagram of the entire record, and labelling the segments individually, write down the order in which the segments would be accessed (the hierarchical sequence).

Control Blocks

Every program which accesses an IMS database needs to have "control blocks" defined, which describe the "logical" database structure, and the program's view of the database.

At Boots, control blocks are created by entering the appropriate information onto the Data Dictionary, and then running a batch job to generate the control block and store it on the appropriate library.

There are four types of control block :-

DBD - Data Base Description

The DBD defines the structure of the database. It contains a description of the database structure, the physical organisation, the names of the segments and their keys, and the names of any other search fields within each segment. (The description of the Curriculum database was taken from the DBD).
There will be one DBD per database, so each program which accesses a particular database, will point to the same DBD. The batch job to generate the DBD will be run by Database Support at Boots.

PCB - Program Control Block

The PCB describes the logical view of the database which the program will use. It will include the available segments, access paths and the type of access allowed (e.g. read, update, insert etc.).

e.g. A program may only need to access the COURSE and TOPIC segments. If read only access is required, this is specified on the PCB.

There may be more than one PCB per program :-

- The program may need to access more than one database.
- Since a PCB acts as a pointer to the database, having a second PCB will allow you to hold two separate positions in the database.

Each PCB is defined by the programmer, on Datamanager.

PSB - Program Specification Block

Each program has one PSB, which lists all the PCB`s available to that program. The "program" is defined on Datamanager, listing all PCB`s, then a batch job is run, to generate a PSB, using the program and PCB definitions on Datamanager.

The PSB is therefore the program's overall logical view of the database(s).

ACB - Application Control Block

The ACB combines the DBD(s) and PSB into a format which can be used by DL/1. An ACB can be created dynamically at run time for batch programs, but at Boots it is normal for it to be stored permanently on the ACB library.

Program Components

The **DL/1 Interface** program enables you to access the IMS database. It acts as a link between your program and the database.

When a DL/1 segment is required, the DL/1 interface program is invoked via a DL/1 call statement embedded in the program, in the same way as control is passed to an ordinary subroutine. The name of the "subroutine" is CBLTDLI, and the call statement must contain the following parameters :

Function Code
PCB Mask
I/O Area
SSA's (optional)

These parameters are usually coded as working storage fields rather than literals.
e.g. In COBOL you might code :

```
CALL 'CBLTDLI' USING FUNCTION, PCB-MASK, IO-AREA (,SSA1).
```

(All further examples will be shown assuming the parameters are working storage defined fields).

Function Code

This is a four byte character field which represents the service being requested from DL/1. There are nine function codes :-

GU (Get Unique)
GN (Get Next)
GNP (Get Next within Parent)
GHU (Get Hold Unique)
GHN (Get Hold Next)
GHNP (Get Hold Next within Parent)
REPL (Replace)
DLET (Delete)
ISRT (Insert)

Each one will be looked at in more detail later.

PCB Mask

This is used to pass control information between DL/1 and your program. There will be one PCB Mask for every PCB used by the program. It contains nine fields, four of which are "static" and five of which are "dynamic". The static fields are set up on entry to the program, the dynamic ones are updated after each call statement. The nine fields are as follows :-

DBD Name (static): This simply contains the name of the DBD associated with a particular application view. The program will know the name of the DBD from the PSB, which is specified in the JCL to run the program.

Segment Level Number (dynamic): Contains the level number of the lowest level segment encountered in a DL/1 call e.g. If a STUDENT segment had just been updated, the Segment Level Number would be 3.

Status Code (dynamic): This field indicates the result of the DL/1 call. The application program should **always** check the value of this field after a call is issued. The common status codes are listed later in the notes.

Processing Options (static): Contains the character codes representing the processing intent defined in the PCB.

e.g. Get (G) allows segment retrieval
Insert (I) allows insertion of segments
Replace (R) allows retrieval and update of segments
Delete (D) allows retrieval and deletion of segments
All (A) allows segments to be retrieved, inserted, updated and deleted.

Reserved for DL/1 (static): A field used by DL/1 for its own internal linkage to the application program.

Segment Name (dynamic): Holds the name of the last segment encountered by a DL/1 call. e.g. If a TOPIC has just been inserted, the Segment Name field will contain "TOPIC".

Length of Key Feedback Area (dynamic): Contains the length of the concatenated key of the lowest level segment processed e.g. If STUDENT 000023456 on COURSE 24 held on 25th July 1994 has just been retrieved, this would be 17.

Number of Sensitive Segments (static): Gives the number of different segment types which can be accessed by the PCB. In all the practical exercises on this course, the PCB has already been set up for you, to access all segments with the "A" processing option. The number in this field would therefore be 7.

Key Feedback Area (dynamic): This contains the concatenated key of the lowest level segment processed. This is a fixed length field, which is NOT cleared after each call, which is why the length of key feedback area is needed. If the above example were followed by a retrieval of course 35, the key feedback area would contain 35950725000023456, but the length of key feedback area would contain 2.

Input Output Area

You will need to code a work area into which each accessed segment can be placed. There will therefore need to be a work area (or segment layout) for each different type of segment to be accessed by your program. When you need either to update or insert new information into the database, the appropriate information can be placed in the working storage segment definition in order that it can then be passed to DL/1 via a Call statement.

Segment Search Arguments (SSA's)

SSA's can be used as parameters to the DL/1 Interface, to be more specific about which segment is to be accessed.

There are two formats of SSA - unqualified, in which only the segment type is specified, and qualified, in which both the segment type and its key field values are given. The names of the segment type and key field are taken from those defined in the DBD.

UNQUALIFIED SSA FORMAT :

Nine bytes character, eight for the segment name followed by one space.
e.g. 'COURSEsss'
(where 's' represents a space)

QUALIFIED SSA FORMAT :

Segment Name	-	eight bytes character
Left Bracket	-	one byte character
Search Field Name	-	eight bytes character - as in DBD
Relational Operator	-	two bytes character
Search Field Value	-	a working storage field or literal, corresponding to the DBD definition
Right Bracket	-	one byte character

e.g. 'COURSEss(CNUMBERs=s25)'

RELATIONAL OPERATORS :

The relational operator can be any of the following :

'= ' or 'EQ'	-	equal to
'¬=' or 'NE'	-	not equal to
'>=' or 'GE'	-	greater than or equal to
'<=' or 'LE'	-	less than or equal to
'> ' or 'GT'	-	greater than
'< ' or 'LT'	-	less than

The use of SSA's will become clearer when we look at retrieving, updating, deleting and inserting segments, in more detail.

EXERCISE 2 - CONTROL BLOCKS & PROGRAM COMPONENTS

1) When might a program have more than one PCB ?

2) Can a program have more than one DBD ? PSB ?

3) What does the ACB do ?

4) A program retrieves a STUDENT segment (number 123456789), for a student who is on the Dressmaking COURSE (86), starting on 2nd September 1994. The PSB for the program is called ECAPPSBR, and the DBD for the Curriculum database is called ECAPDBD.

Ignoring the 'Reserved for DL/1 field, what are the values in the PCB mask fields :

a) On entry to the program ?
b) After the successful retrieval of the STUDENT ?

(The PCB is set up to allow read, update insert and delete access to all segments on the database).

5) Write down the SSAs for accessing :

a) A LECTURER segment.
b) A STUDENT segment with the Student Number 987654321.

Retrieving Segments

There are two types of DL/1 retrieval calls :

a) Direct

b) Sequential

Direct Retrieval Calls

The function codes used for Direct calls are **GU** (Get Unique) and **GHU** (Get Hold Unique). The two will cause exactly the same segment to be retrieved, but GHU also specifies an update intent, i.e. the segment which has just been retrieved can now be updated or deleted.

When a direct call is made, DL/1 starts searching for the required segment at the beginning of the database, and stops when the segment has been found. Once found, the pointer will be positioned **after** the segment just retrieved.

Although direct calls can be made with unqualified SSAs or no SSAs at all, they are normally accompanied by one or more qualified SSA's to specify which segment is to be retrieved.

> e.g. To retrieve the COURSE segment for the Spanish course (number 15), using a Direct call, you would pass the following SSA :
>
> COURSEss(CNUMBERs=s15)

Any "search" field can be used in the qualification statement, but the key field is usually the most efficient, as segments are stored in ascending key sequence. (A "search" field is any field which is defined in the DBD - in the case of the Curriculum database, there are no search fields other than the key fields).

The search will begin at the start of the database, and stop once the requested COURSE has been retrieved. Because CNUMBER is a key field, DL/1 knows that all COURSE segments are stored in ascending order of CNUMBER, therefore if there is no COURSE with a number of 15, the search will end when a CNUMBER higher than 15 is reached.

A direct call with **no SSAs** will retrieve the first Root segment. In the case of the Curriculum database, a call to DL/1 with a function code of GU or GHU without SSAs, will retrieve the first COURSE segment on the database. This sort of call can be used to reposition at the start of the database.

VIEWDL/1

Viewdl/1 is a conversational CICS program designed as an immediate on-line DL/1 testing aid.

The main uses of VIEWDL/1 are :

1) Data Verification
2) Program Output checking
3) Creating test data
4) Validating PSBs.

When learning about DL/1 commands, it can also be useful to practice the use of DL/1 call statements, and to gain understanding of positioning.

Through a 3270-type display screen, you can exactly emulate DL/1 commands.

N.B. Because VIEWDL1 is an on-line conversational CICS program,it holds resources for the duration of the transaction. If the program is heavily used, the performance of the CICS system will be degraded. To avoid this, certain restrictions have been imposed on the VIEWDL1 transaction, which means that the number of users concurrently using the transaction is limited. Users may have to wait for another user to finish before they may proceed. VIEWDL1 "times out" if no processing has taken place for 10 minutes. All updates/ deletes/ inserts which have taken place will be irretrievably lost, as data is not committed until the program terminates susccessfully.

Entering VIEWDL/1

- Go into ZZCICS.
- Type VDL1 and ENTER.

The VIEWDL/1 screen will be displayed :

```
VIEWDL1 V3.1              (C) D.DONOVAN 1986              SCREEN MODE  HEX
FUNCTION      PSB         PCB 01   SSAs  0       RETURN CODE
 OFFSET       SEGMENT           KFBK

    SEG-NAME        C-C    KEYFIELD OP <------------K E Y  V A L U E--------------> X

 SSA1               *--
 SSA2               *--
 SSA3               *--
 SSA4               *--
 SSA5               *--
```

Screen Mode		Contains either 'HEX' or 'CHAR' (default is 'HEX' - can be changed by hitting PF6).
Function	(i)	To contain the DL/1 function code eg. GHU
PSB	(i)	To contain the PSB name eg. in our Curriculum database example, the PSB has been created for us, and is called 'ECAPPSBR'.
PCB	(u)	To contain the number of PCB`s within the PSB - default is 1.
SSAs	(u)	Contains the number of SSAs to be included in the DL/1 call. Up to 5 are allowed, the default is 0. If left as 0, any SSAs typed in will be ignored.
Return Code		Contains the status code and a brief description of the error.
Offset	(u)	Position within the DL/1 segment from which the DISPLAY will commence. Default is 0.
Segment		Contains the name of the lowest level segment retrieved after each DL/1 call. (Like Segment Name in the PCB mask).
KFBK		Contains the concatenated key of the segment currently being accessed. The top line is in hex, the lower is character.

i = input field, u = update field

Up to 5 SSAs can be entered - qualified or unqualified :

Seg-Name	The segment name e.g. COURSE
C-C	Any valid command code (not usually used at Boots).
Keyfield	Field name of the search key e.g. CNUMBER.
Op	Relational Operator e.g. EQ
Key Value	Contents of Search Key e.g. 26 for COURSE number 26.
X	Type an X in this column, if the key value has been entered in hex.

Summary of Key Settings

PF1	Page I/O area forward.
PF2	Page I/O area backward.
PF3	Page to start of I/O area.
PF4	Page to end of I/O area.
PF5	Update I/O area.
PF6	Change screen mode.
PF7	Create I/O area for ISRT.
CLEAR	Exit VIEWDL/1.

VIEWDL1 - Retrieving Segments

Direct Retrieval

e.g. To find the title of the COURSE numbered 36, type in the relevant fields as shown :

```
VIEWDL1 V3.1          (C) D.DONOVAN 1986        SCREEN MODE HEX    FUNCTION
 GU    PSB  ECAPPSBR    PCB 01   SSAs  1       RETURN CODE              OFFSET
       SEGMENT        KFBK

     SEG-NAME         C-C    KEYFIELD OP <------------K E Y  V A L U E--------------> X

 SSA1  COURSE         *--     CNUMBER EQ  36
 SSA2                 *--
 SSA3                 *--
 SSA4                 *--
 SSA5                 *--
```

The COURSE segment will be retrieved into the area below the SSAs.

If the segment is too large for the screen, it is possible to page forward and back :-

PF1	page forward
PF2	page back
PF3	jump to start
PF4	jump to end

Calls with Multiple SSAs

A segment in a database is uniquely defined by its **hierarchical path**, i.e. the sequence of segment occurrences, one per level, leading directly from the root segment down to a particular segment as lower level.

e.g. The hierarchical path to the LECTURER teaching on the Statistics COURSE (number 92) starting on July 12th 1994 consists of

COURSE segment 92,
OFFERING segment 940712,
and LECTURER segment (there is only one LECTURER per OFFERING, so there is no need to qualify this segment)

It might be that the same LECTURER is also teaching on other OFFERINGS of the Statistics COURSE, or maybe he teaches a Maths COURSE as well, so there could be a number of LECTURER segments on the database, with the same LNUMBER, but only one is in the hierarchical path just described.

To retrieve a segment low down in the hierarchy, the most efficient way is to code one qualified SSA **for each level** in the hierarchy. The SSAs are passed to DL/1 in sequence by the level they represent, starting with the root segment, then the second level, and so on.

Coding an SSA for each level is optional however it is the best way of retrieving the correct segment. If you just code one qualified SSA (for the segment you wish to retrieve), using a Get Unique call, the results would be unpredictable.

Any combination of SSAs can be used in a GU or GHU call, as long as only one SSA per level is specified.

VIEWDL1 - Multiple SSAs

Up to 5 SSAs can be entered in VIEWDL/1 (one per level). For example, the most efficient way to retrieve STUDENT A.Dillon (number 000000123) on the 'DL1 Games' COURSE (number 42), starting on 1st February 1991, is to have 3 qualified SSAs in the call :

```
VIEWDL1 V3.1           (C) D.DONOVAN 1986         SCREEN MODE HEX   FUNCTION
 GU    PSB  ECAPPSBR    PCB 01  SSAs  3       RETURN CODE               OFFSET
       SEGMENT        KFBK

     SEG-NAME        C-C   KEYFIELD OP <------------K E Y  V A L U E--------------> X

 SSA1  COURSE        *--    CNUMBER EQ  42
 SSA2   OFFERING     *--    STARTD     EQ  950201
 SSA3   STUDENT      *--    SNUMBER  EQ  000000123
 SSA4                *--
 SSA5                *--
```

The STUDENT segment will be retrieved into the area below the SSAs.

EXERCISE 3

1. Find the name of the LECTURER teaching on the Pottery COURSE (number 15), starting on 12th September 1995. (There is only one LECTURER per OFFERING).

2. What is the description of the first exercise to be completed by STUDENT number 98347383, on COURSE number 33, starting on 21st October 1985 ?

<u>Sequential Retrieval Calls</u>

The function codes for sequential calls are **GN**, **GHN**, **GNP** and **GHNP**. The last two will be covered later. GN and GHN will cause exactly the same segment to be retrieved, but GHN also specifies an update intent, i.e. the segment which has just been retrieved can now be updated or deleted.

Sequential calls are usually more efficient than direct calls, as the search starts from the program's <u>current</u> position in the database, rather than from the beginning. However DL/1 can only search forward.

> e.g. If the last DL/1 call retrieved a STUDENT on October's OFFERING of the French COURSE, and the next segment which the program wants is a STUDENT on May`s OFFERING of the same COURSE, a sequential call will not find the STUDENT. The solution here could be to use a GU call with no SSAs, to reposition at the start of the database, or to use a GU call qualifying exactly which segment is to be retrieved.

As stated earlier it is important to know the hierarchical sequence of the database, and to understand about positioning, and the effects of direct or sequential calls.

> e.g. If a program is required to print out the LECTURER and all the STUDENTs on every OFFERING of every COURSE, using sequential retrieval (which would be the most efficient method), the program would have to retrieve the LECTURER on each OFFERING, before the STUDENTs.

As with direct calls, SSAs are optional. A sequential call with **no SSAs**, can be used within a loop, to retrieve all the segments on a database. In this case, the Input/Output area should be large enough to hold the largest segment in the database.

Sequential calls can be made with **qualified SSA's**.

> e.g. To retrieve the OFFERING segment for the Pottery COURSE (number 15), on 12th September 1995, using a sequential call, you would pass the same SSAs as for a direct call :

```
COURSEss(CNUMBERs=s15)
OFFERING(STARTDss=s950912)
```

The difference would be that a sequential call would start from the program's current position, so this must be before the position of the segment required.

Similarly to a direct call, it is advisable to code an SSA for each level in the hierarchy. Using only one SSA for the segment you wish to retrieve, with a Get Next call would be inefficient,

> e.g. the following SSA :

```
LECTURER(LNUMBERs=s012)
```

with a Get Next function code would cause DL/1 to search from the current position through **every** OFFERING of **every** COURSE until one with LECTURER number 12 was found. .

<u>Get Next within Parent Calls</u>

These use the function codes GNP and GHNP (the latter being to specify an update intent on the segment retrieved).

Like GN and GHN calls, the search starts from the current position in the database, and is therefore usually more efficient than using GU and GHU calls. However they can also be more efficient than GN and GHN calls, as only dependant segments of a particular parent can be retrieved, so the range of DL/1's search is limited.

Establishing Parentage : To make GNP or GHNP calls, parentage must be established, otherwise the call will be rejected, and an error status code passed back to the program. Every successful GU, GN, GHU or GHN call establishes parentage on the segment retrieved.

> e.g. If COURSE 25 has been retrieved using one of the above four calls, it is established as the parent. Every TOPIC on that COURSE can now be retrieved, using a GNP call with an unqualified SSA specifying the TOPIC segment, until a status code is passed back from DL/1 to show that there are no more TOPICs within that parent COURSE.

Parentage will be lost if :

a) There is an unsuccessful retrieval call,
b) The parent segment is deleted,
c) A segment which is not a dependent of the current parent, is inserted into the database.

Like GU and GN calls, any combination of qualified and unqualified calls can be used, providing there is only one per level **below** the established parent.

<u>VIEWDL1 - Sequential Retrieval</u>

e.g. It is possible to retrieve all segments on the database, by typing GN in the *Function* field, leaving SSAs as zero, and hitting the ENTER key after each retrieval.

EXERCISE 4

1. What are the Names, Status Codes and Start Dates of all the STUDENTs enrolled on COURSE number 42 ?

Replacing Segments

For a program to be able to replace segments on the database, it must have a processing option of 'R' (Replace) or 'A' (All).

As stated earlier, a REPL call must be preceded by a retrieval of the segment to be updated, with a 'hold' intent (GHU, GHN or GHNP). If any other DL/1 call is issued using the same PCB, between the retrieval and the update, the hold intent is lost.

SSA's are not usually specified with REPL calls, as the preceding hold call identifies the segment to be replaced. This information is maintained in the PCB Mask, so an SSA is not required.

The Input/Output area associated with the REPL call will usually be the same as the one used in the hold call, with the relevant fields updated.

A REPL call does not affect database positioning or parentage. (The position will remain just after the segment replaced, which is where it was after the hold call).

Note : The key field in a segment **cannot be updated**, as position in the database is determined by the key - to 'change' a key field, the segment must be deleted, and a new segment inserted using the same Input/Output area, with the key field contents changed.

VIEWDL1 - Replacing Segments

- The segment to be replaced must first be retrieved using a HOLD call i.e. GHU, GHN or GHNP.
- Change the *Function* to REPL.
- Set the SSAs number to 0.
- Alter the Input/Output Area as required.
- **Hit PF5 to record the change in the I/O Area.**
- Hit ENTER to perform the Replace function.

EXERCISE 5

1. COURSE number 45 has been extended : update it so that it runs for 10 days.

2. The Status Code of "Connah M" appears to be incorrect. Update the database to set it to a space. The STUDENT number is 000032767, and the Start Date is 1st February 1995. He is enrolled on COURSE 42.

Inserting Segments

For a program to be able to insert segments onto an **empty** database, it must have a processing option of 'LS' (Load in ascending Sequence). To insert segments into a 'loaded' database, it must have a processing option of 'I' (Insert) or 'A' (All). (The call statement will be the same in both cases).

The function code to insert segments is 'ISRT'. There will be one or more SSAs in the call statement. The one for the segment to be inserted (i.e. the lowest level SSA) must be unqualified, the others will describe the hierarchical path into which the segment will be inserted.

e.g. To insert a STUDENT onto COURSE 42, starting on 1st August 1995, the call statement could look like this :

```
CALL 'CBLTDLI' USING 'ISRT', PCB-MASK, IO-AREA,
                                       COURSEss(CNUMBERs=s42),
                                       OFFERING(SDATEsss=s950801),
                                       STUDENT.
```

An alternative would be to position the program in the database just after the appropriate OFFERING segment (using GU with two qualified SSAs for example), then to insert the STUDENT segment using one unqualified SSA.
In both cases the STUDENT segment will be 'attached' to the OFFERING segment in SNUMBER sequence. If the segment had a non-unique key, or no key, the position would depend upon options specified in the DBD (FIRST, LAST or HERE).

After an ISRT, the program is positioned **after** the segment which has been inserted.

VIEWDL1 - Inserting Segments

- Hit PF7 to create a blank I/O area.
- Set up the SSAs required for the call - at least one unqualified SSA to specify which segment is to be inserted, preceded by any SSAs required to define the hierachical path. (ISRT will automatically appear in the *Function* field).
- Complete the I/O area in hex, character or both. (To toggle between, use PF6).
- **Hit PF5 to record the change in the I/O Area.**
- Hit ENTER to perform the Insert function.

Deleting Segments

For a program to be able to delete segments from the database, it must have a processing option of 'D' (Delete) or 'A' (All).

DLET calls must also be preceded by a Get Hold call on the segment to be deleted.

The format of the call is similar to a REPL, in that an SSA is not specified, as the segment to be deleted is identified by the previous Get Hold call.

If a parent segment is deleted, **all** of its dependants will also be deleted. An entire database record can therefore be deleted if the root segment is deleted.

Effect on Positioning : After a successful DLET call

a) If the segment deleted is not a parent segment, the program will be positioned after the deleted segment.
b) If the segment deleted is a parent segment, the program will be positioned after the last dependent segment deleted

VIEWDL1 - Deleting Segments

- The segment to be replaced must first be retrieved using a HOLD call.
- Change the *Function* to DLET.
- Set the SSAs number to 0.
- Hit ENTER to perform the Delete function.

EXERCISE 6

1.COURSE number 15 (Pottery) has 2 TOPICs wrongly set up. Update them, so that they are numbers 1 and 2 respectively. The descriptions of the TOPICs are the same, however "The Wheel" should be an optional TOPIC.

2. A new COURSE is to be set up, which has 2 TOPICs. The COURSE (number 10) will be titled "Chocolate" , and will run over 2 days. The two TOPICs (numbers 1 and 2) are described as "Thorntons" and "Cadburys", both of which are Core TOPICs. As yet, there are no dates set up for the COURSE. Insert the relevant segments onto the database.

Command Codes

Further control over DL/1 calls can be achieved using DL/1 command codes within the SSA. These commands consist of an asterisk followed by a single letter, which are placed immediately after the segment name.

e.g. 'TOPIC   *F' to retrieve the first TOPIC under a previously established parent.

At Boots, these are not generally used, although they occasionally can be useful.

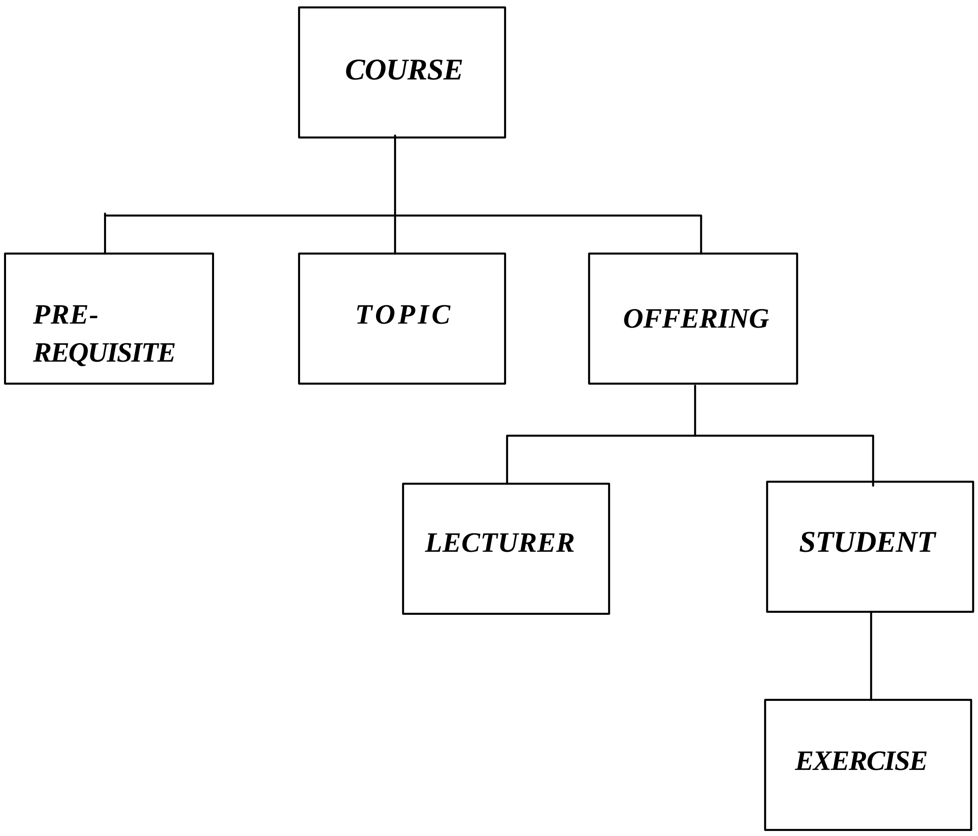
Possible command code values are :

N	-	replace after a path call, do not replace if not changed.
P	-	set parentage at this level.
F	-	get first occurrence of this type of segment under its parent.
L	-	get last occurrence.
D	-	send segment to/from I/O area (needs option P in the PSBGEN).
Q	-	enqueue the segment.
C	-	use concatenated key as qualification.
U	-	hold on to the position of this segment.
V	-	hold on to the chain of segments down to this one.

Appendix

Logical Database Structure

Example : The Curriculum Database used for the Training Exercises



Segment Name	Field	DBD Name	Start	Length
COURSE	Course Number	CNUMBER	1	2
	Title		3	10
	Duration		13	2
PREREQ	Number	PNUMBER	1	2
	Useful/Mandatory		3	1
	Alternative		4	2
TOPIC	Number	TNUMBER	1	3
	Description		4	20
	Core/Optional		24	1
OFFERING	Start Date	STARTD	1	6
	Location		7	10
LECTURER	Number	LNUMBER	1	3
	Name		4	10
	Qualification		14	10
	Spare		24	10
STUDENT	Number	SNUMBER	1	9
	Enrolment Status		10	1
	Name		11	10
EXERCISE	Description		1	10
	Grade		11	1

The information has been taken from the database description (DBD), which includes the names of all segments which form the database, the length of each segment, name of key field, name of any search fields and the segment parent (this will define the hierarchical sequence).

In the case of the Curriculum database, all the DBD names listed above are key fields (notice not every segment has a key field defined). All the field definitions above are zoned numeric or alphanumeric (there is no packed or binary format data on this particular database). The fields which are not defined in the DBD cannot be used as search fields.

Status Codes

The Status Code is a field in the PCB Mask, which is updated after every call to DL/1, and should always be checked to see if the call has been successful.

A successful call returns a Status Code of spaces.

The following are the most common codes returned from DL/1 :

'GE'	-	segment not found
'GB'	-	end of database
'II'	-	segment already exists (ISRT)
'DJ'	-	segment has not previously been held (REPL or DLET)
'AC'	-	hierarchical error in SSAs
'AJ'	-	SSA qualification format invalid
'AK'	-	invalid SSA field name
'GP'	-	parentage not established

Summary of VIEWDL1 Key Settings

PF1	Page I/O area forward.
PF2	Page I/O area backward.
PF3	Page to start of I/O area.
PF4	Page to end of I/O area.
PF5	Update I/O area.
PF6	Change screen mode.
PF7	Create I/O area for ISRT.
CLEAR	Exit VIEWDL/1.