CHAPTER 2

Understanding IMS-DL/I Essentials

Introduction

This chapter introduces SAS System users to IMS-DL/I, a hierarchical database management system by IBM. It focuses on the terms and concepts that will help you use the SAS/ACCESS interface to IMS-DL/I. It includes descriptions of the following:

- hierarchical database structure and elements
- IMS-DL/I databases
- physical databases
- the elements of DL/I calls
- execution modes
- resource sharing.

IMS-DL/I: A Hierarchical Database

An *IMS-DL/I database* is a large, centralized collection of information comprising one or more physical files that can be accessed by the SAS/ACCESS interface to IMS-DL/I. An IMS-DL/I database is a *hierarchical database:* one where information is structured in *records* that are subdivided into a hierarchy of related *segments.*

You can think of a record as a root segment and all of its dependent segments. Segments are further subdivided into *fields.* The data in any record relate to one entity. Ideally, information in the database records is subdivided into segments and fields on some logical basis, either by the inherent structure of the data or by consideration of the uses to which the data will be put.

The term *hierarchical* implies that there are levels of data. You can think of a hierarchical database as one that starts at the top with general information about the item, individual, or case. As you progress from level to level down through the hierarchy, more and more information related to the general information at the top level is given. Each level in the hierarchy has one or more segments.

IMS-DL/I Databases

In some ways the structures of IMS-DL/I databases and tabular files (such as SAS data sets) are comparable, but in other ways they differ. For example, database fields and data set variables are similar, and database records are like data set observations because both contain data about one entity. At the same time, however, database records differ from data set observations because subsets of records can be accessed while you cannot access a subset of a data set observation. The observation is stored and accessed as a unit.

A tabular file has nothing comparable to a segment. The concept of data segments is one of the things that makes a hierarchical database different from a SAS data set and other tabular files.

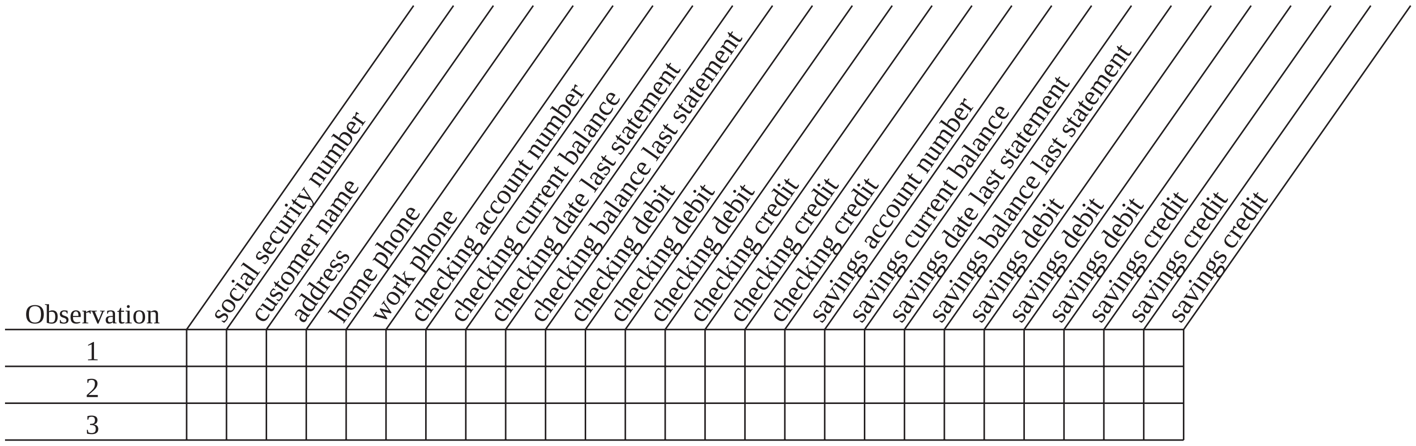
Consider banking data as an example. Customer information maintained by a bank might include the following:

name	checking account debits
Social Security number	checking account credits
address	savings account number
home phone	savings account balance
work phone	savings account date of last statement
checking account number	

checking account balance	savings account balance at last statement
checking account date of last statement	savings account debits
checking account balance at last statement	savings account credits

If this information is stored in a tabular file, each item of information is a *variable,* and all of the variables for any given person comprise one observation. You can visualize the layout of banking data in a tabular file as shown in Figure 2.1 on page 13.

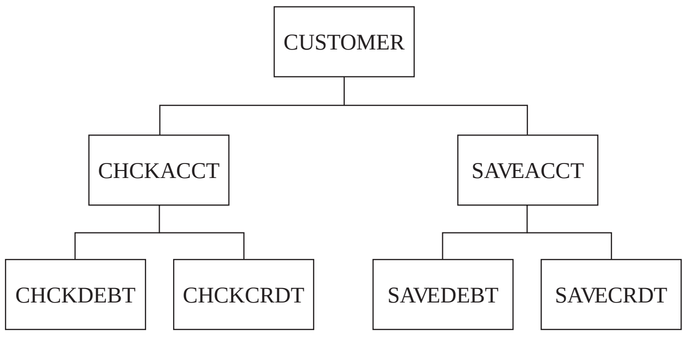
Figure 2.1 Tabular File Structure



The rows in the table represent observations (customers), and the columns represent variables. The structure of the file is such that a maximum number of variables for debits and credits must be defined when the file is created. In Figure 2.1 on page 13 there are variables for up to only three debits and three credits per customer, which presents a problem if a customer has more than three debit or credit transactions.

The same data can also be stored in an IMS-DL/I database but would be structured very differently. For example, Figure 2.2 on page 13 shows one way the banking information could be structured in IMS-DL/I. The sample database, called ACCTDBD, is used in this book and described in “About the Example Data in This Book” on page 8 in Chapter 1, "Introducing the SAS/ACCESS Interface to IMS-DL/I."

Figure 2.2 Hierarchical File Structure



Each block in the figure represents a *segment type*, which is a grouping of related *fields* of data. There are three levels in the ACCTDBD database hierarchy and seven segment types. For each database record, the top or first level has only one segment, called the *root segment*. The root segment in the ACCTDBD database is called CUSTOMER; it contains fields with these data: Social Security number, customer name, address, city, state, country, ZIP code, home phone, and work phone. The segments under the root segment are *dependent segments* called CHCKACCT, CHCKDEBT, CHCKCRDT, SAVEACCT, SAVEDEBT, and SAVECRDT. Each of the dependent segments contains fields of data, as shown in the following table.

Dependent Segment	Fields
CHCKACCT	checking account number, current balance, last statement date, last statement balance
CHCKDEBT	checking account debit date and time, amount, description
CHCKCRDT	checking account credit date and time, amount, description
SAVEACCT	savings account number, current balance, last statement date, last statement balance
SAVEDEBT	savings account debit date and time, amount, description
SAVECRDT	savings account credit date and time, amount, description

Segment Occurrences

The hierarchical database structure is useful for storing multiple occurrences of any given element of information, especially if there are varying numbers of occurrences of the data for each record.

Consider the dependent segment SAVEACCT, which contains the following fields:

- savings account number
- savings account balance
- date of last statement
- savings account balance at last statement.

Different customers can have different numbers of savings accounts; some may have none, others may have two or three. If the data are not segmented, there must be space in each customer's record for the maximum number of savings accounts per customer. With the segmented structure, however, it is possible to have one SAVEACCT segment occurrence for each savings account a customer has. Any segment type can have an unlimited number of segment occurrences. Although the segment types are predefined, the number of segment occurrences is not predefined. Note that each occurrence of a root segment represents a separate record.

Here is an example of how a segment type can have an unlimited number of segment occurrences. A certain customer has two savings accounts. In one month, the customer has two deposits for account number 111 and one deposit and two withdrawals for account number 222. Figure 2.3 on page 15 shows the customer's record.

Figure 2.3 Sample Record

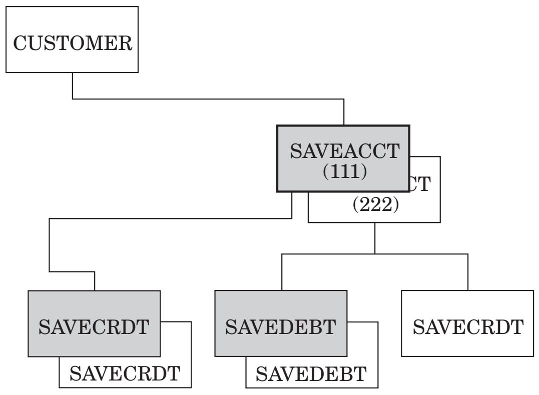


Figure 2.3 on page 15 shows eight segment occurrences within four (shaded) segment types.

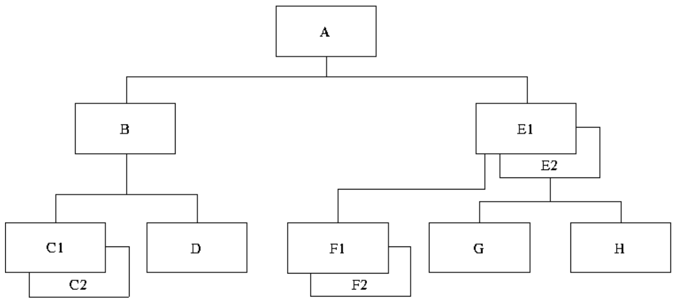
Segment Relationships

The information in a hierarchical database is subdivided or segmented according to a logical scheme. Moving from top to bottom through the database, there is a relationship between the segments. A segment that is hierarchically dependent on a segment one level up in the hierarchy is said to be the *child*. The segment on which it is dependent is the *parent*. CUSTOMER is a parent segment with two children, CHCKACCT and SAVEACCT. CHCKACCT, in turn, is the parent of CHCKDEBT and CHCKCRDT, and SAVEACCT is the parent of SAVEDEBT and SAVECRDT. Segments that share a parent are called *siblings*; for example, CHCKACCT and SAVEACCT are siblings. Multiple segment occurrences of one segment type with the same parent occurrence are called *twins*. For example, SAVEACCT 111 and SAVEACCT 222 are twins.

All dependent segments are children but are not necessarily parents. The root segment (CUSTOMER), on the other hand, is a parent if any dependent segments exist, but it is never a child. (It is possible to have a database with no dependent segments, that is, with only one level, the root segment.) In a hierarchical structure, there can be only one parent segment for a child segment.

Segments can also be grouped by *paths*. Two segments belong to the same path if one is a dependent of the other. You can access multiple segments in a path at the same time. These relationships are shown in Figure 2.4 on page 15.

Figure 2.4 Segment Relationships



Parents:
A, B, E1, E2

Children:
B, E1, E2, C1, C2, D, F1, F2, G, H

Twins:
C1 and C2, E1 and E2, F1 and F2

Siblings:
B, E1, and E2; C1, C2, and D; G and H

Paths:
A, B, and C1; A, B, and C2; A, B, and D; A, E1, and F1; A, E1, and F2; A, E2, and G; A, E2, and H

Path Navigation

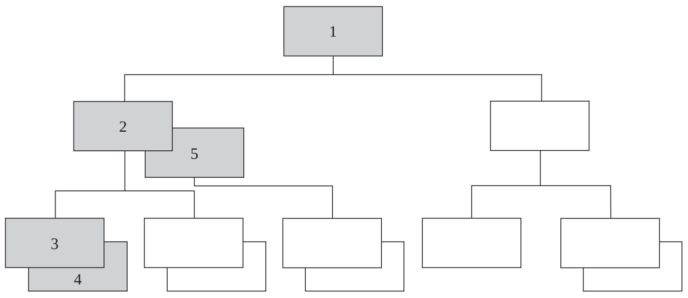
You can navigate one path of an IMS-DL/I database at a time with the interface view engine in Version 7 of the SAS System. That is, you can select items in one path of the database when creating a view descriptor. Consider Figure 2.5 on page 16, which shows one path of data shaded. The SAS/ACCESS interface processes each record occurrence from top to bottom and from left to right following these rules:

1 The first occurrence of a root segment is processed first.
2 Then, the first child of a root segment in the defined path is processed before twins of the root segment.
3 Twins are processed after a child (if any) down that path. Twins are processed in order of occurrence. Any child of a twin is processed according to this rule.
4 After the child and twins are processed for that one path, the next eligible root segment in the path is processed.

Note: No siblings are processed. △

Figure 2.5 on page 16 illustrates a path of data in a particular program view. The numbering indicates the order of processing.

Figure 2.5 A Database Path



Fields

There are three types of fields in the segments of an IMS-DL/I database:

- A *sequence field* (or *key field)* is a field that identifies and provides access to segments in a database. A sequence field is defined to IMS-DL/I in the *Database Description* (*DBD*), which specifies characteristics of a database. In some cases, a sequence field sequences twin segment occurrences in ascending order, according to their sequence field values. For example, if the sequence field of the CHCKACCT segment is ACNUMBER, twin CHCKACCT occurrences in a given customer's record are ordered from the lowest to highest account number. Root segments usually have a sequence field, but dependent segments do not necessarily have them.

  In a root segment, the sequence field also uniquely identifies the record. In dependent segments, the sequence field can provide unique identification, but this is not required. Root segments may or may not be sequenced by the sequence field, depending on the IMS-DL/I access method used to store the database.
- A *search field* is defined to IMS-DL/I in the DBD and is used to search through the database for particular values. For example, CUSTZIP is defined as a search field in the CUSTOMER segment, permitting the SAS/ACCESS interface to IMS-DL/I to search the database for records containing a specific ZIP code.
- An *undefined field* is not defined to IMS-DL/I. All fields other than sequence fields and search fields do not have to be defined in the DBD. IMS-DL/I does not know the format of an undefined field and cannot search for segments based on values in an undefined field. The format of an undefined field is determined by the program that loads the database initially.

The CUSTOMER segment of the ACCTDBD database contains examples of two kinds of fields. There are fields for Social Security number, name, city, state, country, ZIP code, address, home phone, and work phone. The Social Security number field is defined as the sequence field, meaning that it uniquely identifies the record. The name field of CUSTOMER does not uniquely identify a record because customer names may be duplicated. However, because names can be used to search through the database, the name field is defined as a search field, as are the address, city, state, country, ZIP code, home phone, and work phone fields.

The sequence field of a root segment allows direct access to the root segment. The sequence field of a dependent segment does not allow direct access to the record, but IMS-DL/I finds segments faster when searching on sequence fields rather than search fields.

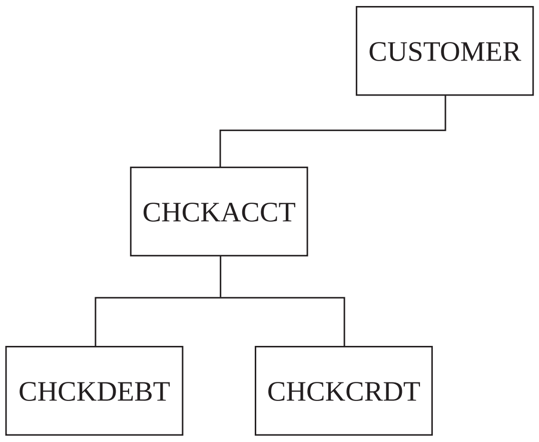
Physical Databases and Program Views

A *physical database* is defined to IMS-DL/I in one DBD, which is described later in this section. A physical database is limited to 15 hierarchical levels and 255 segment types (up to 254 dependent segments organized over 14 levels, plus the root segment). There is no limit to the number of segment occurrences, however. Figure 2.2 on page 13, earlier in this chapter, illustrates the physical database ACCTDBD.

A *program view* of a database consists of the hierarchically structured segments used in a program or application. A particular program view can be composed of all segments in a database or a subset of the segments, depending on the program's requirements. Program views are defined to IMS-DL/I in *Program Communication Blocks* (*PCBs),* which are contained in *Program Specification Blocks* (*PSBs).* (See the next several sections for more information on DBDs, PCBs, and PSBs.)

Figure 2.6 on page 18 illustrates a program view that consists of some segments from the ACCTDBD database. This program view might be used by a program that prints monthly checking account statements. However, a SAS/ACCESS view descriptor can access data in only one path in the database. Therefore, in one invocation, the view descriptor can retrieve data in either the CHCKDEBT segment or the CHCKCRDT segment.*

Figure 2.6 Sample Program View



In order for the SAS/ACCESS interface to access an IMS-DL/I database, certain information about the database must be defined. These definitions are contained in DBDs and PSBs.

What You Need to Know to Create Descriptors

Typically, DBDs and PSBs are generated by the database administration staff, not by application programmers or users. Users do not need to know how to create DBDs, PSBs, or PCBs in order to use the SAS/ACCESS interface to IMS-DL/I. If you will be creating access descriptors and view descriptors, you need to know

- □ the name of the database you want to use (DBD name)
- □ the DDnames and names of database data sets (and for HIDAM, the index data sets)
- □ the name of the PSB that contains the PCBs to be used for your database
- □ which fields are sequence or search fields (as defined in the DBD)
- □ standard segment descriptions, such as field name, field length, and data type. You may need a copybook, segment layout, or some other detailed description of the database if this information is not in the DBD.
- □ what type of access is permitted (as defined in the PCBs in the PSB)
- □ the sensitive segments (as defined in the PCBs in the PSB) and their names and lengths. A *sensitive segment* is a segment in the database that can be accessed only by using a PCB that permits read or update access
- □ the order of the PCBs in the PSB, and which PCBs your program needs to access

* With the SAS/ACCESS DATA step interface, the view shown can be processed in a single DATA step execution. See Chapter 8, “Introducing the IMS-DL/I DATA Step Interface,” on page 151 for more information.

□ the order in which fields are defined in the segment.

You can get all of this information from your installation's database administrator (DBA).

The descriptions of DBDs and PSBs that follow do not need to be understood in detail. They are included here for readers who want this additional information.

Database Description

The Database Description (DBD) is usually created by the DBA at an installation. The DBD specifies characteristics of a database, including

❶ the name of the DBD, which is also used as a shorthand name for the IMS-DL/I database (1-8 characters).

❷ the type and access method for the database (DEDB, MSDB, HDAM, HIDAM, HSAM, HISAM, GSAM, SHISAM, or SHSAM). These database types are defined in the next section.

❸ the randomizing method to assign an address to each record's key (HDAM only).

❹ the DDname for the database.

❺ the device type.

❻ the block size.

❼ the name, parent, and length of each segment type in the database. The parent information enables IMS-DL/I to determine the segment's position in the hierarchy.

❽ the name, length, starting position, and type of data for each sequence and search field in each segment. (In the following example, the code that specifies these characteristics is underlined.)

Note: It is not necessary to specify every field in a segment in the DBD. Only those fields to be used as sequence and search fields are specified in the DBD. △

DBD for the ACCTDBD Database

The following is the DBD for the ACCTDBD database.

```
DBD ❶ NAME=ACCTDBD, ❷ ACCESS=(HDAM,OSAM),         X
     ❸ RMNAME=(DFSHDC40,3,71)
DATASET ❹ DD1=ACCTDD, ❺ DEVICE=3380,              X
❻ BLOCK=2400
❼ SEGM  NAME=CUSTOMER,PARENT=0,BYTES=225
❽ FIELD NAME=(SSNUMBER,SEQ,U),BYTES=11,START=1,  X
   TYPE=C
   FIELD NAME=CUSTNAME,BYTES=40,START=12,TYPE=C
   FIELD NAME=CUSTADD1,BYTES=30,START=52,TYPE=C
   FIELD NAME=CUSTADD2,BYTES=30,START=82,TYPE=C
   FIELD NAME=CUSTCITY,BYTES=28,START=112,TYPE=C
   FIELD NAME=CUSTSTAT,BYTES=2,START=140,TYPE=C
   FIELD NAME=CUSTLAND,BYTES=20,START=142,TYPE=C
   FIELD NAME=CUSTZIP,BYTES=10,START=162,TYPE=C
   FIELD NAME=CUSTHPHN,BYTES=12,START=172,TYPE=C
   FIELD NAME=CUSTOPHN,BYTES=12,START=184,TYPE=C
  ❼ SEGM  NAME=CHCKACCT,BYTES=40,PARENT=CUSTOMER
```

```
❽ FIELD NAME=(ACNUMBER,SEQ,U),BYTES=12,START=1,  X
 TYPE=X
 FIELD NAME=STMTAMT,BYTES=5,START=13,TYPE=P
 FIELD NAME=STMTDATE,BYTES=6,START=18,TYPE=X
 FIELD NAME=STMTBAL,BYTES=5,START=26,TYPE=P
❼ SEGM  NAME=CHCKDEBT,BYTES=80,                  X
      PARENT=((CHCKACCT,DBLE)),RULES=(,LAST)
❽ FIELD NAME=DEBTAMT,BYTES=5,START=1,TYPE=P
 FIELD NAME=DEBTDATE,BYTES=6,START=6,TYPE=X
 FIELD NAME=DEBTBLNK,BYTES=2,START=12,TYPE=X
 FIELD NAME=DEBTTIME,BYTES=8,START=14,TYPE=C
 FIELD NAME=DEBTDESC,BYTES=59,START=22,TYPE=C
❼ SEGM  NAME=CHCKCRDT,BYTES=80,                  X
      PARENT=((CHCKACCT,DBLE)),RULES=(,LAST)
❽ FIELD NAME=CRDTAMT,BYTES=5,START=1,TYPE=P
 FIELD NAME=CRDTDATE,BYTES=6,START=6,TYPE=X
 FIELD NAME=CRDTBLNK,BYTES=2,START=12,TYPE=X
 FIELD NAME=CRDTTIME,BYTES=8,START=14,TYPE=C
 FIELD NAME=CRDTDESC,BYTES=59,START=22,TYPE=C
❼ SEGM  NAME=SAVEACCT,BYTES=40,PARENT=CUSTOMER
❽ FIELD NAME=(ACNUMBER,SEQ,U),BYTES=12,START=1,  X
 TYPE=X
 FIELD NAME=STMTAMT,BYTES=5,START=13,TYPE=P
 FIELD NAME=STMTDATE,BYTES=6,START=18,TYPE=X
 FIELD NAME=STMTBAL,BYTES=5,START=26,TYPE=P
❼ SEGM  NAME=SAVEDEBT,BYTES=80,                  X
      PARENT=((SAVEACCT,DBLE)),RULES=(,LAST)
❽ FIELD NAME=DEBTAMT,BYTES=5,START=1,TYPE=P
 FIELD NAME=DEBTDATE,BYTES=6,START=6,TYPE=X
 FIELD NAME=DEBTBLNK,BYTES=2,START=12,TYPE=X
 FIELD NAME=DEBTTIME,BYTES=8,START=14,TYPE=C
 FIELD NAME=DEBTDESC,BYTES=59,START=22,TYPE=C
❼ SEGM  NAME=SAVECRDT,BYTES=80,                  X
      PARENT=((SAVEACCT,DBLE)),RULES=(,LAST)
❽ FIELD NAME=CRDTAMT,BYTES=5,START=1,TYPE=P
 FIELD NAME=CRDTDATE,BYTES=6,START=6,TYPE=X
 FIELD NAME=CRDTBLNK,BYTES=2,START=12,TYPE=X
 FIELD NAME=CRDTTIME,BYTES=8,START=14,TYPE=C
 FIELD NAME=CRDTDESC,BYTES=59,START=22,TYPE=C
DBDGEN
```

DBD for the WIRETRAN Segment

The following is the DBD for the WIRETRAN segment of the WIRETRN database.

```
DBD   NAME=WIRETRN,ACCESS=(HDAM,OSAM),         X
        RMNAME=(DFSHDC40,3,71)
  DATASET DD1=WIREDD,DEVICE=3380, BLOCK=2400
  SEGM  NAME=WIRETRAN,PARENT=0,BYTES=100
  FIELD NAME=(SSNACCT,SEQ,M),BYTES=23,START=1,  X
   TYPE=C
  FIELD NAME=ACCTTYPE,BYTES=1,START=24,TYPE=C
  FIELD NAME=WIREDATE,BYTES=8,START=25,TYPE=C
  FIELD NAME=WIRETIME,BYTES=8,START=33,TYPE=C
  FIELD NAME=WIREAMMT,BYTES=5,START=41,TYPE=X
FIELD NAME=WIREDESC,BYTES=40,START=46,TYPE=C
DBDGEN
```

IMS-DL/I Database Types

During installation, the database administrator (DBA) chooses the type of database to use for the IMS-DL/I databases. The DBA decides which type of database to use based on how most of the programs that use an IMS-DL/I database will access the data in the database. The following is a list of database types that the DBA can use to define an IMS-DL/I database that is supported by the SAS/ACCESS interface to IMS-DL/I in Version 7 of the SAS System:

Data Entry Data Base (DEDB)
: is a direct-access database that consists of one or more areas, with each area containing both root segments and dependent segments. The database is accessed using VSAM improved control interval processing (ICIP). This database type can only be used with the SAS/ACCESS DATA step interface.

Main Storage Data Base (MSDB)
: is a root-segment database, residing in main storage, which can be accessed to a field level. This database type can only be used with the SAS/ACCESS DATA step interface.

Hierarchical Direct Access Method (HDAM)
: is one of DL/I's two direct-access methods. A direct-access method allows DL/I to locate any database record, regardless of the record sequence in the database, by using a randomizing routine or an index. HDAM provides direct access to data through a randomizing routine. Sequentially accessing an HDAM database, DL/I retrieves data in the order that the data are physically stored in the database.

Hierarchical Indexed Direct Access Method (HIDAM)
: is one of DL/I's two direct-access methods. HIDAM provides direct access to data through an index.

Hierarchical Sequential Access Method (HSAM)
: is one of DL/I's sequential-access methods. In a sequential-access database, segments are stored in a hierarchical sequence, one segment after another. HSAM provides sequential access to root segments and dependent segments. You can access data in HSAM databases, but you cannot update any of the data.

Hierarchical Indexed Sequential Access Method (HISAM)
: processes data sequentially, but has an index that enables you to directly access records in the database.

Generalized Sequential Access Method (GSAM)
: allows IMS/ESA batch application programs to access a sequential OS/390 data set record that is defined as a database record. This database record is handled as one unit, with no segments, fields, or hierarchical structure. Any records to be added are inserted at the end of the database. GSAM does not allow you to update or delete records in the database.

Simple Hierarchical Sequential Access Method (SHSAM)
: is an HSAM database that contains only one segment type, a root segment. Only two types of calls are valid with SHSAM databases: *Get calls* to read a database and *Insert calls* to load a database. You must reload a database in order to update it.

Simple Hierarchical Indexed Sequential Access Method (SHISAM)
: is a HISAM database with only one segment type, a root segment.

IMS-DL/I Data Types

When specifying the characteristics of the physical database in the DBD, the DBA identifies for each segment in the database the fields that IMS-DL/I can use to search or sequence a segment. The DBA can define each individual field, define the entire segment as one field and assign a generic data type, or define some fields individually and other fields as a group. The DBA may define fields in an IMS-DL/I database segment using the following data types:

Data Type Code	Data Type
X	hexadecimal
P	packed decimal
C	alphanumeric character
F	binary fullword
H	binary halfword
Z	zoned decimal
E	short floating point
D	long floating point
L	extended floating point

Note: All COBOL and PL/I data types are supported as hexadecimal data types. △

Using IMS-DL/I Data Types in SAS/ACCESS Descriptors

To create access and view descriptors to be used by the SAS/ACCESS interface to IMS-DL/I, you need to know how the DBA has defined the database fields. You also need to know how the fields are initialized and the order of all the fields in each segment to be accessed. You can get this information from a layout of the database or a COBOL copybook.

Table 2.1 on page 22 shows the DBFORMAT= value that you specify in an access descriptor for some common COBOL and PL/I data types. This table also shows the SAS variable formats that the SAS/ACCESS interface generates from the IMS-DL/I DBFORMAT= value.

Table 2.1 Recommended DBFORMAT= Values to Use for Common COBOL and PL/I Data Types

IMS-DL/I Type	COBOL	PL/I	Description	Standard Length in Bytes	Recommended DBFORMAT=	SAS Format Generated
C	PIC A	Pic 'A'	Alphabetic	<=200	$*w.*	$*w.*
				>200	$200.	$200.
C	PIC X	Char or Pic 'X'	Alphanumeric	<=200	$*w.*	$*w.*
				>200	$200.	$200.
Z	PIC 9	Pic '9'	Numeric Edited			*w.d*
Z	PIC S9	Pic '99T'	Zoned-Decimal		ZD*w.d*	*w.d*
H	PIC 9(4) COMP	Fixed Bin (15)	Fixed-Point Binary	2	IB2.	7.0

IMS-DL/I Type	COBOL	PL/I	Description	Standard Length in Bytes	Recommended DBFORMAT=	SAS Format Generated
F	PIC 9(8) COMP	Fixed Bin (31)	Fixed-Point Binary	4	IB4.	10.0
E	COMP-1	Float Bin (21)	Floating-Point	4	Rb4.	E13.0
D	COMP-2	Float Bin(53)	Floating-Point	8	RB8.	E22.0
P	COMP-3	Fixed Decimal	Packed-Decimal	<=16	PD*w.d*	*w.d*

When you create an access descriptor, you use the ITEM= statement to describe the IMS-DL/I DBD. When you need to specify a SAS informat that corresponds to a COBOL data description, refer to PICTURE and USAGE. If the USAGE is COMP-1 or COMP-2, there is no PICTURE. If no USAGE is specified, it defaults to DISPLAY.

Use the following information to make the conversions:

- □ Pictures that include either **`A`** or **`X`** represent character values.
- □ Pictures that include numbers use **`9`** to represent digits. They might use an **`S`** to mean signed and a **`V`** to show the location of an implied decimal point.

The number of characters or digits is specified either by the number of **`A`**s, **`X`**s, or **`9`**s in the picture or by the number in parentheses immediately after the **`A`**, **`X`**, or **`9`**. For example, **`AAAA`** is the same as **`A(4)`**.

Table 2.2 on page 23 shows other conversions.

Table 2.2

USAGE	PICTURE	SAS Informat	Width	Decimal
COMP-1	None	RB4.		
COMP-2	None	RB8.		
DISPLAY	9(int)V9(fract)	ZD*w.d*	(int + fract)	(fract)
COMP-3	9(int)V9(fract)	PD*w.d*	CEIL((int+fract+1)/2)	(fract)
COMP	9(int)V9(fract)	IB*w.d*	*	

* If the (*int* + *fract*) is 1-4, the width is 2 and decimal is a fraction. If the (*int* + *fract*) is 5-9, the width is 4 and decimal is a fraction. If the (*int* + *fract*) is 10-18, the width is 8 and decimal is a fraction.

Use SAS formats to print the fractional part read with the IB*w.d* and RB*w.d* SAS informats.

Program Specification Block

A Program Specification Block (PSB) is also generally created by the DBA at an installation. A PSB consists of one or more program views of one or more databases. A SAS task using the SAS/ACCESS interface to access an IMS-DL/I database must reference one and only one PSB. Information specified in the PSB includes

- □ at least one program view for each database that is accessed by the SAS/ACCESS interface in the executing task. A program view is defined in a PCB; in fact, you can use the terms program view and PCB interchangeably. Each PCB provides these specifications:
  - □ the database to be accessed.
  - □ the processing options (read-only or various updating options).

- the maximum length of the concatenated key fields (sequence fields) in any path.
- the database segments that can be accessed. These segments are called *sensitive segments*. The name, parent segment, and access mode for each sensitive segment is given.

- the programming language the PSB supports. The SAS/ACCESS interface to IMS-DL/I uses a PSB regardless of the specified programming language. This flexibility means that no additional PSBs are required for the SAS/ACCESS interface.
- the name of the PSB.

Example PSB

Here is a sample of a PSB called ACCTSAM, which contains some database PCBs for the ACCTDBD database and one PCB for the WIRETRN database:

```
PCB    TYPE=DB,DBDNAME=ACCTDBD,PROCOPT=G,       X
       KEYLEN=11
SENSEG NAME=CUSTOMER,PARENT=0,PROCOPT=G
PCB    TYPE=DB,DBDNAME=ACCTDBD,PROCOPT=G,       X
       KEYLEN=23
SENSEG NAME=CUSTOMER,PARENT=0,PROCOPT=GP
SENSEG NAME=CHCKACCT,PARENT=CUSTOMER,PROCOPT=G
SENSEG NAME=SAVEACCT,PARENT=CUSTOMER,PROCOPT=G
PCB    TYPE=DB,DBDNAME=ACCTDBD,PROCOPT=A,       X
       KEYLEN=23
SENSEG NAME=CUSTOMER,PARENT=0,PROCOPT=AP
SENSEG NAME=CHCKACCT,PARENT=CUSTOMER,PROCOPT=AP
SENSEG NAME=CHCKDEBT,PARENT=CHCKACCT,PROCOPT=A
SENSEG NAME=CHCKCRDT,PARENT=CHCKACCT,PROCOPT=A
SENSEG NAME=SAVEACCT,PARENT=CUSTOMER,PROCOPT=AP
SENSEG NAME=SAVEDEBT,PARENT=SAVEACCT,PROCOPT=A
SENSEG NAME=SAVECRDT,PARENT=SAVEACCT,PROCOPT=A
PCB    TYPE=DB,DBDNAME=WIRETRN,PROCOPT=A,       X
       KEYLEN=23
SENSEG NAME=WIRETRAN,PARENT=0,PROCOPT=A
PSBGEN LANG=ASSEM,IOASIZE=500,PSBNAME=ACCTSAM, X
 CMPAT=YES
END
```

Security Options

IMS-DL/I provides security for databases through data sensitivity, a way of controlling which data the SAS/ACCESS interface to IMS-DL/I can access. The SAS/ACCESS interface is allowed to access only data to which it is sensitive. There are three levels of data sensitivity:

segment sensitivity
: allows the IMS-DL/I interface to access only certain segments in a particular hierarchy.

field-level sensitivity
: allows the IMS-DL/I interface to access only certain fields in a particular segment.

key sensitivity
allows the IMS-DL/I interface to access only segments below a particular segment in a hierarchy. It does not allow the IMS-DL/I interface to access that particular segment, and returns only the segment's key to the interface.

The DBA can specify data sensitivity for an IMS-DL/I database in each database PCB in the PSB.

DL/I Calls

The SAS/ACCESS interface to IMS-DL/I accesses IMS-DL/I database segments by issuing DL/I calls. A *DL/I call* is a request made by the interface to DL/I to access one or more segments of a database or message queue, or to perform some system function. Certain information must be specified in the DL/I call to communicate the SAS/ACCESS interface's request to DL/I. The normal information specified in a call is

- □ a *call function* specifying the action DL/I is to perform (get, insert, replace, and so on)
- □ a program view (PCB) in the PSB to use when performing the function. The PSB to be used has been specified earlier in your view descriptor or in your DL/I INFILE statement (for an IMS-DL/I DATA step program).
- □ an I/O area to use for transferring segment data between DL/I and the SAS/ACCESS interface to IMS-DL/I
- □ up to 15 *segment search arguments* (SSAs). An *SSA* is a set of formatted search criteria that specifies the segment type or occurrence on which to perform the function.

Normally, a DL/I call accesses one segment at a time. However, by using a special command code in an SSA, the SAS/ACCESS interface to IMS-DL/I can access multiple segments along a hierarchical path in the database. This type of call is a *path call.*

The following descriptions of the elements of a DL/I call do not need to be understood in detail by most users. They are included here for readers who want this additional information.

DL/I Call Functions

DL/I calls are categorized as Get calls or Update calls. A *Get call* is a call that retrieves (reads) a segment or segments. An *Update call* performs some kind of write function, such as inserting a new segment or replacing or deleting an existing segment.

The basic DL/I database call functions are listed here. Some of the descriptions refer to SSAs, qualified calls, and unqualified calls. These are described in "Segment Search Arguments" on page 28.

GU	get-unique. If unqualified, this call retrieves (reads) the first segment in the PCB view (program view) of the database. If SSAs are specified, the call retrieves the first segment that satisfies qualifications specified by the SSAs.
GN	get-next. If unqualified, this call retrieves the next segment in the hierarchical sequence of the database. If SSAs are specified, the call retrieves the next segment that satisfies qualifications specified by the SSAs.
GNP	get-next-within-parent. This call is like the GN call but is restricted to the subtree of the current parent. (The parent is described in the PCB.)

GHU	get-hold-unique. This call is like the GU call but also holds the segment for the next update call that uses the same PCB.
GHN	get-hold-next. This call is like the GN call but also holds the retrieved segment for the next update call that uses the same PCB.
GHNP	get-hold-next-within-parent. This call is like the GNP call but also holds the segment for the next update call that uses the same PCB.
DLET	delete. This call deletes the segment retrieved by the last get-hold call using the same PCB.
REPL	replace. This call replaces the segment held from the last get-hold call using the same PCB with an updated segment that you provide. The get-hold call must be the last DL/I call that used the same PCB.
ISRT	insert. This call adds new segments using the PCB specified.

Program Communication Block

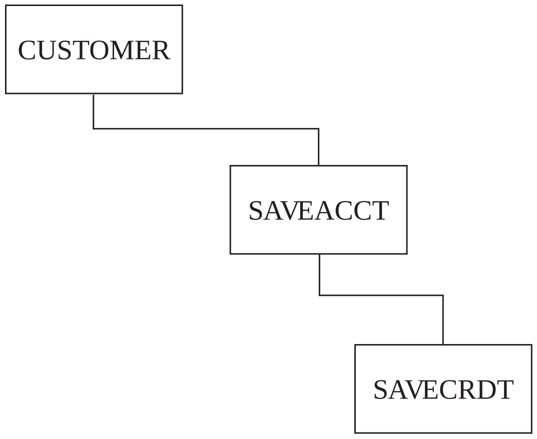
The SAS/ACCESS interface to IMS-DL/I is comprised of two distinct interfaces: the IMS-DL/I engine interface and the DATA step interface. The IMS-DL/I DATA step interface can use any PCB. You can use DL/I INFILE statement extensions to specify the PCB. The DATA step interface offers limited support for TP PCBs and message queue processing.

The IMS-DL/I engine interface (that is, where you use view descriptors) uses only two types of PCBs: the Database (DB) PCB and the Input/Output (I/O) PCB. The engine interface uses the first DB PCB that matches the database you specify in the access descriptor, unless you specify otherwise in the PCB index field of the view descriptor. For updating, the engine interface to IMS-DL/I uses the I/O PCB for checkpointing. The I/O PCB is also used if the kind of DL/I processing environment, or region type, is BMP. An I/O PCB is created when you enter CMPAT=YES in the PSBGEN statement.

Using a program view of the database (that is, using the appropriate PCB), a call can selectively access only the segments that are required by the SAS/ACCESS interface. For example, you may need the interface to retrieve savings account data from the ACCTDBD database without retrieving savings debit segments. The PCB defines as sensitive segments only those segments needed by the SAS/ACCESS interface.

A PCB to be used by the SAS/ACCESS interface that accumulates savings account credit information might use the program view shown in Figure 2.7 on page 26.

Figure 2.7 Program View of Savings Account Segments



In addition to defining a program view of the database by specifying sensitive segments, the PCB also accumulates information about the results of a call. This information, called the *PCB mask data,* includes

segment level
is the hierarchical level of the last segment successfully retrieved or processed.

DL/I status code
is a return code that indicates whether the call was successful.

DL/I processing option
is a code that indicates what kind of access to the database is allowed. The processing option might be

G	for Get
D	for Delete, includes G
I	for Insert
R	for Replace, includes G
A	for All of the above
E	for Exclusive use, in conjunction with G,D,I,R,A
L	for Loading database, excludes HISAM
LS	for Loading sequentially, required for HISAM
O	for inhibiting program isolation, must be used with G
P	for Path calls
GS	for getting segments in ascending sequence

segment name
is the name of the last segment type successfully retrieved or processed by the call.

key feedback data
is the concatenated key (sequence) field values of all segments in the path between the root segment and the last segment that was successfully retrieved or processed.

The PCB mask data also contain other information not described here. *

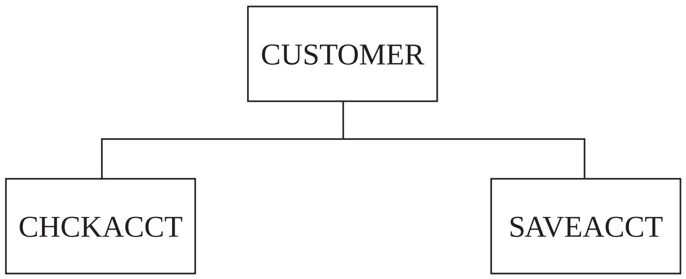
Database Position

For each PCB, DL/I maintains a *current position indicator*. The position indicator points to the last segment accessed or to the top of the database, if no DL/I call has been issued or if the last call failed. The position determines which segment should be processed next, that is, by the current DL/I call.

Suppose your DATA step program uses a PCB that defines the CUSTOMER, CHCKACCT, and SAVEACCT segments as sensitive segments. The program is a read-only program and unqualified GN (get-next) calls are issued. Therefore, the program uses sequential processing. The program view is shown in Figure 2.8 on page 28.

* The IMS-DL/I DATA step programs can return PCB mask data to the user, but the SAS/ACCESS engine interface cannot.

Figure 2.8 Program View of Account Segments



When the first GN call is issued, DL/I is positioned at the front of the database and the call retrieves the first occurrence of a CUSTOMER segment. When the next call is issued, DL/I uses the current position to determine which segment is retrieved next. In this case, CHCKACCT is retrieved before SAVEACCT because the default search sequence for sequential access is top to bottom, left to right.

Note: The database position is influenced by considerations that are not described here, such as the type of call issued and certain command codes. △

Segment Search Arguments

A DL/I call can be qualified or unqualified. A *qualified call* is one that specifies one or more SSAs. An SSA provides additional information for the DL/I call. The simplest SSA identifies a segment type for the call to access. Other SSAs not only identify the segment type, but they also specify a value or a set of values to select a particular segment occurrence. An *unqualified call* does not have any SSAs and, therefore does not specify a particular segment or set of segments.

If an SSA describes only the segment type to be accessed, it is an unqualified SSA. (The call is still a qualified call, but the SSA itself is unqualified.) In an unqualified SSA, you can also specify an optional *command code,* which may affect how the call function is performed or it may affect the qualification of a segment. See “Command Codes” on page 30 for more information.

An unqualified SSA has the form

segment-name <command code>

where *segment-name* is an 8-byte field specifying the segment type, followed by a blank. A blank follows because the minimum SSA length for a DL/I call is 9 bytes. Command codes consist of an asterisk (*) followed by a letter, such as *U or *D.

If an SSA provides a field name and specific value for that field, it is a qualified SSA. A qualified SSA has the form

segment-name <command code> (field-name operator value . . .)

where *segment-name* is the 8–byte segment type, *field-name* is the 8–byte name of a sequence or search field for that segment as defined in the DBD, and *operator* is a 2-byte field that contains a comparison operator. *Value* is a value that is compared to the specified field in the segment. The values for each of the segment type, field name, operator, and value must be padded to represent the total number of bytes used by the particular field in the DBD. IMS-DL/I requires the padding.

The first segment occurrence that satisfies the qualification or qualifications is retrieved.

For example, to retrieve a CUSTOMER segment for Hooper J. Walls, the Get (retrieve) call would be qualified with this qualified SSA:

```
CUSTOMER(CUSTNAME =WALLS, HOOPER J.         )
```

The comparison operators IMS-DL/I uses in a qualified SSA, along with their alternate forms, are listed in the following table.

Operator	Alternate Form
=	= or EQ
>	> or GT
<	< or LT
>=	=> or GE
<=	=< or LE
¬=	=¬ or NE
&	* or AND (dependent AND)
|	+ or OR (logical OR)

* Pad the =, >, and < operators with blanks on the right or left.

Using the IMSWHST= Option for Qualified SSAs

The SAS/ACCESS interface provides certain system and configuration options to use with IMS-DL/I. One such configuration option, IMSWHST=, affects qualified SSAs and applies only to the IMS-DL/I engine interface.

IMSWHST=Y makes sure that any specified WHERE criteria have been incorporated into the SSAs that are generated by the IMS-DL/I engine. Doing so limits the amount of data that the database returns.

If qualified SSAs are not generated by the view descriptor's or application's WHERE statement (or there is no WHERE data set option), the software issues an error message, no IMS-DL/I records are retrieved, and processing stops.

The default, IMSWHST=N, specifies that IMS-DL/I records are retrieved for processing regardless of whether or not qualified SSAs are passed to IMS-DL/I by a view descriptor's or application's WHERE statement.

Multiple SSAs in the DATA Step Interface

If you use the IMS-DL/I DATA step interface and specify more than one SSA for a call, you must specify them in hierarchical order. You can specify as many as 15 SSAs. The segment specified in the last SSA, the *target segment,* is the segment accessed.

For example, if you want to issue a GN (get-next) call to retrieve a CHCKDEBT segment with a DEBTDATE of 28 March 1995 for banking customer Mary T. Summers, you would qualify the GN call with these SSAs:

```
CUSTOMER*U-(CUSTNAME =SUMMERS, MARY T.          )
CHCKDEBT(DEBTDATE =032895)
```

The target segment for the call is CHCKDEBT. It is the only segment returned.

To access more than one segment in one call, you must set up a *path call*, as explained in "Command Codes."

Command Codes

Any SSA, qualified or unqualified, can include a *command code.* A command code provides still more information for the call. It may affect how the call function is performed, or it may affect the qualification of a segment. Command codes consist of an asterisk (*) followed by a letter.

One commonly used command code is *D, which signifies a path call. For a DL/I GET call, this means that the segment named in the SSA with the *D code is returned, even if it is not the target segment (the segment named in the last SSA). The segments retrieved with SSAs that specify *D are returned to the I/O area in hierarchical sequence. The target segment is placed in the I/O area behind the segments whose SSAs specified *D. (This has no effect on what is returned in the key feedback area; it affects only the I/O area.)

For example, one PCB defines CUSTOMER, CHCKACCT, and CHCKDEBT as sensitive segments. (Figure 2.5 on page 16 shows this program view.) In the IMS-DL/I DATA step interface, you specify these SSAs and a GU (get-unique) call function. Two segments are returned to the I/O area: CUSTOMER and CHCKDEBT.

```
CUSTOMER*D-(CUSTNAME =WALLS, HOOPER J.          )
CHCKACCT
CHCKDEBT(DEBTDATE =030594)
```

The PCB mask data will contain CHCKDEBT as the name of the last segment successfully retrieved, and the key feedback data contain the concatenated key fields of the CUSTOMER and CHCKACCT segments. The CHCKDEBT segment has only a search field and no sequence field, and therefore no data for CHCKDEBT are in the key feedback area.

IMS-DL/I Execution Modes

When the SAS/ACCESS interface to IMS-DL/I accesses DL/I databases, it executes within a DL/I subsystem. DL/I subsystems are either batch or online:

□ Usually, an *online DL/I subsystem* is used by multiple terminals or programs at the same time, and databases are shared by the users. The terminal users (for example, bank tellers or airline reservation agents) execute preprogrammed applications to access DL/I databases. These users may be executing the same SAS program or different SAS programs.

□ In a *batch DL/I subsystem,* only one program executes at a time, and it has exclusive use of the databases. Batch subsystems are typically used when one or more functions must be executed repetitively (for example, printing customers' monthly bank statements), and the database is not required for concurrent access by another subsystem.

Batch and online DL/I subsystems can execute concurrently. For example, a bank may run an online subsystem to which all the bank teller terminals are connected. As customers make deposits and withdrawals during the day, the tellers use checking and savings application functions to record these transactions and update a database such as the ACCTDBD database. Simultaneously, a batch DL/I subsystem may execute a SAS program to print a report of all loans with overdue payments from a loan database.

It is important to know how the SAS system options for the SAS/ACCESS interface to IMS-DL/I are set at your site. These SAS system options determine the execution mode of the SAS/ACCESS interface, which must be consistent with your IMS-DL/I system configuration.

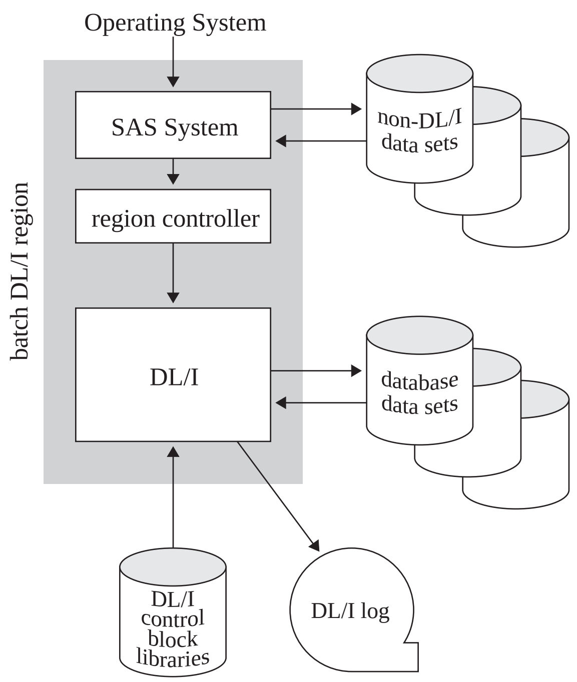
Outline of a Batch DL/I Subsystem

In a batch DL/I subsystem, a *batch region* is a processing environment for running batch mode jobs using a local batch control program. The batch region is initialized by a *region controller*, which is the primary entry point for all DL/I executions. The region controller initializes the batch region according to JCL (job control language) specifications made when the user's SAS program is submitted.

When the program runs in the batch region, the SAS/ACCESS interface to IMS-DL/I communicates with DL/I to access DL/I databases and to issue calls against databases. DL/I also handles the DL/I log, which contains information that is needed to recover changes to the database if the program terminates abnormally.

The batch region in which the SAS/ACCESS interface executes is either a DLI region or DBB region. The use of these regions depends on the operating system. Under OS/390, a *DBB region* uses the Application Control Block library (ACBLIB) to get the DBDs and PSBs. A *DLI region* uses the Database Description library (DBDLIB) and Program Specification Block library (PSBLIB) to get the DBDs and PSBs. Figure 2.9 on page 31 shows the typical batch DL/I subsystem.

Figure 2.9 Batch DL/I Subsystem



The following steps are performed when a SAS program is executed in the batch DL/I subsystem:

1 The operating system passes control to the SAS System. When the SAS/ACCESS interface initializes, it attaches a subtask to execute the DL/I region controller. Parameters that are passed to the region controller specify the type of batch region to execute (DLI or DBB), the name of the program (IMSEXEC under OS/390), PSB to use, and other execution options.

2 The region controller establishes the DL/I region environment and passes control to the SAS/ACCESS interface.

3 The SAS/ACCESS interface receives pointers to the PCBs in the PSB. It uses these PCBs in DL/I calls.

4 The SAS/ACCESS interface formats a DL/I call and passes control to DL/I to access DL/I databases.

5 DL/I accesses the database data sets, performs the requested call function, and logs any information required for recovery in the DL/I log.

6 A return code and other information for the PCB mask are placed in the PCB, and control returns to the SAS/ACCESS interface.

7 Steps 4 through 6 are repeated until the SAS procedure or SAS DATA step is completed. The region controller subtask is detached and SAS continues to process the other SAS PROC or DATA steps.

Outline of an Online DL/I Subsystem

In an online DL/I subsystem, an *online control region* is initialized and uses JCL specifications to set up the environment in which user programs execute. Under OS/390, types of online control regions include IMS/ESA DB/DC regions or CICS regions.

An online control region also allocates and controls access to DL/I database data sets for multiple-user programs, ensuring the integrity of the databases being used by many programs. Normally, the online control region obtains exclusive control of the database data sets so that other DL/I subsystems do not update the database data sets concurrently. This preserves database integrity within the overall system.

When the online control region allocates a database, it is referred to as an *online database*. The ACCTDBD database is an online database when it is allocated to an online subsystem. When the online control region is terminated, any associated databases can be used in a batch processing region. Databases can be freed to allow access by a batch program concurrent with online control region execution. Alternatively, batch and online processing can concurrently share access to databases by using IMS/ESA data sharing support.

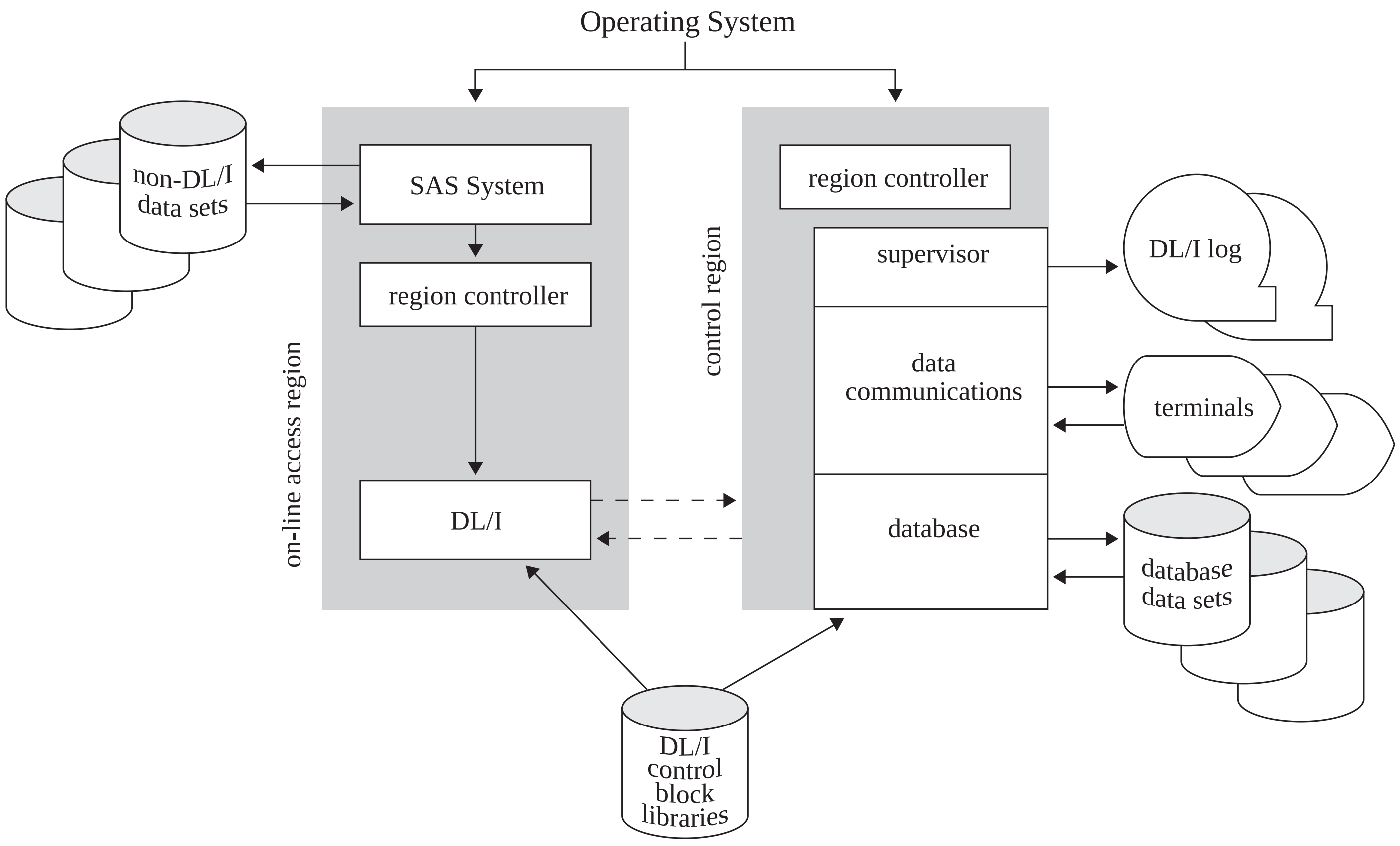
The SAS/ACCESS interface to IMS-DL/I interacts with an online control region through a DL/I online access region. The *online access region* is used when a batch program requires access to a database allocated by the online control region, that is, to an online database. There are two types of online access regions under OS/390:

- a BMP region is used to access an IMS/ESA DB/DC online control region
- a BMP region is used to access a CICS region (the DBCTL facility of IMS/ESA provides this functionality).

For example, the ACCTDBD database must be updated periodically with another database, which contains information on transactions using automated teller machines (ATMs). There is a batch program to read this transactions database and update the ACCTDBD database. However, because the ACCTDBD and transaction database data sets are allocated exclusively to the online subsystem for the tellers, a batch subsystem cannot allocate the data sets. This kind of conflict is resolved by an online access region, in which a batch program executes but issues the DL/I calls under the control of the online control region. This method preserves the integrity of the online databases.

The typical online access region and online control region interaction is depicted in Figure 2.10 on page 33.

Figure 2.10 Online DL/I Subsystem



Summary of Region Types

The following summarizes the region types that are used in the various DL/I subsystems discussed in this chapter.

Table 2.3

IBM Product	Type of Subsystem	Database Controlled by	Batch Region	Online Access Region
IMS/ESA DB	batch	region controller	DLI or DBB	
IMS/ESA DB/DC	online	control region		BMP
CICS	online	control region		BMP

Shared IMS-DL/I Database Access

Each of the IBM IMS-DL/I products allows some capability for sharing IMS-DL/I resources. Two general categories of sharing exist:

- □ sharing resources within one IMS-DL/I subsystem
- □ sharing resources between multiple IMS-DL/I subsystems.

The concepts of read integrity and update integrity are important in a description of resource sharing. *Read integrity* means that two programs cannot access a record simultaneously if one has update intent. Read integrity guarantees that the data are current when reading a record. *Update integrity* means that two programs cannot access a record simultaneously if both have update intent. Update integrity guarantees that data accessed for update are current, but it does not guarantee that data accessed for reading are current.

Resource sharing within one subsystem is the most common form of resource sharing and is available with an online IMS-DL/I subsystem. In the online IMS-DL/I subsystem, the online IMS-DL/I control region allocates the database data sets and controls concurrent access to the databases by multiple programs. Read integrity is guaranteed when sharing within an online subsystem unless the processing option GO has been specified in the PCB. For more information on the GO option, see Chapter 2, "Program Specification Block (PSB) Generation," in the *IMS/ESA Utilities Reference Manual*, or *DL/I Resource Definition and Utilities*. Update integrity is always guaranteed in an online subsystem.

In the second form of sharing, sharing resources between multiple IMS-DL/I subsystems, there are two subcategories:

□ *Database-level sharing* allows multiple IMS-DL/I subsystems to access a database concurrently. Both online and batch regions can be used.

  One subsystem can update a database while other subsystems access the same database in read-only mode. When sharing takes this form, update integrity is guaranteed, but read integrity is not guaranteed. Read integrity is guaranteed only if all subsystems use read-only access.

  Database-level sharing is available in IMS/ESA DB and IMS/ESA DB/DC systems.

□ *Block-level sharing* allows multiple IMS-DL/I subsystems to have concurrent update access to a database.

When sharing resources, IMS-DL/I preserves both read and update integrity.

Note: GSAM databases cannot be shared. △

General Considerations for Sharing Resources

When resources are shared, whether within a subsystem or between subsystems, many users can access a given database at the same time. Consequently, one invocation of the SAS/ACCESS interface to IMS-DL/I can have an impact on the performance of several users' programs.

When read integrity is guaranteed, the SAS/ACCESS interface has read-only access and *owns* (has exclusive access to) the last database record it accessed. Even under these circumstances, the SAS/ACCESS interface with read-only access does not normally affect the performance of other programs. However, if the SAS/ACCESS interface is positioned on one database record for a long time, it affects other programs by preventing them from accessing that record. If read integrity is not guaranteed, the SAS/ACCESS interface does not own records and, therefore, does not affect other programs.

The SAS/ACCESS interface is more likely to affect the performance of other programs if it updates database records. When the SAS/ACCESS interface updates records, it owns any record that has been updated since the interface's last synchronization point. A *synchronization point* occurs when the SAS/ACCESS interface issues a CHKP (checkpoint) call. This synchronization point saves the changes the SAS/ACCESS interface has made since the last CHKP call it issued to the database. By default, the SAS/ACCESS interface issues CHKP calls at the beginning and end of processing. With SAS/FSP software, use the AUTOSAVE option to increase the frequency of issuing CHKP calls.

Synchronization points are important because they cause IMS-DL/I to release some resources allocated to the SAS/ACCESS interface. These resources include the database records owned by the interface, the IMS-DL/I enqueue table entries that mark this ownership, and the dynamic log records required to back out (cancel) updates since the prior synchronization point. When IMS-DL/I releases the SAS/ACCESS interface's ownership of updated database records, other programs can access the record with the updated information.

Database-Level Shared Access

In database-level shared access, multiple IMS-DL/I subsystems (batch or online or both) allocate the database data sets concurrently. Concurrent allocation is possible in a single operating system with shared disposition allocation. It may be possible between multiple operating systems, regardless of the allocation disposition, if the database data sets reside on shared Direct Access Storage Device (DASD).

CAUTION:

If the IMS-DL/I requirements for database-level sharing are not followed closely, IMS-DL/I database integrity can be compromised by multiple allocations. Be sure that database-level sharing or block-level sharing is implemented for a database before you allocate a database data set with shared disposition. △

In database-level sharing, one subsystem can have update access to a database while other subsystems have read access to the same database. In this case, update integrity is guaranteed, but read integrity is not guaranteed. Alternatively, all subsystems can be restricted to read access, in which case read integrity is guaranteed because there is no danger of a record being updated. The remainder of this section on database-level sharing discusses sharing when one subsystem has update access and other systems have read access.

When one subsystem has update access and the others have read access, it is possible for a read-access invocation of the SAS/ACCESS interface to obtain uncommitted update data from a program that later backs out the updates.

If the subsystem with update access is a batch subsystem, only one program or invocation of the SAS/ACCESS interface has update access to the database (since only one program executes in a batch subsystem). No other program or invocation of the interface with update intent (indicated in the PCB) can execute until the first subsystem completes, so there is no contention for the database records. (Remember that read integrity is not guaranteed in this situation and programs with read access do not own records.) Since other executing programs are not waiting for records, you do not have to be concerned about releasing records for other programs to use.

If the subsystem with update access is an online subsystem, other subsystems (whether batch or online) are still restricted to read access. However, unlike a batch subsystem, multiple programs in the update-access online subsystem can update the database. In other words, two forms of sharing occur at once:

- □ database-level sharing between subsystems, with one updating and others reading
- □ sharing within one online subsystem, with multiple programs sharing the databases.

Database-level sharing is specified by

- □ registering the database with Database Recovery Control (DBRC) for database-level sharing
- □ ensuring that DBRC is used in the IMS/ESA IMS-DL/I region

- □ specifying a share option of (2,3) or (3,3) when the VSAM data set is defined.

Under OS/390, if DBRC is not used, database integrity is compromised. DBRC is active in SAS System executions of application regions as long as the value of the SAS system option IMSDLDBR= is not N.

Block-Level Shared Access

In block-level shared access, multiple IMS-DL/I subsystems allocate the database data sets concurrently. This shared allocation is possible in a single operating system with shared disposition allocation. Block-level shared access is possible between multiple operating systems regardless of the allocation disposition if the database data sets reside on shared DASD.

If the IMS-DL/I requirements for block-level sharing are not followed completely, the IMS-DL/I database integrity may be compromised by this multiple allocation. Be sure that you implement block-level sharing for a database before you allocate a database data set with shared disposition.

Block-level shared access differs from database-level shared access in that it guarantees both read and update integrity for the shared database. It is not possible for the SAS/ACCESS interface to IMS-DL/I to obtain uncommitted update data that is later backed out.

A disadvantage of block-level sharing is that different subsystems must contend for database records. Therefore, synchronization-point processing becomes essential when updating a database that is shared at the block level with other IMS-DL/I subsystems.

An advantage of block-level sharing over database-level sharing is that the SAS/ACCESS interface that updates does not have to wait to obtain exclusive update control of the database.

Block-level sharing is specified by

- □ registering the database with DBRC for block-level sharing
- □ ensuring that DBRC is used in the application region
- □ establishing communication with an IMS/ESA Resource Lock Manager (IRLM), which is executing under the same operating system as the IMS-DL/I region
- □ specifying (for VSAM data sets) a share option of (3,3) when the VSAM data set is defined.

If DBRC is not active, database integrity is compromised. If DBRC was included in IMS/ESA during operating system generation, DBRC is active in SAS System executions of application regions as long as the SAS system option IMSDLDBR= does not have a value of N.

Similarly, if communication with the IRLM is not established, database integrity is compromised. The IMS-DL/I region establishes communication with the IRLM specified by the SAS system option IMSDLIRN= as long as the IRLM is active and the SAS system option IMSDLIRL= does not have a value of N.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ ACCESS® Interface to IMS-DL/I Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 316 pp.

SAS/ACCESS® Interface to IMS-DL/I Software: Reference, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–548–5

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, by any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute, Inc.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software by the government is subject to restrictions as set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The Institute is a private company devoted to the support and further development of its software and related services.