CHAPTER

9

Using the SAS/ACCESS Interface to IMS-DL/I DATA Step Interface

Introduction

The SAS/ACCESS interface to IMS-DL/I can access databases through a DLI or DBB batch region, and an IMS/ESA DB/DC BMP region.* Chapter 8, "Introducing the IMS-DL/I DATA Step Interface," on page 151 describes DATA step programming statements and DL/I statements that are available with the IMS-DL/I DATA step interface. This chapter describes Fast Path DL/I database access and non-database access calls.

* Beginning with Version 6, the SLI region type is not supported; SLI functionality is supported through BMP regions. Databases that are allocated to CICS control regions can be accessed by SAS applications through a BMP region by using the DBCTL facility of IMS/ESA and CICS/ESA.

Fast Path DL/I Database Access

The following two Fast Path database types are supported by the IMS-DL/I DATA step interface by using a BMP region:

- Main Storage Data Bases (MSDBs) store and provide access to an installation's most frequently used data, which reside in virtual storage during execution. The data are stored in segments, and each segment can be available to all terminals or to specific terminals.
- Data Entry Data Bases (DEDBs) provide a high level of availability for, and efficient access to, large volumes of detailed data. They are hierarchic structures that contain a special type of segment that is used for the fast collection of detailed information. The segments are called *sequential dependent segments* because they are stored in time sequence as they are committed to the database.

Standard DL/I database calls can be used with a PCB that references an MSDB or DEDB to access database segments. Two additional calls are available:

- The FLD call allows read and update access to a field in an MSDB.
- The POS call returns information about the position of the current sequential dependent segment in a DEDB and free space in the DEDB area.

The IMS-DL/I DATA step interface supports the FLD and POS calls from a BMP region.

FLD Call

The FLD call is used to verify and, optionally, to update the contents of one or more fields in an MSDB segment. Individual field verification or change specifications are specified in *Field Search Arguments* (FSAs). (The format and use of FSAs are described in the IBM publication *IMS/ESA: Application Programming: EXEC DLI Commands for CICS and IMS*.) FSAs are passed to DL/I in the I/O area. Therefore, in the IMS-DL/I DATA step interface, the PUT statement is used to format the FSAs in the output buffer and to execute the FLD call.

Like any DL/I call, the FLD call returns a status code. In addition, DL/I returns abnormal status information for each FSA in the call. If a non-blank status code is returned from a FLD call, it may be necessary to examine the contents of the FSA return codes. The DL/I INFILE statement option FSARC= specifies a 200-byte character variable to which the first 200 FSA status code bytes can be returned.

The following example issues a FLD call against an MSDB called INVNTORY:

```
ssa1='PRODUCT (PRODUCT = LOCKS     )';
infile msdbpsb dli call=cfunc dbname=database
   ssa=ssa1 fsarc=fsa_rc;
file msdbpsb dli;
cfunc = 'FLD ';
database = 'INVNTORY';
put @1 'QUANTITY  H100*QUANTITY -100*ORDERS +1 ';
```

The call accesses a segment called PRODUCT containing data on locks. The FLD call performs these functions:

- verifies that the QUANTITY field is greater than 100
- updates the QUANTITY field by subtracting 100 from its current value
- updates the ORDERS field by adding 1 to its value.

If the QUANTITY field value is not greater than 100 when the FLD call is executed, the return code for the first FSA contains a D. The following statements check for errors in the call and print an appropriate message on the SAS log for this error:

```
if _error_ then do;
   file log;
   if substr(fsa_rc,1,1) = 'D'
   then put / '*** Quantity of Product Locks Less
      Than 100 ***';
   put _all_;
   _error_=0;
   end;
```

POS Call

The POS call is used with a DEDB to perform one of the following:

- □ retrieve the position of a specific sequential dependent segment
- □ retrieve the position of the last inserted sequential dependent segment
- □ find out how much free space is available within a DEDB area.

In an IMS-DL/I DATA step program, the POS call is issued with a DL/I INPUT statement and a DB PCB. After a POS call is issued, the input buffer is formatted with the requested data as explained in the IBM publication *IMS/ESA: Application Programming: EXEC DLI Commands for CICS and IMS*.

The SAS statements below execute a POS call for a DEDB called ORDERS:

```
retain ssa1 'PRODUCT (PRODUCT = LOCKS     )';
infile dedbpsb dli call=cfunc dbname=database
   ssa=ssa1;
cfunc = 'POS ';
database = 'ORDERS  ';
input @3 areaname $char8.
      @11 cycl_cnt $pib4.
      @15 vsam_rba $pib4.;
```

The call obtains the position of the last inserted ORDRITEM sequential dependent segment for the locks PRODUCT segment.

Non-Database Access Calls

Some DL/I calls communicate with DL/I for reasons other than database access. This section describes how to use the non-database calls in IMS-DL/I DATA step programs.

Most non-database calls can be used only in particular DL/I region types. An exception is the basic CHKP call, which is supported in all DL/I region types that can be accessed through OS/390. The basic CHKP call is described in the next section.

Most of the other DL/I calls in this section require an I/O PCB or TP PCBs, so descriptions of the I/O and TP PCBs follow the section on the basic CHKP call. Following that section is a section on calls that can be issued by all OS/390 DL/I region types. Finally, calls that can be executed only from an IMS/ESA BMP region are described.

Basic CHKP Call

The basic CHKP call can be issued in batch DL/I regions as well as in online DL/I regions. This call establishes a program synchronization point.* (Synchronization points are described in "General Considerations for Sharing Resources" on page 34.)

The following example shows SAS programming statements that issue a CHKP call. The example is run using the SAS system option IMSREGTP=DLI:

```
data _null_;
  retain chkpnum 0;
  infile acctsam dli call=func pcb=pcbindex
    status=st;
  file acctsam dli;
  func = 'CHKP';
  pcbindex = 1;
  chkpnum = chkpnum +1;
  put @1 'SAS'
      @4 chkpnum z5.;
  if st = '  ' then
    return;
  file log;
  put _all_;
  abort;
run;
```

The CHKPNUM variable, first referenced in the RETAIN statement, is used to build a *checkpoint ID*. A checkpoint ID is an 8-byte value that is written to the DL/I log record to identify the program checkpoint. A checkpoint ID is not required but is very useful and should be included routinely in programs that issue CHKP calls. In this example, the checkpoint ID is built in the output buffer. If the same sequence of statements is used for each CHKP call, the checkpoint ID is incremented by 1 for each call.

The PCB= variable, PCBINDEX, has a value of 1. This indicates that the first eligible PCB is used for the CHKP call. A CHKP call requires the I/O PCB that is the first PCB in the PSB (see "I/O PCBs" on page 202).

Note: An I/O PCB is always generated for PSBs in a BMP region. If you are going to issue a CHKP call under DL/I, you must use the CMPAT=YES option in the PSBGEN statement for batch regions DLI and DBB. If an I/O PCB is not present, you get the message that the call is invalid for a DB PCB. △

The CHKP call is successful if _ERROR_=0 and the STATUS= variable (ST) is blank. Otherwise, the STATUS= variable contains a status code that indicates the cause of the failure. In particular, an **XD** status code in an IMS/ESA BMP region indicates that the IMS/ESA control region is being shut down.

CHKP Calls in IMS/ESA BMP Regions

A CHKP call performs an additional function when it is issued in an IMS/ESA BMP transaction-processing program (that is, when the SAS system option IMSBPIN= specifies a valid transaction code and the PCB used is type TP). In addition to establishing a synchronization point, the call returns the first segment of the next message to the call's I/O area. Since a CHKP call is issued by a DL/I PUT statement, the I/O area is the SAS output buffer.

* The OS/VS checkpoint option of the CHKP call in an IMS/ESA DL/I region is not supported in the IMS-DL/I DATA step interface.

You cannot read from the output buffer in a DATA step, but you can access the message segments that are placed in the output buffer. You format a GU call that uses the I/O PCB. When the DL/I INPUT statement executes, the SAS/ACCESS interface remembers that the output buffer contains data from a previous CHKP call. Instead of issuing the GU call, the SAS/ACCESS interface moves the segment from the output buffer to the input buffer, where it can be read. Therefore, in a BMP transaction-processing program, the first call issued after a CHKP call *must be* a GU that references the I/O PCB.*

Consider the previous example in "Basic CHKP Call" on page 200, which shows SAS statements that issue a CHKP call. If you issue the CHKP call in a BMP transaction-processing program, additional statements are needed. This example issues one CHKP call and optionally moves a message segment to the input buffer.

In this example, change *trancode* in the OPTIONS statement to a valid transaction code at your site. This example is run using the SAS system options IMSREGTP=BMP and IMSBPIN=*trancode*:

```
options imsbpin=trancode;
data _null_;
  retain chkpnum 0;
  infile acctsam dli call=func pcb=pcbindex
     status=st;
  file acctsam dli;
  func = 'CHKP';
  pcbindex = 1;
  chkpnum = chkpnum +1;
  put @1 'SAS'
      @4 chkpnum z5.;
  if st = '  ' then
    do;
      func = 'GU  ';
      input @;
      if st ¬= '  ' then
        if st= 'QC' then
          do;
            _error_ = 0;
            stop;
          end;
        else
          link abendit;
    end;
  else
    if st = 'QC' then
      do;
        _error_ = 0;
        stop;
      end;
  else
    link abendit;
  stop;

  abendit:
    file log;
```

* This is not the call sequence that would be used if programming in PL/I, COBOL, or Assembler, but it is consistent with the actions taken by DL/I after a CHKP call.

```
      put _all_;
      abort;
  run;
  options imsbpin=*;
```

If DL/I did not return the first segment of the next message automatically after a CHKP call, the GU call would be necessary to retrieve the next message.

I/O PCBs

An I/O PCB is a Program Communication Block that is used only in OS/390 DL/I environments. An I/O PCB is similar to a DB PCB, but an I/O PCB communicates non-database access requests to DL/I instead of database requests. The type of DL/I region executed and an option specified when PSBs are generated determine whether an I/O PCB is included in a PSB. The IMS/ESA control region automatically provides an I/O PCB for BMP regions. The I/O PCB is generated in batch DL/I regions if the CMPAT=YES option is specified in the PSBGEN statement when the PSB is generated.

If an I/O PCB is present, it is always the first PCB in the PSB. Therefore, be careful in how you specify the DL/I INFILE statement options PCBNO=, PCB=, and DBNAME= when you need the I/O PCB. The value of PCBNO= must be 1. If the DBNAME= option is specified, that variable's value must be set to blanks. Finally, if a PCB= variable is specified, it must have a value of 1.

In all OS/390 DL/I regions, the I/O PCB is used to issue the CHKP and LOG calls. In an IMS/ESA BMP region, the I/O PCB is also used to read transaction messages from the IMS/ESA message queues, to insert response messages to the terminal that originated the transaction, and to communicate certain system calls that are unique to the IMS/ESA DB/DC system.

TP PCBs

A TP PCB is a Program Communication Block used with the IMS-DL/I DATA step interface only in IMS/ESA BMP regions. It is similar to the I/O PCB, but there are two important differences:

- A TP PCB is used *only* to insert messages to terminal or transaction message queues. A TP PCB cannot be used for a Get call to a message queue.
- Unlike an I/O PCB, a TP PCB can direct a message to a destination (transaction or terminal message queue) other than the terminal that originated the message.

There are two kinds of TP PCBs: *nonmodifiable* and *modifiable*. A *nonmodifiable* TP PCB has a fixed destination that is specified when the PSB is generated. The destination can be either a terminal or transaction message queue. A *modifiable* TP PCB does not have a destination associated with it when the PSB is generated. Instead, the program must set the destination before using the PCB to insert a message to the message queue. The destination can be changed between messages so that more than one destination can be accessed by one TP PCB.

When TP PCBs are present, they follow the I/O PCB (if any) and precede the DB PCBs. Unless the TP PCB is the first PCB in the PSB, you must use the PCB= option in the DL/I INFILE statement to select the appropriate TP PCB. You cannot use the DBNAME= option because no DBD name is associated with a TP PCB.

Feedback Data

Just as information from DB PCBs is available to the SAS program through the STATUS= and PCBF= variables after a DL/I call, so is information from the I/O and TP PCBs.* The format of the data in the PCBF= variable differs, however, according to the PCB type.

If a DL/I call uses the I/O PCB, the PCBF= variable data are formatted as shown in Table 9.1 on page 203.

If a DL/I call uses a TP PCB, the data in the PCBF= variable are formatted as shown in Table 9.2 on page 204.

Table 9.1 Format of I/O PCB Feedback Data

Bytes	Description
1-8	These bytes of the PCBF= variable contain the name of the logical terminal (LTERM) that issued the message.
9-10	These bytes are reserved for IMS/ESA usage.
11-12	These bytes contain the DL/I status code. The status code may also be obtained by specifying the STATUS= option in the DL/I INFILE statement.
13-16	These bytes contain the date that the message was queued. The date is in packed decimal, right aligned, Julian date format (YYDDD).
17-20	The time that the message was queued is contained in these bytes in packed decimal format (HHMMSS.S).
21-24	The input message number assigned by IMS/ESA is contained in these bytes in IB4. (full-word binary) format.
25-32	These bytes contain the Message Output Descriptor (MOD) name. An MOD name is connected to this PCB if Message Format Services (MFS) is used. If MFS is not used, there is no MOD, and this field is blank.
33-40	These bytes contain the User identification data. The contents vary according to the source of the message

If a DL/I call uses a TP PCB, the data in the PCBF= variable are formatted as shown in Table 9.2 on page 204.

* *IMS/ESA: Application Programming: EXEC DLI Commands for CICS and IMS*, an IBM publication, describes the PCB mask data.

Table 9.2 Format of TP PCB Feedback Data

Bytes	Description
1-8	These bytes of the PCBF= variable contain the name of the destination associated with the PCB.
9-10	These bytes are reserved for IMS/ESA usage.
11-12	These bytes contain the DL/I status code. The status code may also be obtained by specifying the STATUS= option in the DL/I INFILE statement.

OS/390 DL/I System Calls

Table 9.3 on page 204 summarizes the functions and region types for non-database access calls that are supported by the IMS-DL/I DATA step interface.

Table 9.3 Summary of Non-Database Access Calls

Function	Purpose	Valid Region Types	Notes
CHKP	create the synchronization point, recovery purposes	all IMS-DL/I DATA step interface region types	OS/VS option not supported. In transaction-processing BMPs, next call must be GU using I/O PCB.
CHNG	change destination for messages	IMS/ESA BMP regions	sets the destination for a modifiable TP PCB
CMD	issue IMS/ESA commands from a program	IMS/ESA BMP regions	when CC status returned, must next issue GU to retrieve response
DEQ	release a class of segments enqueued with the Q command code	IMS/ESA BMP regions	specify class (A-J) of segments to dequeue
FLD	access fields in MSDBs	IMS/ESA BMP regions	Fast Path Facility only
GCMD	retrieve additional response segments to a command if more than one	IMS/ESA BMP regions	functions as a GN to the queue after first response segment retrieved with GU
GN	retrieve additional segments of a message with more than one segment	IMS/ESA BMP regions	uses I/O PCB
GU	retrieve the first segment of a message	IMS/ESA BMP regions	uses I/O PCB
ISRT	format and send message segment to the queue	IMS/ESA BMP regions	uses I/O or TP PCB
LOG	insert a record to the DL/I system log	MVS DL/I regions	uses I/O PCB
POS	return position information from DEDBs	IMS/ESA BMP regions	Fast Path Facility only.

Function	Purpose	Valid Region Types	Notes
PURG	terminate the current message being inserted; optionally, insert the first segment of the next message	IMS/ESA BMP regions	uses TP PCB
ROLB	back out database updates since last sync point	IMS/ESA BMP regions and batch DL/I regions in IMS/ESA Release 3	in BMP regions, also back out messages inserted to the queue since the last synchronization point. Next call must be GU using I/O PCB if ROLB requested return of previous message.
ROLL	back out database updates since last sync point, and abend	IMS/ESA BMP regions, batch DL/I regions in IMS/ESA Release 3, and CICS/VS shared DL/I regions	in BMP regions also back out messages inserted to the queue since the last synchronization point

LOG Call

A LOG call inserts user log records in the DL/I log with the I/O PCB (see “I/O PCBs” on page 202). To insert a log record, you must specify

- □ the text of the log record
- □ a valid log code
- □ a value for the ZZ field
- □ the value of the LL field, which is the sum of the lengths of the log record, log code, ZZ field, and LL field.

In an IMS-DL/I DATA step program, the LOG call is issued with the DL/I PUT statement. The PUT statement must format the log record being inserted. The following statements from a sample program insert a log record with a code of **'A0'x** in the IMS log. The example can be run using the SAS system options IMSREGTP=DLI or IMSREGTP=BMP:

```
data _null_;
  infile acctsam dli call=func pcb=pcbindex
    status=st;
  file acctsam dli;
  func = 'LOG ';
  pcbindex = 1;
  ll = 23;
  zz = '0000'x;
  logcode = 'A0'x;
  logsegm = 'Text of Log Record';
  put @1 ll pib2.
      @3 zz
      @5 logcode
      @6 logsegm;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
```

```
run;
```

After the LOG call, you can check the values of the STATUS= variable and _ERROR_ to see whether the call was successful. If _ERROR_=0, the log record was inserted properly. Otherwise, the STATUS= variable contains an error code that indicates why the call was not successful.

If the PSB is generated with LANG=PLI, then the PUT statement must be modified because the LL field has a 4-byte length:

```
put @1 ll pib4.
       @5 zz
       @7 logcode
       @8 logsegm;
```

The value of the LL variable does not change.

ROLL Call

In an online access region, the ROLL call has two purposes:

- to back out any DL/I updates to database segments or message queues that have been made since the last program synchronization point
- to abend the program with a user 0778 completion code.

The ROLL call performs the same functions in a batch DL/I region if the following conditions are present:

- a DASD log data set is used
- the IMS-DL/I DATA step interface option IMSDLBKO= specifies a value of Y.

Otherwise, the ROLL call in a batch DL/I region only causes the program to abend with a user 0778 completion code. In this latter case, the database back-out utility must be run with the log data set in order to back out any database updates made since the last program synchronization point.

The following example shows statements that issue a ROLL call. This example is run using the SAS system option IMSREGTP=DLI:

```
data _null_;
  infile acctsam dli call=func pcb=pcbindex
    status=st;
  file acctsam dli;
  func = 'ROLL';
  pcbindex = 1;
  put;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
run;
```

ROLB Call

A ROLB call is used in a batch DL/I region to back out any DL/I database updates that have been made since the last program synchronization point. ROLB differs from the ROLL call because it does not cause an 0778 abend. The ROLB call requires use of the I/O PCB (see “I/O PCBs” on page 202).

The ROLB call can be issued in batch DL/I regions if

- □ a DASD log data set is used
- □ the IMS-DL/I DATA step interface option IMSDLBKO= specifies a value of Y.

Otherwise, the ROLB call can be issued only from an IMS/ESA BMP region, as described in "IMS/ESA Message Queue Access" on page 212.

The following sequence of SAS statements issues a ROLB call. This example is run using the SAS system options IMSREGTP=DLI and IMSDLBKO=Y:

```
options imsdlbko=y;
data _null_;
  infile acctsam dli call=func pcb=pcbindex
    status=st;
  file acctsam dli;
  func = 'ROLB';
  pcbindex = 1;
  put;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
run;
```

The ROLB call has been successfully executed if _ERROR_=0 after the call; otherwise, you can check the value of the STATUS= variable to see why the call did not complete successfully.

IMS/ESA BMP System Calls

DEQ Call

The DEQ call is used in a BMP region to dequeue a class of database segments that have been enqueued with the Q command code of a Get call. The DEQ call is issued with the PUT statement and requires the use of the I/O PCB. The PUT statement specifies the class of segments to be dequeued. The following sequence of SAS statements dequeue the segments that have been enqueued to Class A with a QA command code in a Get call. This example is run using the SAS system option IMSREGTP=BMP:

```
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'DEQ ';
  pcbindex = 1;
  put @1 'A';
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
```

```
run;
```

The call has been successfully executed if _ERROR_=0 after the call. Otherwise, the STATUS= variable contains a status code that indicates the reason for the failure.

ROLB Call

The ROLB call is used in a BMP region to back out any DL/I updates to database segments or message queues that have been made since the last program synchronization point. The ROLB call is issued with a PUT statement and requires the use of the I/O PCB.

Examples 1 to 3 are run using the SAS system options IMSREGTP=BMP and IMSBPIN=*trancode*. Example 1 shows a sequence of SAS statements that issue a ROLB call.

```
options imsbpin=trancode;
data _null_;
  infile acctsam dli call=func pcb=pcbindex
     status=st;
  file acctsam dli;
  func = 'ROLB';
  pcbindex = 1;
  put;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
run;
```

The call has been successfully executed if _ERROR_=0 after the call. Otherwise, the ST variable contains a status code that indicates the reason for the failure.

If the ROLB call is issued in a BMP transaction processing program *and* the DL/I PUT statement issuing the call formats non-blank data in columns 1 through 6, the call also returns the first segment of the previous message. Any non-blank data can be written in columns 1 through 6 of the output buffer.

When these conditions are fulfilled, the IMS-DL/I DATA step interface saves the returned message segment. The next call *must be* a GU that uses the I/O PCB. The DATA step interface intercepts the GU call when the INPUT statement executes, so the call is not actually issued. Instead, the returned segment is moved to the input buffer where it can be read.

Example 2 shows a sequence of SAS statements that issue a ROLB call and then a GU call with the I/O PCB:

```
/* put a message in the queue      */
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'ISRT';
  pcbindex = 2;
  ll = 33;
  zz = '0000'x;
  msgsegm = 'trancode Message for Example # 2.';
  put @1 ll pib2.
```

```
        @3 zz
        @5 msgsegm;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
run;
data _null_;
  infile acctsam dli call=func pcb=pcbindex
       status=st;
  pcbindex = 1;
  file acctsam dli;
  func = 'ROLB';
  put @1 'SAVEIO';
  if st ¬= '  ' then
    if st = 'QC' then
      _error_ = 0;
    else
       link abendit;
    func = 'GU  ';
    input @;
    if st = '  ' then
       _error_ = 0;
    else
       link abendit;
    stop;

    abendit:
      file log;
      put _all_;
      abort;
run;
```

Example 3 shows a sequence of SAS statements that issue a ROLB call and with no GU call to the message queue:

```
data _null_;
  infile acctsam dli call=func pcb=pcbindex
     status=st;
  file acctsam dli;
  func = 'ROLB';
  pcbindex = 1;
  put @1 'SAVEIO';
  if st ¬= '  ' and
     st ¬= 'QC' then
    link abendit;
  return;

  abendit:
    file log;
    put _all_;
    abort;
```

```
run;
options imsbpin=*;
```

The message segment has been successfully moved if _ERROR_=0 after the INPUT statement executes.

If the PUT statement above is changed to **PUT;**, the message segment would *not* be returned by the ROLB call.

CMD Call

A SAS program that executes in a BMP region can insert commands to IMS/ESA with the CMD call if

- □ the IMS/ESA security allows the PSB and transaction to do so and
- □ BMPREAD= does not specify Y.

The CMD call is issued by a PUT statement and uses the I/O PCB.

For example, the following sequence of SAS statements issues the **'/START DB ACCTDBD. '** command. This example is run using the SAS system options IMSREGTP=BMP and IMSBPIN=*trancode* :

```
options imsbpin=trancode;
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'CMD ';
  pcbindex = 1;
  ll = 23;
  zz = '0000'x;
  put @1 ll pib2.
      @3 zz
      @5 '/START DB ACCTDBD. ';
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
run;
options imsbpin=*;
```

If _ERROR_=0 after the call, the command was issued properly. If a blank STATUS= code is returned, the command may have completed or it may be in progress, depending on the IMS/ESA command issued.

If a CC status code is returned, the command returned a response message to the output buffer and the IMS-DL/I DATA step interface saved the response. To retrieve the response, the next call must be a GU that uses the I/O PCB, as is done after CHKP and ROLB calls in the IMS-DL/I DATA step interface. If subsequent response segments are queued, a CC status code is returned as a result of the GU call. The program can issue GCMD calls (see "GCMD Call" on page 211) to retrieve the subsequent response segments.

See the IBM publication *IMS/ESA: Application Programming: EXEC DLI Commands for CICS and IMS* for more information on the CMD call.

If the PSB is generated with LANG=PLI, the format specified for the LL field must be changed to PIB4.:

```
put @1 ll pib4.
       @5 zz
       @7 '/START DB D1MK0001.';
```

However, the value of the LL variable does not change.

GCMD Call

A SAS program that issues CMD calls can retrieve additional response segments with the GCMD call. The GCMD call acts like a GN to the queue and is issued with an INPUT statement. The first segment must have been retrieved with a GU call by using the I/O PCB.

The following sequence of statements issues a GCMD call. This example is run using the SAS system options IMSREGTP=BMP and IMSBPIN=*trancode* :

```
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  func = 'GU  ';
  pcbindex = 1;
  input @;
  if st = 'CC' then
    do;
      func = 'GCMD';
      input @;
      if st = '  ' or
         st = 'QD' then
        do;
          _error_ = 0;
          stop;
        end;
      else
        link abendit;
    end;
  else
    if st = 'QC' then
      do;
        _error_ = 0;
        stop;
      end;
    else
      link abendit;
  return;

  abendit:
    file log;
    put _all_;
    abort;
run;
options imsbpin=*;
```

If _ERROR_=0 after the call, the next response segment is in the input buffer. If a QD status code is returned, there are no more response segments for this response.

IMS/ESA Message Queue Access

If you use the IMS-DL/I DATA step interface to access IMS-DL/I data and use that data in programs with a BMP region, you can access the IMS/ESA control region message queues as well as DL/I databases. A BMP program accesses message queues in two ways:

1. A program that is *transaction driven* reads a transaction message from the message queues using the I/O PCB.
2. A program can insert messages to terminal message queues or transaction message queues. When responding to the terminal that originated a transaction, the I/O PCB is used. When inserting a message to a terminal queue that did not originate the message or to a transaction queue, a TP PCB is used.

See the IBM publication *IMS/ESA: Application Programming: EXEC DLI Commands for CICS and IMS* for more information about IMS/ESA data communications programming. This section describes the use of an IMS-DL/I DATA interface to issue DL/I message queue access calls.

Get Calls That Use the I/O PCB

To retrieve message segments for transaction processing, an IMS-DL/I DATA step interface program

- □ must have the IMS-DL/I DATA step interface option IMSBPIN= set to a valid transaction code
- □ issues Get calls with the I/O PCB using DL/I INPUT statements.

To retrieve the first segment of any message, use a GU call. To retrieve subsequent segments of the same transaction message, issue a GN call. You can use the same sequence of SAS statements that issued a GU call for the first segment of a message, but the value of FUNC must be changed to GN. (For more information on GU and GN calls, see Table 9.3 on page 204 in “OS/390 DL/I System Calls” on page 204.)

In this example, change *trancode* in the OPTIONS statement to a valid transaction code at your site. This example is run using the SAS system options IMSREGTP=BMP and IMSBPIN=*trancode* :

```
options imsbpin=trancode;
data _null_;
  infile acctsam dli call=func pcb=pcbindex
     status=st;
  func = 'GU  ';
  pcbindex = 1;
  input @;
  if st = '  ' then
    do;
      func = 'GN  ';
      do while (st = '  ');
        input @;
        if st ¬= '  ' then
          if st = 'QD' then
            do;
              _error_ = 0;
              stop;
            end;
          else
            link abendit;
          end;
        end;
      else
        if st = 'QC' then
          do;
            _error_ = 0;
            stop;
          end;
        else
          link abendit;
      stop;

      abendit:
        file log;
        put _all_;
        abort;
    run;
    options imsbpin=*;
```

A transaction message segment has been successfully retrieved if _ERROR_=0 or if the STATUS= variable is blank after the call. If _ERROR_ does not equal 0, check the value of the STATUS= variable. When _ERROR_=1 and ST='QC' or ST='QD', there are no more messages in the queue. To find out if there are more messages in the queue, issue another GU call.

The format of a retrieved message segment in the SAS input buffer differs depending on the language that generated the PSB. If an Assembler PSB is used, the message segment is formatted as shown in Table 9.4 on page 213.

Table 9.4 Assembler PSB Input Buffer Message Segment Format

Bytes	Description
1-2	These bytes of the SAS buffer contain a value that is the length of the segment data plus 4 (2 for the LL field and 2 for the ZZ field) in the PIB2. format.
3-4	These bytes contain the ZZ fields and are reserved for IMS usage.
5-*n*	The segment data begin at byte 5. If this is the first segment of the message, the transaction code (up to 8 bytes in length) is in the first bytes of the message data.

If a PL/I PSB is used, the message segment is formatted as shown in Table 9.5 on page 214.

Table 9.5 PL/I PSB Input Buffer Message Segment Format

Bytes	Description
1-4	These bytes of the SAS buffer contain a value that is the length of the segment data plus 4 (2 for the LL field and 2 for the ZZ field) in the PIB4. format. (The length here will be 2 bytes less than the total message segment.)
5-6	These bytes contain the ZZ fields and are reserved for IMS usage.
7-*n*	The segment data begin at byte 7. If this is the first segment of the message, the transaction code (up to 8 bytes in length) is in the first bytes of the message data.

ISRT Calls to Message Queues

A SAS program executing in a BMP region can insert messages to the IMS/ESA control region message queues with an ISRT call and the I/O or TP PCBs. For message segments to be inserted

- either IMSBPIN= or IMSBPOUT= must specify a valid IMS/ESA destination
- BMPREAD= must not equal Y
- the message segment text must be specified
- a value must be assigned to the ZZ field
- the value of the LL field must be specified. The LL field contains the length of the message segment, which is the sum of the lengths of the text, the ZZ field, and the LL field.

The following SAS statements insert a message segment. This example uses the second PCB in the PSB, which is assumed to be a TP PCB. In this example, change *trancode* in the OPTIONS statement to a valid transaction code at your site. This example is run using the SAS system options IMSREGTP=BMP and IMSBPIN=*trancode* :

```
options imsbpin=trancode;
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'ISRT';
  pcbindex = 2;
  ll = 35;
  zz = '0000'x;
  msgsegm = 'trancode Text of Message Segment';
  put @1 ll pib2.
      @3 zz
      @5 msgsegm;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
     end;
   stop;
run;

data _null_;
    infile acctsam dli call=func pcb=pcbindex
      status=st;
    func='GU  ';
    pcbindex= 1;
    input @;
    if st ¬= '  ' then
      if st = 'QC' then
        do;
          _error_ = 0;
          stop;
        end;
      else
        do;
          file log;
          put _all_;
          abort;
        end;
   stop;
run;
options imsbpin=*;
```

If _ERROR_=0 after the ISRT call, the segment was inserted properly. Otherwise, the STATUS= variable contains a status code that indicates why the call was not successful.

If the PSB is generated with LANG=PLI, the PUT statement must be modified because the length of the LL field is 4 bytes. For example:

```
put @1 ll pib4.
       @5 zz
       @7 msgsegm;
```

The value of the LL variable does not change.

Notes on Inserting Message Segments

- If the destination of the message is a transaction queue, the text of the first segment of the message must contain the transaction code. This code must match the destination in the TP PCB.
- If Message Format Services (MFS) is used, a Message Output Descriptor (MOD) is associated with the PCB used for the call. If you want to change the MOD that is associated with the PCB, specify an SSA value of "#MODNAME=*modname*" when the first message segment is inserted.* In the previous example, you could add this statement before the first DL/I PUT statement for the message:

  ```
  SSA1='#MODNAME=DFSMO4';
  ```

  This causes the message to be formatted with the MOD DFSMO4. The **`SSA1=' ';`** statement should follow the first DL/I PUT so that the MOD is not re-specified on ISRT calls for subsequent message segments.

* Although a message queue call does not use an SSA, it is provided as a way to specify the MOD.

PURG Calls for Message Segments

You might want your SAS DATA step program to insert multiple messages with one TP PCB. The requirements for this may vary depending on whether the messages go to the same destination or to different destinations.

When you insert more than one message to the same destination, you can use a PURG call to terminate the current message *and* to insert the first segment of the next message. You issue the PURG call with a PUT statement that formats the first segment of the message to be inserted.

For example, consider the following SAS statements:

```
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'PURG';
  pcbindex = 2;
  ll = 27;
  zz = '0000'x;
  msgsegm = 'Text of Message';
  put @1 ll pib2.
      @3 zz
      @5 msgsegm;
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
run;
```

The PCBNDX variable is set to 2, so that a TP PCB is used. The values of the LL and ZZ fields are set by assignment statements, and then the message segment text is specified. Notice that the PUT statement, which issues the PURG call, formats the output buffer just as if this were an ISRT call. This example is run using the SAS system option IMSREGTP=BMP.

If you want to change the MOD, use an SSA variable, as described in "ISRT Calls to Message Queues" on page 214.

When you insert messages to different destinations with one TP PCB, you cannot use the PURG call to insert the first segment of the next message. Instead, you should

- issue a PURG call with the TP PCB to end the current message. The PUT statement that issues the PURG call *must not* format a message segment. The PUT statement should simply be **PUT;**
- issue a CHNG call to change the TP PCB destination.
- issue an ISRT call to insert the message segment.

"CHNG Call to TP PCBs" on page 216 shows an example of this sequence of calls. Remember that you must use a modifiable TP PCB in order to change destination between calls.

CHNG Call to TP PCBs

A CHNG call is issued to set or change the destination for a modifiable PCB. Issue CHNG calls to alter the destination before the ISRT calls when you need to

- set a destination for a modifiable TP PCB

- insert message segments in more than one message queue by using one modifiable PCB.

For example, the following SAS statements issue a CHNG call to set the destination of the third PCB in the PSB to *destname*, where *destname* must be a valid IMS/ESA transaction code or logical terminal name. This example is run using the SAS system option IMSREGTP=BMP:

```
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'CHNG';
  pcbindex = 3;
  put @1 'destname';
  if st ¬= '  ' then
    do;
      file log;
      put _all_;
      abort;
    end;
  stop;
run;
```

The destination has been changed successfully if _ERROR_=0 after the call. Otherwise, the STATUS= variable contains a status code that indicates the reason for the failure.

If a modifiable TP PCB is used to send messages to more than one destination, the PURG call must be used to complete the current message prior to issuing a CHNG call to alter the destination for a new message. The following example shows the PURG, CHNG, and ISRT call sequence. It is run using the SAS system option IMSREGTP=BMP:

```
data _null_;
  infile tranpsb dli call=func pcb=pcbindex
     status=st;
  file tranpsb dli;
  func = 'PURG';
  pcbindex = 3;
  put;
  if st = '  ' then
    do;
      func = 'CHNG';
      put @1 '<destname>';
      if st = '  ' then
        do;
          func = 'ISRT';
          ll = 27;
          zz = '0000'x;
          msgsegm = 'Text of Message Segment';
          put @1 ll pib2.
              @3 zz
              @5 msgsegm;
          if st = '  ' then
            stop;
          else
            link abendit;
            end;
          else
            link abendit;
        end;
      else
        link abendit;
      return;

      abendit:
        file log;
        put _all_;
        abort;
    run;
```

In the above example, the PCBINDEX variable points to the third PCB, which is a modifiable TP PCB. The PURG call is issued by a PUT statement. Because this PURG call only terminates the current message and does not insert a message segment, the PUT statement has no specifications. If _ERROR_=0, the PURG call is successful and the program goes on to issue a CHNG call. The destination specified for the TP PCB is changed.

If the CHNG call is successful, a message segment is built and an ISRT call is issued. The PUT statement issuing the ISRT call formats the output buffer.

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

ISBN 1–58025–548–5

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

1st printing, October 1999