TUTORIAL OVERVIEW

This tutorial provides the SAS® user who needs to access data stored in DL/I data bases with the fundamental concepts and techniques necessary to write a SAS/IMS-DL/I® DATA step. The session includes an introduction to DL/I programming concepts and an examination of SAS/IMS-DL/I® statement extensions. Sample DL/I data bases and SAS/IMS-DL/I® programs illustrate these concepts and programming techniques. The material focuses on retrieving data from DL/I data bases. Due to time restrictions, data base update considerations remain outside the scope of this presentation.

Since the tutorial is geared to experienced SAS users who know little or nothing about DL/I, much of the discussion deals with basic data base concepts and the components of DL/I. While it is impossible to teach you everything you need to know about DL/I in such a short session, we aim to provide you with a basic DL/I vocabulary and a foundation of concepts on which you can build. After this session you will not have all the answers, but you should know what questions to ask.

DL/I CONCEPTS

What Is DL/I?

DL/I stands for Data Language/1 and is IBM’s data base manager that forms the core of two IBM products: IMS/VS and DL/I DOS/VS. IMS/VS is IBM’s data base management system for OS environments and consists of two separate products: IMS/VS DB and IMS/VS DC. IMS/VS DB represents the data base manager DL/I and can be used alone for managing DL/I data bases in batch mode. IMS/VS DC adds data communications facilities for on-line access and control of DL/I data bases, and is only run in conjunction with IMS/VS DB. While IMS/VS DC is commonly used with IMS/VS DB, CICS/VS also provides on-line facilities for accessing DL/I data bases under OS.

DL/I DOS/VS represents the data base manager for DOS environments. IBM does not provide a facility for DOS parallel to IMS/VS DC. Under DOS, CICS/VS is used for on-line access to DL/I data bases.

What Is a Data Base?

DL/I is IBM's data base manager, but what is a data base? This term is much abused. 'Data base' is sometimes used to refer to any file or set of files in order to project a state-of-the-art image. We can, however, define the term more precisely.

A data base is a collection of interrelated data stored in one place for access by multiple users. The data are collected together and are in one place to provide control and eliminate duplication. In addition, the physical structure of a data base is transparent to the users. Only the data base system must know the physical organization of the data, since users and applications access this data through the data base manager. Then, if the data base changes physically, the applications are not necessarily affected. Often just the definition of the data base to the data base manager must change. Finally, a data base is organized according to a particular data model, which is used to characterize the data base system. Data base models include relational, network, and hierarchical types. DL/I uses a hierarchical data model.

SAS Data Sets As Data Bases

According to our definition, SAS data sets are data bases. A SAS data set can contain interrelated data for easy processing by various SAS procedures. The physical organization of a SAS data set is totally transparent to the users. You select the data needed simply by naming the variables. And while users do not need to know the details of the data set structure, SAS data sets use a rectangular, self-describing data model.

A Sample Data Base

A sample data base illustrates the data base concepts discussed. Let's consider a sample application environment and examine how the data might be collected into a data base. This environment provides the context for all the examples in this tutorial as well as in the SAS/IMS-DL/I User's Guide, 1984 Edition.

Information regarding customer accounts in a banking environment includes three main groups of data: customer data, account data and transaction data. The bank has customers; the customers hold accounts and initiate transactions against the accounts. While the bank has...
different application programs to maintain and report on the data, the data are interrelated.

Consider how you might store this interrelated data in a SAS data set. Assuming one observation per customer, how many accounts would you allow per customer, and how many transactions per account? It would be necessary to set an arbitrary limit for these occurrences. This limit might not always be adequate, and yet in many cases would exceed a given customer's requirements, resulting in much wasted space. How might you organize the data more efficiently?

In order not to waste space, you could set up three different SAS data sets -- one for customers, one for accounts and one for transactions. When the number of occurrences of the data varies unpredictably, store that data in a SAS data set. Assuming one application program to maintain and report on the data, the data are interrelated. A segment also represents the structure of the data varies unpredictably, store that data in a SAS data set. However, you would then need to tie the data in the different data sets together by duplicating identifier variables such as CUST_SSN and ACCT_NUM. In addition, the organization of the data is no longer completely transparent, because you need to know which data sets contain the information needed. Accessing the data becomes more complex also, when you need to perform data set merges. Thus, while space is used more effectively, other disadvantages arise.

**DL/I Data Bases and Terminology**

With DL/I you can easily store all customer accounts data in one data base. Figure 12 represents the hierarchy of the ACCOUNT data base. A hierarchy is a pyramid-like structure of SEGMENTS. Data items that tend to be used together are stored in the same segment. A segment contains one or more fields of data and represents a unit of organization between a field and a record, a variable and an observation. A segment also represents the basic unit of transfer between DL/I and an application program.

Figure 12 shows the seven different SEGMENT TYPES in the ACCOUNT data base: at the top level, the CHECKACCT and SAVEACCT segments with the account data; on the bottom level, the CHECKCRDT, CHECKDEBT, SAVERCDT and SAVERDEBT segments with the transaction data. It is this segmentation that allows DL/I to handle variable occurrences of data without wasting space. If a particular customer, for example, has no savings account at the bank, no space is allocated for possible savings account data for the customer. The SAVEACCT and associated segments simply would not exist.

Relationships among the segments define the hierarchy of a DL/I data base. The primary relationship is that of PARENT and CHILD segments. In Figure 12, CUSTOMER is the parent of CHECKACCT and SAVEACCT; CHECKACCT is in turn the parent of CHECKCRDT and CHECKDEBT. Any given segment can be both the parent of one segment and the child of another -- except the ROOT segment. The one segment type at the top level of the hierarchy represents the root segment, and it is never a child. In fact, all the other segments are DEPENDENTS of the root segment, just as any child segment is a dependent of its parent. In the ACCOUNT data base, the CHECKACCT and SAVEACCT segments are also SIBLINGS, segments of different types under a single parent.

A DL/I data base is a series of DATA BASE RECORDS, each composed of a root segment and its dependents. The order of the data base records depends on the DL/I access method used. The sequential access methods store the data base records in key sequence; the direct access methods store them randomly. No matter what the DL/I access method, however, segments within a data base record are stored, and retrieved when requested one after the other, in HIERARCHICAL PROCESSING SEQUENCE.

Figure 15 illustrates a particular data base record from the ACCOUNT data base and the hierarchical processing sequence. Customer Jane Smith holds two checking accounts at the bank. The two CHECKACCT segments that contain information for these accounts represent two different SEGMENT OCCURRENCES of the CHECKACCT segment type and since they have the same parent, they are called TWINS. The numbers in the lower right corners of the segments indicate the hierarchical processing sequence. Starting with the root, you move first from top-to-bottom along a segment path. Then at the bottom level, you move from front-to-back processing twins. After twins, process siblings, moving from left-to-right. Movement through a data base proceeds in this fashion: always first to a lower level to process a segment's children before moving back through a twin chain, or to the right to process siblings and their children.

While DL/I processes all segments of a data base record in hierarchical sequence, the segment SENSITIVITY of certain programs may prevent DL/I from returning segments below a specific level. Segment sensitivity refers to the relationship of an application program to a data base. Application programs must be defined to DL/I in terms of the data bases they access. While a program may access a particular data base, it may only be sensitive to certain segment types in that data base. A given program, for example, may only need to report on what accounts customers have. The program is then defined as sensitive only to the CUSTOMER, CHECKACCT and SAVEACCT segments. DL/I provides sensitivity as a security feature.

**DL/I Segment Retrieval**

While the SAS/IMS-DL/I product supports DL/I data base update calls, as well as message queue and system service calls, this tutorial focuses on DL/I segment retrieval. Retrieval calls are perhaps the most complex, because so many factors are involved. Stricter rules govern the
other call types. In order to retrieve data from DL/I data bases, it is necessary to understand the DL/I hierarchical processing sequence and the concept of sensitivity. However, other factors also affect the results of a DL/I retrieval request. The particular call function and the search criteria specified, as well as the position in the hierarchy after the prior call, determine the results of a DL/I retrieval call.

DL/I provides three different call functions for retrieval calls: Get-Unique ('GU'), Get-Next ('GN') and Get-Next-within-Parent (GNF). Get-Unique can move forward or backward through the data base. Get-Next can only move forward in hierarchical processing sequence, and Get-Next-within-Parent only moves forward through the dependents of the established parent in hierarchical processing sequence.

Any of these calls may be qualified by supplying specific search criteria in Segment Search Arguments (SSAs). When no SSAs are specified, the calls are unqualified. An unqualified Get-Unique call always returns the first segment in the data base. If you issue eleven unqualified Get-Unique calls against the data base segments in Figure 15, DL/I returns segment #1 each time. If you issue eleven unqualified Get-Next calls one after the other, and your program is sensitive to all segments, DL/I returns the eleven segments in hierarchical processing sequence as shown. After retrieving segment #1 to establish parentage, ten unqualified Get-Next-within-Parent calls return segments #2 through #10 again only if your program is sensitive to all segments. An eleventh unqualified Get-Next-within-Parent results in a segment-not-found condition, because no more segments exist under that parent.

Segment search arguments can specify just a segment type, or a segment type with a particular search field value or range of values. After you retrieve segment #1, a Get-Next for a CHCKCRDT segment returns segment #5; a Get-Next for a CHCKCRDT segment with CRDTDATE= 02/15/84 returns segment #5. Refer to Figure 16 for examples of segment search arguments. After you retrieve segment #5, a Get-Next that specifies the SSA shown in Figure 16 for the customer with social security number 123-45-6789 results in a segment-not-found condition, because the Get-Next call only moves forward in the data base from the position after the prior call. Issue a Get-Unique instead to move backward to retrieve segment #1.

Figure 17 summarizes these various factors that affect the results of a DL/I retrieval call.

DL/I Control Blocks

With DL/I you cannot just start using a data base by allocating space for it and assigning it a name, as you can with standard files and SAS data sets. You must first define the data base and your programs to DL/I by setting-up DL/I control blocks. The DBD (Data Base Description) defines the physical characteristics of a data base to DL/I. You code the appropriate Assembler macros and process them by running the DL/I DBDGEN utility to create a DBBLIB member that DL/I will use when you access the data base. The PSB (Program Specification Block) defines the application program to DL/I. As for the Data Base Description, you code Assembler macros and create a PSBLIB member by running the PSBGEN utility. A PSB is composed of PCBs (Program Communication Blocks), each of which identifies a DL/I resource for program use. A data base PCB describes the program view of the data base. In other words, it specifies which segments of the data base are sensitive for the application program that the PSB defines.

Figure 19 illustrates the relationship between physical data bases, DL/I control blocks, and application programs. A DBD defines one data base, and a PSB defines one application. However, a PSB can contain multiple PCBs for the various data bases the program might access. Figure 20 represents the PSB for the customer accounts ACCUPDT application. This PSB contains two data base PCBs -- one for the ACCOUNT data base and one for the WIRETRN transaction data base. The SENSEG statements in each PCB specify the segments in the data base to which the program is sensitive. Since the PSBGEN statement specifies 'CMPAT=YES', another PCB, the IO PCB, is generated as the first PCB in this PSB. Generation of the IO PCB ensures compatibility between batch and on-line environments. While this PCB is not needed for DL/I data base retrieval functions, you need to recognize that it exists as the first PCB in the PSB when 'CMPAT=YES' is specified.

It is important to understand the program specification block, because it provides the link between the DL/I program and the data base descriptions that describe the data bases the program accesses. You must always specify a PSB when executing a program to access DL/I data bases. Thus, in order to access DL/I data bases using the SAS/IMS-DL/I interface, you must also specify a PSB. While you do not need to generate special SAS/IMS-DL/I PCBs, you must know which PCBs you provide access to the DL/I resources you need and which PCBs within the PSBs to reference.

Data Bases and Data Base Management Systems

Before proceeding to our examination of the SAS/IMS-DL/I interface and program examples, let's review our discussion of DL/I and data base concepts. A data base is a collection of interrelated data for access by multiple users. The users do not need to know the physical organization of the data because they only access it through the data base system. The data is organized according to a particular data model. Both SAS data sets and DL/I data bases fit this definition. DL/I, however, provides other features that the SAS System does not, such as keyed access to data, concurrent access to data, logging, backup and recovery facilities. While
the SAS System provides the advantages of a data base approach, it is not a full-function data base management system like DL/I. With the SAS/IMS-DL/I interface you can take advantage of the desirable features of both systems.

A SIMPLE SAS/IMS-DL/I PROGRAM

Figure 22 represents a simple SAS/IMS-DL/I program, a streamlined version of the CUSTLIST sample program from the SAS/IMS-DL/I User's Guide, 1984 Edition. This CUSTLIST program reads customer information from the ACCOUNT data base and prints a sorted list of customers with phone numbers. Note that the only difference between this SAS/IMS-DL/I program and a standard SAS program is the DLI parameter on the INFILE statement. This positional parameter indicates to the SAS supervisor that the external file to access is a DL/I data base.

The program is simple, though not quite as simple as it appears. Several options may be specified on the DL/I INFILE statement. In this case they are omitted, because the defaults are appropriate. A later section examines these defaults.

As mentioned earlier, a program specification block must be referenced when executing a program to access DL/I data bases. This example refers to the CUSREAD PSB; the PSB name precedes the DLI parameter on the DL/I INFILE statement. Figure 23 contains the CUSREAD PSB. Note that this PSB contains only one program communication block (PCB), the one for the ACCOUNT data base. Since no 'CMPAT=YES' appears on the PSBGEN statement, no IO PCB will be generated as the first PCB. The ACCOUNT data base PCB is the first and only PCB in the PSB and the one that DL/I will use when issuing the calls requested by the program. Since this PCB specifies only the CUSTOMER segment as sensitive, DL/I will return only CUSTOMER segments to the program. Although the program retrieves the segments one after the other in sequential fashion, they are not returned in key sequence, because the ACCOUNT data base uses a direct rather than a sequential DL/I access method. Therefore, PROC SORT is executed to sort the data before printing.

Figure 24-25 show the output from this sample program. The SAS log in Figure 24 contains two NOTE messages that do not appear in standard SAS program logs. The first NOTE after the DATA step statements indicates that the program involves DL/I access and specifies the PSB referenced. The following NOTE, issued only when a DL/I data base is processed in sequential fashion, verifies that the data base was processed to the end when DL/I returned its end-of-data-base condition, a 'GB' status code. Figure 25 shows the expected PROC PRINT output.

THE DL/I INFILE EXTENSIONS

In the CUSTLIST sample program above, the basic form of the DL/I INFILE statement was used, and default values were assumed for the other parameters. Use of the basic form implies certain conditions. Figure 27 outlines these conditions. The program can only retrieve segments one after the other in sequential fashion, because the DL/I call function defaults to Get-Next with no SSAs. In addition, for the INPUT statements to specify correct segment formats when segments are simply returned one after the other, either the segment formats must always be the same or the program must be able to determine the segment format before reading the entire segment. Also, since the default size of the buffer used for segments returned from DL/I is 1000 bytes, retrieved segments must not exceed this length. Finally, the interface specifies the first PCB in the PSB as the default for DL/I to use in communicating call information.

When these conditions do not reflect your programming requirements, use the appropriate DL/I INFILE extensions. Figure 28 lists the most commonly used extension options; they are all keyword parameters. On the INFILE statement, assign variable names for the options needed. (Exceptions are the LRECL= and PCBNO= options, for which you simply specify numeric values.) Then, before issuing an INPUT statement, assign appropriate values to these variables. For clarity, you can categorize the various options by function. The first six represent information you can pass to DL/I; the last two represent information DL/I can return to your program.

Of the six options that pass data to DL/I, three of them specify which PCB in the PSB to use. PCBNO= allows you to bypass PCBs not relevant to your program. Then, if a non-blank value is assigned to the DBNAME= variable, the appropriate PCB is selected by matching on DBD name. Otherwise, the PCB= variable value is used as an index to the appropriate PCB. If none of these are specified, the default is the first PCB in the PSB.

To issue calls other than Get-Next, assign the appropriate call functions to the CALL= variable. To qualify calls, assign correctly formatted segment search argument values to the SSA= variable or variables. Since you can specify multiple SSAs on a DL/I call, multiple SSA variables are allowed. If you retrieve any segments or segment paths that exceed 1000 bytes in length, assign the appropriate value to the LRECL= parameter; otherwise, results are unpredictable.

You might also want to check information returned from DL/I before continuing processing. Specify SEGMENT= to request that the interface return to your program the name of the segment retrieved. When various segment types can be returned, issue the INPUT with a trailing .
After each call, the DL/I status code is checked. A 'GB' status signals the end of the data base, and the program terminates. A 'GE' indicates that no segment meeting the search criteria was found. In this case, the DELETE statement causes an immediate return to the top of the DATA step and thus a repetition of the call. On this next call, DL/I returns the 'GB' status, and the program terminates. Since this program prints the report as part of the DATA step, the program is complete when the DATA step reaches the end of the data base.

TUTORIAL SUMMARY

These sample programs appear in full, along with a range of other examples, in the SAS/IMS-DL/I User’s Guide, 1984 Edition. You may want to review these programs in greater detail before attempting to write SAS/IMS-DL/I programs of your own. Also, you will need to gather information pertaining to your data bases and how you may access them. Do you know what questions to ask? Figure 32 summarizes key points covered in the tutorial and provides the beginnings of a checklist of information you need to collect.

SAS and SAS/IMS-DL/I are registered trademarks of SAS Institute, Inc., Cary, N.C., USA.
A SAS/IMS-DL/I SAMPLE APPLICATION
ACCOUNT DATA BASE

Figure 12

CALL FUNCTIONS
GU
GN
GNP

SEGMENT SEARCH ARGUMENTS
CUSTOMER
CUSTOMER(SSNUMBER = 123-45-6789)

RESULTS DEPEND ON
• Call function specified
• SSAs specified
• Segment sensitivity of the program
• Position in the data base after the prior call

Figure 16

Figure 17
A SAS/IMS-DL/I SAMPLE APPLICATION
ACCUPDT DATA BASE UPDATE
PROGRAM SPECIFICATION BLOCK

PCB       TYPE=DB, NAME=ACCOUNT, PROCOPT=A
SENSEG    NAME=CUSTOMER
SENSEG    NAME=CHCKACCT
SENSEG    NAME=CHCKCRDT
SENSEG    NAME=CHCKDEBT
SENSEG    NAME=SANEACCT
SENSEG    NAME=SANEDEBT
PCB       TYPE=DB, NAME=WIRETRN, PROCOPT=A
SENSEG    NAME=WIRETRAN
PSBGEN    NAME=ACCUPDT, CMPAT=YES
END

A SAS/IMS-DL/I SAMPLE APPLICATION
CUSTLIST

DATA CUSTLIST;
INFIL CUSREAD DL1;
INPUT
@1 CUST_SSN  $CHAR11.
@12 NAME  $CHAR40.
@172 H_PHONE  $CHAR12.
@184 O_PHONE  $CHAR12;
IF _ERROR_ THEN ABORT;
PROC SORT DATA=CUSTLIST; BY NAME;
PROC PRINT;
VAR H_PHONE O_PHONE;
ID NAME;
TITLE 'CUSTOMER PHONE LIST';

Figure 19

Figure 20

Figure 22

Figure 23
**SAS DL/I INFFILE STATEMENT**

**ASSUMPTIONS AND DEFAULTS**

- Sequential processing
- Read-only
- Segment types known, or format always the same
- Segments retrieved not longer than 1000 bytes
- PCB needed is first in the PSB

**SAS DL/I INFFILE STATEMENTS**

**Figure 24**

**CUSTOMER PHONE LIST**

<table>
<thead>
<tr>
<th>NAME</th>
<th>H_PHONE</th>
<th>O_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnhardt, Pamela</td>
<td>803-345-0346</td>
<td>803-355-2543</td>
</tr>
<tr>
<td>Booker, April</td>
<td>803-657-1346</td>
<td>803-657-1346</td>
</tr>
<tr>
<td>Booker, Ralph</td>
<td>803-657-1346</td>
<td>803-657-1346</td>
</tr>
<tr>
<td>Jones, J.M.</td>
<td>803-657-7636</td>
<td>803-657-7636</td>
</tr>
<tr>
<td>Jones, Roger</td>
<td>803-657-5656</td>
<td>803-657-5656</td>
</tr>
<tr>
<td>Little, Nancy</td>
<td>803-657-2566</td>
<td>803-657-2566</td>
</tr>
<tr>
<td>Smith, James</td>
<td>803-657-7437</td>
<td>803-657-7437</td>
</tr>
<tr>
<td>Stoppers, Mary</td>
<td>803-657-1687</td>
<td>803-657-1687</td>
</tr>
<tr>
<td>Walls, Hooper</td>
<td>803-657-2098</td>
<td>803-657-2098</td>
</tr>
<tr>
<td>Windsor, Jonathan</td>
<td>803-657-7330</td>
<td>803-657-7330</td>
</tr>
</tbody>
</table>

**Figure 25**

**SAS DL/I INFFILE STATEMENTS**

**Figure 27**

**SAS DL/I INFFILE EXTENSIONS**

**Figure 28**

INFFILE psbname DLI

- [CALL = variable]
- SSA = (variable1, ...)
- LRECL = number
- PCBNO = number
- DBNAME = variable
- PCB = numeric variable
- STATUS = variable
- SEGMENT = variable
CODING TECHNIQUES

MACRO FACILITY FOR SEGMENT DESCRIPTIONS

allocate MACRO data set in job control

%INCLUDE MACRO(CUSTOMER);
DATA CUSTLIST;
INFILE CUSREAD DLJ;
INPUT %CUSTOMER;

Figure 29

A SAS/IMS-DL/I SAMPLE PROGRAM

TRANREAD

%INCLUDE SOURCE(ACCTTRAN);
DATA _NULL_; RETAIN SSA1 'CHKACC'T'D';
SSA2 'CHKDEBT(DATE= 03/28/83)' DB 'ACCOUNT';
INFILE TRANREAD DLJ SSA1(SSA1,SSA2) STATUS=ST
DBNAME=DB;
INPUT %ACCITRAN;
IF _ERROR_ THEN DO;
_ERROR_=0;
IF ST='G' THEN STOP;
IF ST='G' THEN DELETE;
PUT _ALL_;
ABORT 888 ABEND;
END;
FILE TRANREPT HEADER=NEWPAGE NOTITLES;
PUT @10 ACCTNUM @30 AMOUNT DOLLAR13.2
@45 TRAN_TIME TIME8. @55 DESCRIPT;
RETURN;
NEWPAGE:
PUT / 'CHECKING ACCOUNT DEBITS FOR 03/28/83
ACCOUNT NUMBER' @33 'AMOUNT' @47 'TIME'
@65 'DESCRIPTION';

Figure 31

TUTORIAL SUMMARY

DL/I INFORMATION CHECKLIST

◊ What PSBs to use; PSB names?
◊ What databases to access; DBD names?
◊ Which PCBs in the PSBs refer to data bases needed?
◊ Need to allow for IO PCB as 1st PCB?
◊ Appropriate segment sensitivity in the PCB/PSB for data needed?
◊ Names and layouts of segments to retrieve?
◊ Any segments/segment paths to retrieve longer than 1000 bytes?
◊ Sequential processing: Is the database sequentially organized?
◊ Random processing: Search field names and lengths defined in the DBD?

Figure 32