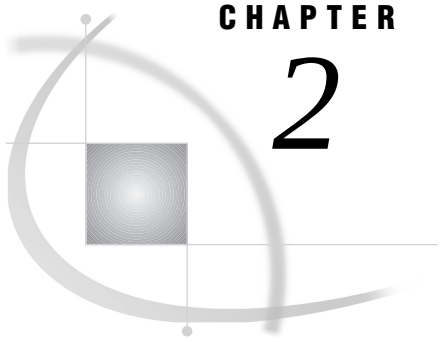

CHAPTER 2

Using the SAS/ACCESS Interface to CA-IDMS

Overview of the DATA Step Statement Extensions

Special SAS System extensions to the standard SAS INFILE statement enable you to access CA-IDMS data in a SAS DATA step. The extended statement is referred to as the CA-IDMS INFILE statement and its corresponding INPUT statement is referred to as the CA-IDMS INPUT statement. The CA-IDMS INFILE and CA-IDMS INPUT statements work together to generate and issue calls to CA-IDMS. A CA-IDMS DATA step can contain standard SAS statements as well as the SAS statements that are used with the SAS/ACCESS interface to CA-IDMS.

The CA-IDMS INFILE statement defines to the SAS System the parameters that are needed to build CA-IDMS calls. The CA-IDMS INFILE statement

- names the subschema
- names SAS variables to contain
  - the dictionary name
  - the database name
  - the node name (for distributed DBMS)
  - CA-IDMS functions (for example, OBTAIN or FIND)
  - the area name
  - the set name
  - the record name
  - the sort field
  - the database key
  - the CALC key
  - the key offset
  - the key length
  - the status returned by the call.

When it is executed, the CA-IDMS INPUT statement formats and issues the CA-IDMS function call using the parameters specified in the CA-IDMS INFILE statement.

The CA-IDMS INFILE statement is required in any DATA step that accesses a CA-IDMS database because the special extensions of the CA-IDMS INFILE statement specify the variables that set up the CA-IDMS calls. When a CA-IDMS INFILE statement is used with a CA-IDMS INPUT statement, the database function calls are issued.

The syntax and usage of the CA-IDMS INFILE and INPUT statements are described in detail later in this chapter.

CA-IDMS Record Currency

You need to understand the concept of currency before using the DATA step interface to CA-IDMS. CA-IDMS keeps track of the most recently accessed record by its database location or db-key. As each record is accessed, it becomes current of the run-unit, record type, set, or area. Some DML calls require that certain currencies are established before the call is issued. See your CA-IDMS documentation for more information about currency.

CA-IDMS Input Buffer

A buffer is allocated by the SAS System as an input area for data retrieval. The length of this buffer is specified by the LRECL= option in the CA-IDMS INFILE

statement. The input buffer is formatted by CA-IDMS in the same way an input area for any CA-IDMS program is formatted.

The data INFORMATS specified in the CA-IDMS INPUT statement must match the original data format. This information can be obtained from CA-IDMS Integrated Data Dictionary (IDD) or from a COBOL or Assembler copy library, source programs, a SAS macro library, or other documentation sources. Database Administrator (DBA) staff at your installation can help you find the segment data formats you need.

Introductory Example of a DATA Step Program

The following example is a simple DATA step program that reads record occurrences from a CA-IDMS database and creates a SAS data set. Next, the program processes the SAS data set with PROC PRINT.

The example accesses the EMPLOYEE database with the subschema EMPSS01. This subschema allows access to all of the DEPARTMENT records. This example uses the IDMS option in the INFILE statement, which tells the SAS System that this particular external file reference is for a CA-IDMS database.

The numbers in the program correspond to the numbered comments following the program.

```
❶ data work.org_department;
   retain iseq;
❷ infile empss01 idms func=func1 record=recname
      area=iarea sequence=iseq errstat=err
      set=iset;

   /* BIND the DEPARTMENT record  */
❸ if_n_ = 1 then do;
      func1   = 'BIND';
      recname  = 'DEPARTMENT';
❹     input;
      if (err ne '0000') then go to staterr;
      iseq = 'FIRST'
   end;

   /* Now get the DEPARTMENT records by issuing    */
   /* OBTAIN for DEPT record and test for success  */

   func1     = 'OBTAIN';
   recname   = 'DEPARTMENT';
   iarea     = 'ORG-DEMO-REGION';
❺    input @;
❻ if (err ne '0000' and err ne '0307') then go to
      staterr;
   if err eq '0307' then do;
      _error_ = 0;
      /* No more DEPT records so STOP */
      stop;
   end;
❼ input
      @1   department_id     4.0
      @5   department_name   $char45.
      @50  department_head   4.0;
```

```
❽ iseq = 'NEXT';
❾ return;
   staterr:
❿ put @1 'WARNING: ' @10 func1 @17
      'RETURNED ERR =' @37 err;
      atop;
   end;
   run;

⓫ proc print data=work.org_department;
   run;
```

❶ The DATA statement references a temporary SAS data set called ORG_DEPARTMENT, which is opened for output.

❷ The INFILE statement tells the SAS System to use the EMPSS01 subschema. The IDMS option tells SAS that EMPSS01 is a CA-IDMS subschema instead of a fileref. This statement also tells the CA-IDMS interface to use the named SAS variables as follows:

- FUNC1 to store the function type
- RECNAME to store the record name
- IAREA to store the area name
- ISEQ to store the function call sequence information
- ISET to store the set name

The CA-IDMS INFILE statement also tells the interface to store the error status from the call in ERR.

❸ The first time through the DATA step, all CA-IDMS records that will be accessed must be bound to CA-IDMS. To bind the DEPARTMENT record type, the program sets FUNC1 to BIND and RECNAME to DEPARTMENT.

❹ The CA-IDMS INPUT statement uses the values in the SAS variables FUNC1 and RECNAME to generate the first call to CA-IDMS. In this example, the call generated is a BIND for the DEPARTMENT record. All records must be bound to CA-IDMS before any data retrieval calls are performed. A null INPUT statement is used because the BIND function does not retrieve any CA-IDMS data.

❺ This INPUT statement also uses the values in the SAS variables FUNC1 and RECNAME, along with the values in ISEQ and IAREA to generate an OBTAIN FIRST DEPARTMENT RECORD IN AREA ORG-DEMO-REGION call. However, no data are moved into the program data vector because no variables are defined in the **`INPUT @;`** statement. The call holds the contents of the input buffer and allows the DATA step to check the call status that is returned from CA-IDMS.

❻ The program examines the status code returned by CA-IDMS. If CA-IDMS returns 0000, then the program proceeds to the next INPUT statement. If CA-IDMS does not return 0000 or 0307, then the program branches to the error routine.

❼ When this INPUT statement executes, data are moved from the input buffer into the program data vector.

❽ The ISEQ value is changed to NEXT to generate an OBTAIN NEXT DEPARTMENT RECORD IN AREA ORG-DEMO-REGION.

❾ For the subsequent interations of the DATA step, the RETURN statement causes execution to return to the beginning of the DATA step.

❿ For any unexpected status codes, a message is written to the SAS log and the DATA step stops.

⓫ The PRINT procedure prints the contents of the WORK.ORG-DEPARTMENT data set.

Output 2.1 on page 7 shows the SAS log for this example.

Output 2.1 SAS Log

```
1          data work.org_department;
2          infile empss01 idms func=func1 record=recname area=iarea
3             sequence=iseq errstat=err set=iset;
4
5          err = '0000';
           .
           .
           .
37         end;
38         run;

NOTE: The infile EMPSS01 is:
      Subschema=EMPSS01
NOTE: 11 records were read from the infile EMPSS01.
      The minimum record length was 0.
      The maximum record length was 56.
NOTE: The data set WORK.ORG_DEPARTMENT has 9 observations and 3 variables.
NOTE: The DATA statement used 0.22 CPU seconds and 2629K.
39         proc print data=work.org_department;
40         run;

NOTE: The PROCEDURE PRINT printed page 1.
```

Output 2.2 on page 7 shows the output of this example.

Note: The log shows that 11 records were read from the infile, but Output 2.2 on page 7 shows only 9 observations. Every time the SAS System encounters a CA-IDMS INPUT statement that submits a call, it increments by one an internal counter that keeps track of how many record occurrences are read from the database. The count is printed to the SAS log as a NOTE. Because this program contains CA-IDMS INPUT statements that do not retrieve data, this count can be misleading. △

Output 2.2 Department List

```
                    The SAS System
Obs    department_id   department_name             department_
                                                       head
 1       2000           ACCOUNTING AND PAYROLL          11
 2       3200           COMPUTER OPERATIONS              4
 3       5300           BLUE SKIES                     321
 4       5100           BRAINSTORMING                   15
 5       1000           PERSONNEL                       13
 6       4000           PUBLIC RELATIONS                 7
 7       5200           THERMOREGULATION               349
 8       3100           INTERNAL SOFTWARE                3
 9        100           EXECUTIVE ADMINISTRATION        30
```

Creating DATA Step Views

The preceding introductory DATA step example can be made into a DATA step view. A DATA step view is a SAS data set of type VIEW that contains a definition of the data rather than containing the physical data. For CA-IDMS, a DATA step view is a compiled version of statements that, when executed, access and retrieve the data from CA-IDMS.

A DATA step view is a stored SAS file that you can reference in other SAS tasks to access data directly. A view's input data can come from one or more sources, including external files and other SAS data sets. Because a DATA step view only reads (opens for input) other files, you cannot update the view's underlying data. For a complete description of using DATA step views, refer to *SAS Language Reference: Dictionary*.

Note: Version 7 does not allow you to name a fileref for a task that has the same name as the CA-IDMS subschema. △

The following DATA step code is part of a SAS macro that is invoked twice to create two DATA step views. When the DATA step views are referenced in the SET statements of the subsequent DATA step executions, DEPARTMENT records are read from the CA-IDMS database and selected record data values are placed in two SAS data sets. Then, each SAS data set is processed with PROC PRINT to produce the same output as shown in Output 2.2 on page 7.

The numbers in the program correspond to the numbered comments following the program.

```
❶ %macro deptview(viewname=,p1=,p2=,p3=);
❷ data &viewname / view &viewname;
❸ keep &p1 &p2 &p3;
    retain iseq;
    infile empss01 idms func=func1 record=recname
          area=iarea sequence=iseq errstat=err
          set=iset;

   /* BIND the DEPARTMENT record  */
   if _n_ eq 1 then do;
      func1    = 'BIND';
      recname  = 'DEPARTMENT';
      input;
      iseq     = 'FIRST';
   end;

   /* Now get the DEPARTMENT records  */
   func1     = 'OBTAIN';
```

```
   recname   = 'DEPARTMENT';
   iarea     = 'ORG-DEMO-REGION';
   input @;
   if (err ne '0000' and err ne '0307') then go to
      staterr;
   if err eq '0307' then do;
      _error_ = 0;
      /* No more DEPT records so STOP */
      stop;
   end;
   input
   @1   department_id     4.0
   @5   department_name   $char45.
   @50  department_head   4.0;
   iseq = 'NEXT';
   return;
   staterr:
   put @1 'WARNING: ' @10 func1 @17
       'RETURNED ERR = '@37 err;
       stop;
❹ %mend;
❺ %deptview(viewname=work.deptname , p1=DEPARTMENT_ID,
      p2=DEPARTMENT_NAME);
❻ %deptview(viewname=work.depthead , p1=DEPARTMENT_ID,
      p2=DEPARTMENT_HEAD);

   options linesize=132;

❼ data work.deptlist;
   set work.deptname;

❽ proc print data=work.deptlist;
   title2 'DEPARTMENT NAME LIST';

❾ data work.headlist;
   set work.depthead;

❿ proc print data=work.headlist;
   title2 'HEADS OF DEPARTMENTS LIST';

   run;
```

❶ %MACRO defines the start of the macro DEPTVIEW, which contains 4 parameter variables: one required and three input overrides. VIEWNAME is required; it is the name of the DATA step view. VIEWNAME can be overridden at macro invocation. The overrides are P1, P2, and P3. These may or may not be specified, but one must be specified to avoid a warning message.

P1	name of the first data item name to keep
P2	name of the second data item name to keep
P3	name of the third data item name to keep

Three data items are allowed because there are 3 input fields in the CA-IDMS INPUT statement for the database.

❷ The DATA statement specifies the DATA step view name.

❸ The KEEP statement identifies the variables that are available to any task that references this input DATA step view.

❹ %MEND defines the end of macro DEPTVIEW.

❺ %DEPTVIEW invokes the macro and generates a DATA step view named WORK.DEPTNAME that, when referenced as input, supplies observations containing values for the variables DEPARTMENT_ID and DEPARTMENT_NAME.

❻ %DEPTVIEW invokes the macro and generates a DATA step view named WORK.DEPTHEAD that, when referenced as input, supplies observations containing values for the variables DEPARTMENT_ID and DEPARTMENT_HEAD.

❼ Data set WORK.DEPTLIST is created using the DATA step view WORK.DEPTNAME as input.

❽ PROC PRINT prints WORK.DEPTLIST.

❾ Data set WORK.HEADLIST is created using the DATA step view WORK.DEPTHEAD as input.

❿ PROC PRINT prints WORK.HEADLIST.

The CA-IDMS INFILE Statement

If you are unfamiliar with the standard INFILE statement, refer to *SAS Language Reference: Dictionary* for more information.

A standard INFILE statement specifies an external file to be read by an INPUT statement. A CA-IDMS INFILE statement specifies a subschema, which in turn identifies the CA-IDMS database, records, and elements to be accessed with CA-IDMS calls. Special extensions in the CA-IDMS INFILE statement specify SAS variables and constants that are used to build a CA-IDMS call and to handle the data returned by the call. A subset of the standard INFILE statement options can also be specified in a CA-IDMS INFILE statement.

Use the following syntax when you issue a CA-IDMS INFILE statement:

INFILE *SUBSCHname* IDMS <*options*>;

where

SUBSCHname
specifies the name of the subschema used to communicate with CA-IDMS in the current DATA step. A subschema name is required and must immediately follow INFILE. (A standard INFILE statement would specify a fileref in this position.) You can open only one subschema per DATA step.

IDMS
tells the SAS System that this INFILE statement refers to a CA-IDMS database. IDMS is required and must follow the subschema name.

options
usually define SAS variables that contain CA-IDMS information used to generate DML calls. These variables are not added automatically to a SAS output data set

(that is, they have the status of variables that are dropped). To include the variables in an output SAS data set, create separate variables and assign values to them. The variables do not need to be predefined before specification in the CA-IDMS INFILE statement. The SAS System defines them automatically with the correct type and length. The following sections describe the options that are valid in the INFILE statement.

CA-IDMS Environment Options

The following options affect how the bind-run call is generated. All of the environment options are optional. If any of the next four options' values should change during the execution of the DATA step, a finish call is executed, followed by a new bind-run call.

DANAME=*variable*
specifies a SAS variable that contains the logical CA-IDMS database name, as defined in the database name table.

DANODE=*variable*
specifies a SAS variable that contains the DC/UCF of CA-IDMS where the database is defined. Use this option only if you are running a Distributed Database System.

DCNAME=*variable*
specifies a SAS variable that contains the name of the CA-IDMS dictionary where the subschema is defined. Use this option only if you are using a subschema that is defined in a dictionary other than the default dictionary.

DCNODE=*variable*
specifies a SAS variable that contains the DC/UCF system needed to process the database requests. Use this option only if you are running a Distributed Database System.

Other CA-IDMS Options

The following list describes additional options that are available only on the CA-IDMS INFILE statement.

AREA=*variable*
names a SAS variable that contains the name of the CA-IDMS AREA you want to access. The AREA must be included in the subschema that was specified on the INFILE statement.

DBKEY=*variable*
names a SAS variable to which the database record's key, db-key, is assigned after successful execution of an ACCEPT or a RETURN call to the database. A record's db-key can then be used to access a record directly. In this case, the DBKEY variable contains the db-key of the record that you want to access directly, along with FIND or OBTAIN in the FUNC= variable.

ERRSTAT=*variable*
names a SAS variable to which the CA-IDMS call status is assigned after each CA-IDMS call. If ERRSTAT= is not specified, call status codes are not returned. The variable is a character variable with a length of 4.

It is highly recommended that you check the call status codes that CA-IDMS returns, and this option provides a convenient way to do so. (See "Checking Call Status Codes" on page 35 for more information on checking call statuses in CA-IDMS DATA step programs.)

FUNC=*variable*
names a SAS variable that contains the CA-IDMS call function that is used when the CA-IDMS INPUT statement is executed. The variable must be assigned a valid CA-IDMS call function code before a CA-IDMS INPUT statement is executed. The value of the FUNC= variable can be changed between calls. The valid function calls are BIND, FIND, OBTAIN, ACCEPT, GET, IF, and RETURN. Each of these function calls is described in "Specifying DML Function Calls" on page 14.

IKEY=*variable*
specifies a SAS variable that contains the CALC KEY. Owner records of a set can be predefined to have a CALC key. Using the CALC key enables direct access to the owner records. The IKEY option is used with the IKEYLEN and KEYOFF options.

IKEYLEN=*variable*
specifies a SAS variable that contains the length of the CALC key. The SAS variable for the IKEYLEN option is defined as a numeric variable.

KEYOFF=*variable*
specifies a numeric SAS variable that is set to the position of the CALC key within the CA-IDMS record.

LRECL=*length*
specifies the length of the SAS buffers that are used as I/O areas when CA-IDMS calls are executed. The length must be greater than or equal to the length of the longest record accessed. If LRECL= is not specified, the default buffer length is 1000 bytes. See "CA-IDMS Input Buffer" on page 4 for more information.

RECORD=*variable*
specifies a SAS variable that contains the name of the CA-IDMS record type you want to access. The record type must be included in the subschema that was specified on the INFILE statement.

SEQUENCE=*variable*
names a SAS variable that contains the requested record location within the set or area. This variable can also establish currency and/or determine the direction of the traversal. Valid values for the SEQUENCE SAS variable are:

- NEXT
- FIRST
- LAST
- PRIOR
- *n*th
- CURRENT
- OWNER
- DUP
- USING

SET=*variable*
names a SAS variable that contains the name of the CA-IDMS set you want to access. The set must be included in the subschema that was specified on the INFILE statement.

SORTFLD=*variable*
names a SAS variable that contains the sort control element to be used in searching the sorted set. If the FUNC= variable contains RETURN, SORTFLD= will contain the record's symbolic key, after successful completion of the call to CA-IDMS.

Standard INFILE Statement Options

The following standard INFILE statement options can be specified in a CA-IDMS INFILE statement:

OBS=*n*
specifies, in a CA-IDMS DATA step program, the maximum number of CA-IDMS function calls to execute. This number includes INPUT statements that do not retrieve data, such as BIND.

STOPOVER
stops processing if the record returned to the input buffer does not contain values for all the variables that are specified in the CA-IDMS INPUT statement.

OBS= and STOPOVER are the only standard INFILE options that can be specified in a CA-IDMS INFILE statement.

One other standard INFILE statement option, the MISSOVER option, is the default for CA-IDMS INFILE statements and does not have to be specified. The MISSOVER option prevents the SAS System from reading past the current record data in the input buffer if values for all variables specified by the CA-IDMS INPUT statement are not found. Variables for which data are not found are assigned missing values. Without the default action of the MISSOVER option, SAS would issue another function call anytime the INPUT statement execution forced the input pointer past the end of the record.

Refer to *SAS Language Reference: Dictionary* for complete descriptions of these options.

Summary of CA-IDMS INFILE Statement Options

Table 2.1 on page 13 summarizes the CA-IDMS INFILE statement options.

Table 2.1 Summary of CA-IDMS INFILE Statement Options

Option	Specifies
AREA=	variable that contains CA-IDMS area name
DANAME=	variable that contains database to be accessed by the run unit
DANODE=	variable that contains the central version of CA-IDMS where the database resides
DBKEY=	variable that contains a database record's key
DCNAME=	variable that contains the name of the CA-IDMS dictionary where the subschema is defined
DCNODE=	variable that contains the DC/UCF system needed to process the database requests
ERRSTAT=	variable to which the CA-IDMS error status is assigned after each CA-IDMS call
FUNC=	variable that contains the CA-IDMS call function used when a CA-IDMS INPUT statement is executed
IKEY=	variable that contains the value of the CALC KEY
IKEYLEN=	variable that contains the length of the CALC key

Option	Specifies
KEYOFF=	variable that is set to the position of the CALC key within the CA-IDMS record
LRECL=	the length of the SAS buffers used as I/O areas when CA-IDMS calls are executed
<MISSOVER>	prevents the SAS System from reading past the current record data in the input buffer if values for all variables specified by the CA-IDMS INPUT statement are not found. Specified by default.
OBS=	the maximum number of CA-IDMS function calls to be issued by the DATA step
RECORD=	variable that contains the name of the CA-IDMS record you want to access
SEQUENCE=	variable that contains the requested record location within the set or area, and/or establishes currency, and/or determines the direction of the traversal
SET=	variable that contains the name of the CA-IDMS set you want to access
SORTFLD=	variable that contains the value of the sort-control element to be used in searching the sorted set
STOPOVER	stops processing if the record returned to the input buffer does not contain values for all variables specified in the CA-IDMS INPUT statement

Using the CA-IDMS INFILE Statement

You access CA-IDMS records and sets, one record at a time, using the CA-IDMS INFILE and INPUT statements.

By specifying options on the INFILE statement, you can generate navigational DML calls to CA-IDMS. To issue the appropriate DML calls, you need a thorough knowledge of the database structure.

The CA-IDMS access method that you need to use depends on how the sets were defined to the database. The access methods are CALC, CURRENT, DBKEY, OWNER, SORT KEY, or WITHIN.

The DATA step interface determines what type of access method to generate the calls for, based on the DML function call and options that you specify in the INFILE statement. Valid DML functions are OBTAIN, FIND, BIND, ACCEPT, GET, IF, and RETURN. The OBTAIN and GET functions are the only functions that retrieve a record's contents from the database.

Specifying DML Function Calls

The following sections describe which options to use to issue each of the CA-IDMS function calls: ACCEPT, BIND, FIND, OBTAIN, GET, IF, and RETURN.

Each box that appears below shows the required and optional information that needs to be specified in INFILE statement option variables. The INFILE statement option variables are SAS variables assigned in the INFILE statement.

For example, to generate the ACCEPT CURRENCY function call, you must first assign INFILE statement option variables by using FUNC=, RECORD=, and SEQUENCE=. Then you can give the variables the values ACCEPT, DEPARTMENT, and CURRENT, respectively. See the example below for a detailed description of the ACCEPT CURRENCY function call.

Note: The values of INFILE statement option variables remain set and are used for each subsequent function call unless you override or reassign their values. △

ACCEPT

The ACCEPT db-key statement moves the db-key of the current record to the DBKEY= option variable that you have defined in the CA-IDMS INFILE statement. After accepting the db-key, you can use the FIND or OBTAIN db-key statements to access records directly by using the db-key you saved from the ACCEPT db-key function call.

The db-key is a unique 4-byte identifier assigned to a record when the record is stored in the database. The db-key remains unchanged until the record is erased or the database is unloaded and reloaded. Any record in the subschema can be accessed directly using its db-key, regardless of its location.

Note: If other function calls to CA-IDMS are made before you want to use the db-key again, it must be copied into another variable. If the db-key is not needed for the next function call, it must be blanked out, or its value will be used in the function call, which will produce unexpected results. △

To generate the ACCEPT CURRENCY <*record-name*|*set*|*area*> INTO DBKEY function call, specify these options:

- FUNC= ACCEPT
- DBKEY= contains the current record's DBKEY
- SEQUENCE= CURRENT|NEXT|PRIOR|OWNER

And one of these options:

- RECORD= the IDMS record name
- SET= the IDMS set name
- AREA= the area the record participates in

The following example shows the ACCEPT CURRENCY function call for the DEPARTMENT record. The numbers in the program correspond to numbered comments following the program.

```
    infile empss01 idms func=func1 record=rec1
           dbkey=key1 errstat=err sequence=seq1;
    .
    .
    .
❶ func1 = 'ACCEPT';
❷ rec1  = 'DEPARTMENT';
❸ seq1  = 'CURRENT';
    input;
    if err eq '0000' then do
❹ put @1 'DBKEY OF RECORD = '  @19 key1;
    .
    .
    .
```

❶ FUNC1 is assigned the value of ACCEPT.

❷ REC1 is assigned the record name DEPARTMENT because you want the db-key of this record. Before you can issue an ACCEPT function call for a specific record, you must first establish currency on the record.

❸ SEQ1 is set to CURRENT to indicate that you want the db-key of the DEPARTMENT record which is current of the run unit.

❹ after successful execution of the the ACCEPT function call, KEY1 contains the db-key for the current DEPARTMENT record. The PUT statement prints the value of KEY1 on the SAS log.

The following example shows the ACCEPT NEXT function call for the DEPT-EMPLOYEE set. The numbers in the program correspond to the numbered comments following the program.

```
    infile empss01 idms func=func1 set=set1
           dbkey=key1 errstat=err sequence=seq1;
    .
    .
    .
❶ func1   = 'ACCEPT';
❷ set1    = 'DEPT-EMPLOYEE';
❸ seq1    = 'NEXT';
    input;
    if err eq '0000' then do
❹ put @1 'DBKEY OF RECORD = '  @19 key1;
    .
    .
    .
```

❶ FUNC1 is assigned the function of ACCEPT.

❷ SET1 is assigned the set name that is current of the run unit. If, for example, you have currency on the EMPLOYEE record, the ACCEPT NEXT causes the db-key of the next record in the DEPT-EMPLOYEE set to be returned from the function call to CA-IDMS. The next record in the DEPT-EMPLOYEE set could be either an EMPLOYEE record or a DEPARTMENT record, depending on your location in the set when the ACCEPT NEXT function call is issued.

❸ SEQ1 is set to NEXT to indicate that you want the db-key from the next record in the DEPT-EMPLOYEE set.

❹ after successful execution of the ACCEPT function call, KEY1 contains the db-key for the NEXT record. The PUT statement prints the db-key on the SAS log.

You can now save the db-key to use now or later with the OBTAIN or FIND functions. Using the db-key gives you direct access to the record regardless of established currencies.

BIND

The only form of the BIND function that is needed in the CA-IDMS DATA step is the BIND RECORD. The BIND RECORD statement establishes addressability for a CA-IDMS record so that its data can be retrieved and placed into the input buffer. A BIND RECORD must be issued for every record type the DATA step will access before any data are retrieved. The BIND RECORD function call does not retrieve any data from CA-IDMS. A BIND function call is not necessary if no data are being retrieved, i.e., you are issuing a FIND, ACCEPT, or RETURN function call.

To generate the BIND RECORD function call, specify these options:

- FUNC= BIND
- RECORD= the IDMS record name

The following code example shows the BIND RECORD function call. The numbers in the program correspond to the numbered comments following the program.

```
   infile empss01 idms func=func1 record=recname
   .
   .
   .
❶ func1    = 'BIND';
❷ recname  = 'DEPARTMENT';
❸ input;
   .
   .
   .
```

❶ FUNC1 is assigned the function of BIND.

❷ RECNAME is assigned the value of DEPARTMENT because this is the record on which you want to perform the BIND RECORD.

❸ This INPUT statement generates and submits the BIND RECORD function call to CA-IDMS.

FIND and OBTAIN

The FIND function locates a record in the database. The OBTAIN function locates a record and moves the data from the record to the input buffer. The FIND and OBTAIN functions have identical options so they will be discussed together. There are six formats of the FIND and OBTAIN functions. Each one will be described individually.

FIND/OBTAIN CALC

The FIND/OBTAIN CALC function accesses a record by using its CALC key value. The record must be stored in the database with a location mode of CALC. The FIND/OBTAIN CALC DUP function accesses duplicate records with the same CALC key as the current record, provided that the current record of the same record type had been accessed using FIND/OBTAIN CALC.

For an example program that locates records directly using CALC key values that have been stored in a SAS data set, see “Example: Using the Trailing @ and the INPUT with No Arguments” on page 42.

To generate the FIND|OBTAIN CALC *record-name* function call, specify these options:

- FUNC= FIND or OBTAIN
- RECORD= an IDMS record name
- IKEY= a valid IDMS record CALC key
- KEYOFF= the offset into the record where the CALC key is located
- IKEYLEN= the length of the CALC key

To generate the FIND|OBTAIN CALC DUP *record-name* function call, include:

- SEQUENCE = 'DUP'

The following example shows a FIND CALC function call for the EMPLOYEE record followed by an OBTAIN CALC DUP for the same record. The numbers in the program correspond to the numbered comments following the program.

```
    infile empss01 idms func=funct record=recname
           ikey=ckey keyoff=key0 errstat=stat
           sequence=seq ikeylen=klen;
    .
    .
    .
❶ funct      = 'FIND';
❷ recname    = 'EMPLOYEE';
❸ ckey       = '0101';
❹ key0       = 0;
❺ klen       = 4;
❻ input;
    .
    .
    .
❼ funct      = 'OBTAIN';
❽ seq        = 'DUP';
    if stat eq '0000' then do
❾ input @1   employee_id    4.0
          @5   firstname        $char10.
          @15  lastname         $char15.
          @30  street           $char20.
          @50  city             $char15.
          @65  state            $char2.
          @67  zip              $char9.
          @76  phone            10.0
          @86  status           $char2.
          @88  ssnumber         $char9.
          @97  startdate        6.0
          @103 termdate         6.0
          @109 birthdate        6.0;
    .
    .
    .
```

❶ FUNCT is assigned the value of FIND.

❷ RECNAME is assigned the name of the record that you want to access. In this example, the record is the EMPLOYEE record.

❸ CKEY is assigned the character value of '0101', which is the value of the CALC key of the EMPLOYEE record you want to access. Upon

successful execution of the FIND CALC function call, currency is set to the EMPLOYEE record with the employee ID number of 0101. The CALC key for the employee record is the employee ID.

❹ KEYO is set to zero because the employee ID or the CALC key is at offset zero in the employee record. In other words, the employee ID is the first element in the employee record.

❺ KLEN is set to 4, which is the length of the CALC key, the employee ID.

❻ This INPUT statement generates and submits the FIND CALC function call to CA-IDMS. No SAS variables are created. The FIND function establishes currency but does not retrieve data.

❼ FUNCT is set to OBTAIN to generate an OBTAIN CALC function call to CA-IDMS.

❽ SEQ is set to DUP so the code will generate an OBTAIN CALC DUP function call. RECNAME, CKEY, KLEN, and KEYO are still set from the previous FIND CALC function call and do not have to be set.

❾ This INPUT statement contains SAS variables because the OBTAIN function call causes CA-IDMS to locate the specified record and move the data associated with the record to the record buffer.

The INPUT keyword submits the generated function call, which, if successful, returns a record to the buffer. The remaining portion of the INPUT statement maps fields from the buffer to the program data vector.

FIND/OBTAIN CURRENT

The FIND/OBTAIN CURRENT function accesses records by using established currencies. You can FIND or OBTAIN records that are current of the record type, set, or area. You can also use this form of the FIND or OBTAIN function call to establish the appropriate record as current of the run unit.

To generate the FIND|OBTAIN CURRENT OF *<record|set|area>* function call, specify these options:

- FUNC= FIND or OBTAIN
- SEQUENCE= CURRENT

And optionally use one of the following:

- RECORD= a IDMS record name
- SET= an IDMS set name
- AREA= the area in which the record is a participant

The following example shows a FIND CURRENT function call for the DEPARTMENT record. The numbers in the program correspond to the numbered comments following the program.

```
     infile empss01 idms func=funct record=recname
            errstat=stat sequence=seq;
     .
     .
     .
  ❶ funct      = 'FIND';
  ❷ seq        = 'CURRENT';
```

```
❸ recname    = 'DEPARTMENT';
❹ input;
   .
   .
   .
```

❶ FUNCT is assigned the value of FIND.

❷ SEQ is assigned CURRENT so the function call to CA-IDMS will locate the current record of the specified record type, set, or area. In this example, the code is looking for the current record of the record type DEPARTMENT.

❸ RECNAME specifies the name of the record type that is to be accessed. In this example, the record is the DEPARTMENT record.

You can use the AREA option or the SET option instead of the RECORD option with the FIND/OBTAIN CURRENT function to locate the current record of the named area or set, respectively.

❹ This INPUT statement generates and submits the FIND CURRENT function call to CA-IDMS.

FIND/OBTAIN DB-KEY

The FIND/OBTAIN DBKEY function locates a record directly using a db-key that has been stored previously by your DATA step program. The ACCEPT function is used to acquire the record's db-key. Any record in the subschema can be accessed directly using the db-key, regardless of its location mode.

To generate the FIND|OBTAIN DBKEY function call, specify these options:

- FUNC= FIND or OBTAIN
- DBKEY= a db-key value

And optionally:

- RECORD= the IDMS record name

The following example shows an ACCEPT NEXT function call, which acquires the db-key of a record. It is followed by an OBTAIN DBKEY function call, which uses the db-key acquired by the ACCEPT NEXT function call. The numbers in the program correspond to the numbered comments following the program.

```
   infile empss01 idms func=funct dbkey=dkey
          errstat=stat sequence=seq;
   .
   .
   .
❶ funct      = 'ACCEPT';
   seq        = 'NEXT';
❷ dkey       = '    ';
   input;
   .
   .
   .
   funct      = 'OBTAIN';
❸ seq        = '       ';
❹ input @1   department_id      4.0
        @5   department_name    $char45.
```

```
      @50  department_head    4.0;
   .
   .
   .
```

❶ FUNCT is assigned the value of ACCEPT to get the db-key for the next record, based on currency.

❷ DKEY is set to blanks to receive the new db-key.

After the ACCEPT function call has successfully executed, the db-key is returned to the DATA step in the DKEY variable. The db-key can be saved and used later to access the record directly.

❸ The SEQ option is set to blanks because it is not used with the OBTAIN DBKEY function call.

If the RECORD option is used with FIND/OBTAIN DBKEY, the db-key value must contain a db-key of the named record type.

❹ The INPUT statement generates and submits the OBTAIN DBKEY function call. If successful, data returned to the buffer are mapped to the named variables.

FIND/OBTAIN OWNER

The FIND/OBTAIN OWNER function locates the owner record of the current set. This function call can be used to return the owner record of any set, whether or not the set has been assigned owner pointers.

To generate the FIND|OBTAIN OWNER function call, specify these options:

- FUNC= FIND or OBTAIN
- SET= an IDMS set name
- SEQUENCE= OWNER

The following example shows an OBTAIN OWNER function call. This example assumes currency is on an employee record occurrence. The numbers in the program correspond to the numbered comments following the program.

```
   infile empss01 idms func=funct set=inset
          errstat=stat sequence=seq;
   .
   .
   .
❶ funct      = 'OBTAIN';
❷ seq        = 'OWNER';
❸ inset      = 'DEPT-EMPLOYEE';
❹ input @1   department_id      4.0
        @5   department_name    $char45.
        @50  department_head    4.0;
   .
   .
   .
```

❶ FUNCT is assigned the value of OBTAIN so that the data for the owner record is returned to the DATA step program.

❷ SEQ is assigned OWNER to generate an OBTAIN OWNER function call.

❸ INSET specifies the set whose owner record is to be retrieved.

❹ The INPUT statement generates and submits the OBTAIN OWNER function call. If successful, data returned to the buffer are mapped to the named variables.

FIND/OBTAIN WITHIN SET USING SORT KEY

The FIND/OBTAIN SORT KEY function locates a member record in a sorted set. Sorted sets are ordered in ascending and descending sequence based on the sort field value. The search for member records begins with either the current record of the set or the owner of the set. The record that is retrieved will be the first record that has a sort field value that is equal to the value in the SORTFLD SAS variable. If no record matches the SORTFLD value, currency to the next and prior records of the set are maintained so that the DATA step program can traverse the set using the SORTFLD value to perform a generic search.

To generate the FIND|OBTAIN *record* WITHIN *set*|*record* USING *sortfield* function call, specify these options:

- FUNC= FIND or OBTAIN
- SORTFLD= a valid sort field value
- RECORD= a IDMS record name SET= an IDMS set name

To generate the FIND|OBTAIN *record* WITHIN *set*|*record* CURRENT USING *sortfield* function call, include:

- SEQUENCE= CURRENT

The following example shows an OBTAIN *record* WITHIN CURRENT *set* USING *sortfield* function call. The numbers in the program correspond to the numbered comments following the program.

```
     infile empss01 idms func=funct record+recname
            errstat=stat sequence=seq set=inset
            sortfld=skey;
     .
     .
     .
❶ funct      = 'OBTAIN';
❷ seq        = 'CURRENT';
❸ skey       = 'GARFIELD' || 'JENNIFER';
❹ recname    = 'EMPLOYEE';
❺ inset      = 'EMP-NAME-NDX';
❻ input @1   employee_id       4.0
        @5   firstname        $char10.
        @15  lastname         $char15.
        @30  street           $char20.
        @50  city             $char15.
        @65  state            $char2.
        @67  zip              $char9.
        @76  phone            10.0
        @86  status           $char2.
        @88  ssnumber         $char9.
        @97  startdate        6.0
        @103 termdate         6.0
```

```
      @109 birthdate        6.0
      @115 filler01         $char2. ;
.
.
.
```

❶ FUNCT is assigned the value of OBTAIN to retrieve the data for the employee record with the sort key of JENNIFER GARFIELD.

❷ SEQ is set to CURRENT to indicate that the search begins with the current record of the set specified in INSET.

❸ SKEY contains the value of the sort control element to be used in searching the sorted set. In this example, SKEY is set to the last and first name value of the employee name sort control element in the EMP-NAME-NDX set where you want to begin the search.

❹ RECNAME is set to the name of the record to retrieve. In this example, you are looking for the EMPLOYEE record.

❺ INSET is assigned the name of a sorted set.

❻ The INPUT statement generates and submits the OBTAIN SORTFLD WITHIN CURRENT set function call. If successful, data are mapped from the buffer to the named variables.

FIND/OBTAIN WITHIN SET or AREA

The FIND/OBTAIN WITHIN function locates a record either logically, based on set relationships, or physically, based on database location. Using various options with FIND/OBTAIN WITHIN, you can either access each record sequentially in a set or area, or select specific occurrences of a given record within a set or area.

Follow these rules when selecting members *within a set*:

- Currency must be established on a set before attempting to access records in the set.
- The next or prior records in the set are determined by the record that is current of the set named in the SET= option. The set must have prior pointers defined in order to retrieve records using the SEQUENCE= option of PRIOR.
- The first or last record in a set is the first or last member in the logical order of the set. The last record in a set can only be accessed if prior pointers have been established for the set.
- The *n*th record in a set is the set member in the *n*th position of the set. The search for the *n*th member begins with the owner of the current set and continues until the *n*th record is located or until an end-of-set condition occurs. If the *n*th number is negative, the search uses prior pointers. To use negative numbers, prior pointers must have been established for the set.
- When an end-of-set occurs, the owner of the set becomes the current record of the run-unit, the record type, its area, and its set.

Follow these rules when selecting records *within an area*:

- The first record within an area is the record with the lowest db-key. The last record within an area is the record with the highest db-key.
- The next record within an area is the record with the next highest db-key in relationship to the record which is current of the named area. The prior record works the same way, except the prior record is the record with the next lowest db-key.

- Before the next or prior record within an area can be requested, the first, last, or *n*th record within an area must be accessed to correctly establish a starting position within the area.

To generate the FIND | OBTAIN NEXT | PRIOR | FIRST | LAST | *n*th <*record*> WITHIN *set*|*area* function call, specify:

- FUNC= FIND or OBTAIN

And one of these options:

- SET= an IDMS set name
- AREA= the area that the record participates in
- SEQUENCE= NEXT|PRIOR|FIRST|LAST|*n*th

And optionally:

- RECORD= a IDMS record name

The following example shows an OBTAIN PRIOR *record* WITHIN AREA function call. Currency has already been established on an EMPLOYEE record. The numbers in the program correspond to the numbered comments following the program.

```
     infile empss01 idms func=funct area=subarea
            record=recname errstat=stat
            sequence=seq;
     .
     .
     .
❶    funct      = 'OBTAIN';
❷    seq        = 'PRIOR';
❸    subarea    = 'EMP-DEMO-REGION';
     recname    = 'EMPLOYEE'
❹    input @1   employee_id    4.0
           @5   firstname        $char10.
           @15  lastname         $char15.
           @30  street           $char20.
           @50  city             $char15.
           @65  state            $char2.
           @67  zip              $char9.
           @76  phone            10.0
           @86  status           $char2.
           @88  ssnumber         $char9.
           @97  startdate        6.0
           @103 termdate         6.0
           @109 birthdate        6.0
           @115 filler01         $char2. ;
     .
     .
     .
```

❶ FUNCT is assigned the function of OBTAIN to retrieve the data for the EMPLOYEE record.

❷ SEQ is set to PRIOR to indicate that the prior EMPLOYEE record is requested.

❸ SUBAREA contains the name of the current area from which to retrieve the EMPLOYEE record.

❹ The INPUT statement generates and submits the OBTAIN PRIOR function call. If successful, data are mapped from the buffer to the named variables.

GET

The GET statement moves the record that is current of the run unit into the input buffer. The GET function is used in conjunction with the FIND function. The FIND function locates records in the database without moving the data associated with the record to the record buffer.

To generate the GET <*record-name*> function call, specify:

- FUNC= GET

And optionally:

- RECORD= the IDMS record name

The following example shows the GET function call with no other options.

```
infile empss01 idms func=func1 record=rec1
       errstat=err;
.
.
.
❶ func1      = 'GET';
❷ input @1   department_id    4.0
      @5   department_name   $char45.
      @50  department_head   4.0;
.
.
.
```

❶ FUNC1 is assigned the value of GET.

❷ The record that is current of the run unit is moved into the input buffer. Currency must be established before issuing the GET function.

The following example shows the GET function call for the DEPARTMENT record.

```
infile empss01 idms func=func1 record=rec1
       errstat=err;
.
.
.
func1      = 'GET';

❶ rec1       = 'DEPARTMENT';
input @1   department_id      4.0
      @5   department_name    $char45.
      @50  department_head    4.0;
.
.
.
```

❶ The difference between this GET function call and the previous GET call is the use of the SAS variable REC1. This variable is set to the name of the specific record to move into the record buffer. In this example, the data associated with the DEPARTMENT record is moved. Currency must be established on the DEPARTMENT record before a GET call can be made for the record.

IF

The DML IF statement tests for the existence or membership of a record occurrence in a named set occurrence, and returns the result in the ERRSTAT variable.

There are two formats for the DML IF statement:

- IF SET <NOT> EMPTY tests for the existence of a record occurrence and returns a status value of 0000 if the set occurrence is empty, and a status value of 1601 if the set occurrence is not empty.
- IF <NOT> SET MEMBER checks the membership of the current record occurrence and returns a status value of 0000 if the record occurrence is a member of the named set occurrence, and a status value of 1608 if the record occurrence is a non-member.

To issue the DML IF statement, specify these options:

- FUNC= IF
- INSET= an IDMS set name
- SEQUENCE= EMPTY|NEMPTY|MEMBER| NMEMBER

The following is an example of a DML IF function call.

```
infile empss01 idms func=funct record=recname
       area=subarea errstat=stat sequence=seq
       set=inset;

❶ funct   = 'FIND';
  seq     = 'FIRST';
  recname = 'DEPARTMENT';
  subarea = 'ORG-DEMO-REGION';
  input;
  if (stat ^= '0000') then go to staterr;

❷ funct   = 'IF';
❸ seq     = 'NEMPTY';
❹ inset   = 'DEPT-EMPLOYEE';
  recname = '                ';
  subarea = '                ';
  input;

❺ if (stat = '1601') then do;
  put @1 'Set ' @5 inset @14 'is not empty';
  stat    = '0000';
  _error_ = 0;
  end;
❻ else if (stat = '0000') then
  put @1 'Set' @5 inset @14 'is empty';
  else go to staterr;
  stop;
```

❶ Run-unit currency for the DML IF statement is established by the previous function call. Here, a FIND function call establishes run-unit currency on the record DEPARTMENT for the DML IF statement, but does not retrieve the record.

❷ FUNCT is assigned the value of IF to indicate that a test will be performed. Set currency is determined by the owner of the current record in the set named in INSET.

❸ SEQ is set to NEMPTY to indicate the type of test.

❹ INSET names the set to test.

❺ The first SAS IF statement directs the DATA step to write a message to the log if the value of STAT is 1601, which means that the set is not empty.

❻ The second SAS IF statement directs the DATA step to stop if the value of STAT is 0000, which means the set is empty.

RETURN

The RETURN function retrieves the db-key and the symbolic key for an indexed record without retrieving the record's data. This function establishes currency on the index set.

There are two formats for the RETURN function:

- The RETURN CURRENCY function retrieves the db-key and symbolic key for an index entry based on established currencies or its position in the index set.
- The RETURN USING SORTKEY function retrieves the db-key and symbolic key associated with a specific index key entry.

To generate the RETURN CURRENCY <*set*> NEXT |PRIOR|FIRST|LAST INTO DBKEY *key* INTO SORTKEY *skey* function call, specify these options:

- FUNC= RETURN
- SET= an IDMS index set name
- SEQUENCE= FIRST|LAST|NEXT|PRIOR
- SORTFLD= upon successful completion of the function call, this SAS variable will contain the current record's symbolic key.
- DBKEY= upon successful completion of the function call, this SAS variable will contain the current record's db-key.

The following example shows the RETURN FIRST function call.

```
   infile empss01 idms func=func1 errstat=err
          sequence=seq set=inset sortkey=skey dbkey=dkey;

   .
   .
❶ func1      = 'RETURN';
❷ seq        = 'FIRST';
❸ inset      = 'EMP-NAME-NDX';
   input;
❹ put @1 'DBKEY OF RECORD = '  @19 dkey;
   put @1 'SKEY OF RECORD = '  @19 skey;
   .
   .
```

```
.
```

❶ FUNC1 is assigned the function of RETURN.

❷ SEQ is assigned the value of FIRST. FIRST returns the db-key for the first index entry in the set EMP-NAME-NDX. You could also request the db-key from the PRIOR, FIRST, or LAST index entry in the set by assigning these values to the SEQUENCE= option.

❸ SET is assigned the name of the index set (INSET) from which the specified db-key is to be returned.

❹ DKEY will contain the db-key for the first entry in EMP-NAME-NDX. SKEY will contain the symbolic key for the entry. The PUT statements print the db-key and the symbolic key on the SAS log.

To generate the RETURN USING SORTKEY <*set*> INTO DBKEY *key* INTO SORTKEY *skey* function call, specify these options:

- FUNC= RETURN
- SEQUENCE= USING
- SET= an IDMS set name
- SORTKEY= the index key entry to search for. After successful completion of the function call, this SAS variable will contain the record's symbolic key.
- DBKEY= upon successful completion of the function call, this SAS variable will contain the record's db-key.

The following example shows the RETURN USING function call.

```
infile empss01 idms func=func1 record=recname
       ikeylen=keyl errstat=err sequence=seq
       set=inset dbkey=dkey sortkey=skey;
.
.
.
❶ func1      = 'RETURN';
❷ seq        = 'USING';
❸ inset      = 'EMP-NAME-NDX';
❹ skey       = 'GARFIELD  JENNIFER';
❺ keyl       =  25;
❻ dkey       =  '  ';
  input;
.
.
.
```

❶ FUNC1 is assigned the function of RETURN.

❷ SEQ is set to USING to indicate that the index key entry in SKEY will be used to locate the db-key. In this example, SKEY is set to the last name and first name GARFIELD JENNIFER. The call will return the db-key and symbolic key of the first record it encounters which contains the name GARFIELD JENNIFER.

❸ INSET is the name of the index set to be searched.

❹ SKEY specifies the index key value to search for.

❺ KEYL specifies the length of index key value.

❻ DKEY is set to blanks to receive the db-key.

After the RETURN function call has successfully executed, the db-key is returned to the DATA step in the DKEY variable.

Summary of Options Needed to Generate CA-IDMS Function Calls

Table 2.2 on page 29 outlines the SAS INFILE parameters that are required to generate each of the CA-IDMS function calls for COBOL DML.

Table 2.2 Options Needed to Generate Function Calls for COBOL DML

COBOL DML Call	**INFILE Statement Options**				
ACCEPT db-key FROM CURRENCY	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	ACCEPT	CURRENT	-	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
ACCEPT db-key FROM record-name CURRENCY	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	ACCEPT	CURRENT	Required	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
ACCEPT db-key FROM set-name CURRENCY	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	ACCEPT	CURRENT	-	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
ACCEPT db-key FROM area-name CURRENCY	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	ACCEPT	CURRENT	-	-	Required
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
ACCEPT db-key FROM set-name NEXT|PRIOR|OWNER CURRENCY	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	ACCEPT	NEXT PRIOR OWNER	-	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
BIND record-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	-	BIND	-	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	

COBOL DML Call	INFILE Statement Options				
FIND/OBTAIN CALC* record-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	-	Required	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	Required	Required	
FIND/OBTAIN DUPLICATE* record-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	DUP	Required	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	Required	Required	
FIND/OBTAIN CURRENT	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	CURRENT	-	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
FIND/OBTAIN CURRENT record-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	CURRENT	Required	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
FIND/OBTAIN CURRENT|NEXT|PRIOR|FIRST|LAST|Nth WITHIN set-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	NEXT PRIOR FIRST LAST Nth	Optional	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
FIND/OBTAIN CURRENT|NEXT|PRIOR|FIRST|LAST|Nth WITHIN area-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	NEXT PRIOR FIRST LAST Nth	Optional	-	Required
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
FIND/OBTAIN OWNER WITHIN set-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	OWNER	-	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
FIND/OBTAIN record-name WITHIN set-name USING sort-key	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	-	Required	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	

COBOL DML Call	INFILE Statement Options				
FIND/OBTAIN record-name WITHIN set-name CURRENT USING sort-key	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	CURRENT	Required	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
FIND/OBTAIN DBKEY db-key	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	-	-	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
FIND/OBTAINrecord-name DB-KEY IS db-key	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	FIND OBTAIN	-	Required	-	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	-	-	-	
GET record-name	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	-	GET	-	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	-	-	-	-	
RETURN db-key FROM index-set-name CURRENT|FIRST|LAST|NEXT| PRIOR KEY INTO symbolic-key	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	RETURN	CURRENT FIRST LAST NEXT PRIOR	-	Required	-
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	Required	-	-	
RETURN db-key FROM index-set-name USING index-key-value KEY INTO symbolic-key	*FUNC*	*SEQUENCE*	*RECORD*	*SET*	*AREA*
	RETURN	USING	-	Required	
	DBKEY	*SORTFLD*	*IKEY*	*IKEYLEN*	
	Required	Required	-	-	

* KEYOFF= INFILE statement option required for these calls

How the CA-IDMS Function Call Is Generated

To determine which type of DML function call you want to generate, the CA-IDMS DATA step access method must make some assumptions from the various options that you specify. The access method first determines what value is specified in the FUNC option.

- If the FUNC option contains BIND, GET, ACCEPT, or RETURN, the required options are checked for a value, then the optional options are checked, and the appropriate function call is generated.
- If the FUNC option contains FIND or OBTAIN, the access method checks whether a value was entered for the following options:

SORTFLD
If the SORTFLD option was entered, the required and optional options for the OBTAIN or FIND with the SORTFLD are verified before a function call is generated. If the SORTFLD option was not entered, the access method then determines if the IKEY option was entered to generate a function call using the CALC key.

IKEY
If the IKEY option was entered, then all of the required and optional options are verified for a function call using the CALC key. If the IKEY option was not entered, the access method then looks to see if the DBKEY option was entered.

DBKEY
If the DBKEY was entered, the same verification is done for the options as before and a function call is generated. If DBKEY was not entered, then the access method looks to see if the SEQUENCE option was entered.

SEQUENCE
If a value was entered for the SEQUENCE option, the value is examined. If the value is

CURRENT
The other options are checked to determine what type of currency call to generate.

OWNER
An OBTAIN or FIND OWNER or a FIND DUP OWNER function call is generated.

NEXT, PRIOR, FIRST, LAST, or *n*th
The access method tries to generate an OBTAIN or FIND WITHIN function call by using the other options that were entered.

If the access method cannot generate a function call from the options that you entered or if the options for a particular function call are incorrect, an error message is returned, the automatic variable _ERROR_ is set to 1, and the CA-IDMS call status is set to 9999. Your DATA step program should check for these conditions after each function call to the database.

Using Multiple Sources of Input

You can have more than one input source in a DATA step. For example, you can read from a CA-IDMS database and a SAS data set in the same DATA step. You cannot, however, read from more than one subschema in a single DATA step. If you want to use several external files (MVS data sets) in a DATA step, use separate INFILE statements for each source.

The input source is set (or reset) when an INFILE statement is executed. The file or CA-IDMS subschema referenced in the most recently executed INFILE statement is the *current input source* for INPUT statements. The current input source does not change until a different INFILE statement executes, regardless of the number of INPUT statements executed.

If after you change input sources by executing multiple INFILE statements you want to return to an earlier input source, it is not necessary to repeat all options specified in the original INFILE statement. The SAS System remembers options from the first INFILE statement with the same fileref or subschema name. In a standard INFILE statement, you need only specify the fileref. In a CA-IDMS INFILE statement, specify

the subschema and IDMS. Options specified in a previous INFILE statement with the same fileref or subschema name cannot be altered.

Note: The subschema name cannot be the same name as a fileref on a JCL DD statement, a TSO ALLOC statement, or a filename's fileref for the current execution of the SAS System. △

The CA-IDMS INPUT Statement

If you are unfamiliar with the INPUT statement, refer to *SAS Language Reference: Dictionary* for more information.

An INPUT statement reads from the file specified by the most recently executed INFILE statement. If the INFILE statement is a CA-IDMS INFILE statement, the INPUT statement issues a CA-IDMS function call as formatted by variables specified in the INFILE statement.

There are no special options for the CA-IDMS INPUT statement as there are for the CA-IDMS INFILE statement. The form of the CA-IDMS INPUT statement is the same as that of the standard INPUT statement:

INPUT <*specification-1* > <*...specification-n* > <@|@@ >;

For example, suppose you issue an OBTAIN function call for the EMPLOYEE record. The CA-IDMS INPUT statement might be coded as follows:

```
input @1   employee_id      4.0
      @5   firstname        $char10.
      @15  lastname         $char15.
      @30  street           $char20.
      @50  city             $char15.
      @65  state            $char2.
      @67  zip              $char9.
      @76  phone            10.0
      @86  status           $char2.
      @88  ssnumber         $char9.
      @97  startdate        8.0
      @105 termdate         8.0
      @113 birthdate        8.0;
```

When this CA-IDMS INPUT statement executes, the DATA step interface generates and submits a function call from the options you entered on the CA-IDMS INFILE statement. If the FUNC= variable specified in the INFILE statement is assigned a value of GET or OBTAIN, an EMPLOYEE record is retrieved and placed in the input buffer. Data for the variables specified in the CA-IDMS INPUT statement are then moved from the input buffer to SAS variables in the program data vector.

Depending on which options you specify in the CA-IDMS INFILE statement and which form of the CA-IDMS INPUT statement you use, the INPUT statement will do one of the following:

- retrieve a record from the database, place it into the input buffer without moving any variables into the program data vector, and possibly hold the record for the next INPUT statement. If the FUNC= variable specifies GET or OBTAIN, but the INPUT statement does not list any variables, then data are placed into the input buffer without being moved into the program data vector. If the INPUT statement specifies a trailing @ or @@, the record is held for processing by the next INPUT statement. See "Using the Null INPUT Statement" on page 34 and "Holding Records in the Input Buffer" on page 35 for more information.

- retrieve a record from the database, place it into the input buffer, move data from the input buffer into variables in the program data vector, and possibly hold the record for the next INPUT statement. If the FUNC= variable specifies GET or OBTAIN, and the INPUT statement specifies one or more variables, then data are placed into the input buffer and mapped into variables in the program data vector. If the INPUT statement specifies a trailing @ or @@, the record is held for processing by the next INPUT statement. See "Holding Records in the Input Buffer" on page 35 for more information.
- submit a DBMS request without retrieving a record. If the FUNC= variable specifies BIND, FIND, ACCEPT or RETURN, then no record data are retrieved from the database. These functions are described in "Specifying DML Function Calls" on page 14. See "Using the Null INPUT Statement" on page 34 for more information.
- release a previously held record from the input buffer. If the previous INPUT statement specified a trailing @ or @@, and the current INPUT statement is a null INPUT statement (**`input;`**), then the previously held record is released. See "Holding Records in the Input Buffer" on page 35 for more information.

Note: Every time the SAS System encounters a CA-IDMS INPUT statement, it increments by one an internal counter that keeps track of how many function calls are issued from the input data set. The count is printed to the SAS log as a NOTE. Because you may have coded several CA-IDMS INPUT statements that do not retrieve data, this count may not accurately reflect the actual number of records retrieved from the database. △

Although the syntax of the CA-IDMS INPUT statement and the standard INPUT statement are the same, your use of the CA-IDMS INPUT statement is often different. Suggested uses of the CA-IDMS INPUT statement are described in the following sections.

Using the Null INPUT Statement

When an INPUT statement does not specify any variable names or options, it is called a null INPUT statement:

```
input;
```

A null INPUT statement serves three purposes:

- A null CA-IDMS INPUT statement generates and submits a CA-IDMS function call to the database. To issue a CA-IDMS function call that does not retrieve data (FIND, ACCEPT, RETURN, and BIND), use a null INPUT statement.
- A null CA-IDMS INPUT statement retrieves a record from the database and places it in the input buffer, but does not move data values to the program data vector. When you want to issue an OBTAIN or GET function call, you can use the INPUT statement with a trailing '@' or '@@' to retrieve a record from the database, then check the status code returned from CA-IDMS before moving data values to the program data vector.
- If the previous INPUT statement was **`input @;`** or **`input var1 var2 var3 @;`**, a null INPUT statement releases the previously held record. See "Holding Records in the Input Buffer" on page 35 for information.

Holding Records in the Input Buffer

The trailing @ and @@ pointer controls tell the SAS System to hold the current record in the input buffer so that it can be processed by a subsequent INPUT statement. The trailing @ tells the SAS System to hold the record for the next INPUT statement in the same iteration of the DATA step. The double trailing @ tells the SAS System to hold the record for the next INPUT statement across iterations of the DATA step.

Assuming the FUNC= variable in your INFILE statement specifies GET or OBTAIN, the following INPUT statement submits a function call to the database, retrieves a record from the database, places it in the input buffer, and places a hold on the buffer:

```
input @;
```

The next INPUT statement that is executed does not issue another function call and does not place a new record in the input buffer. Instead, the second INPUT statement uses the data placed in the input buffer by the first INPUT statement.

If your INPUT statement also specifies variable names, then that statement issues a function call to the database, retrieves a record, places the record into the input buffer, and moves data values for the named variables into the program data vector:

```
input ssnumber $char11. @;
```

The SAS System holds the record in the input buffer for use with the next INPUT statement.

If you have used an INPUT statement with a trailing @ or @@, and you now want to release the record from the input buffer, use a null INPUT statement as described in "Using the Null INPUT Statement" on page 34.

Checking Call Status Codes

For each function call issued, CA-IDMS returns a call status code that indicates whether or not the function call was successful. Because the success of a function call can affect the remainder of the program, you should check call status codes after every call to CA-IDMS. SAS provides the automatic SAS variable _ERROR_, whose values indicate the success of a function call.

Table 2.3 on page 35 shows the _ERROR_ values and their meaning.

Table 2.3 Summary of _ERROR_ Values

Value of _ERROR_	Possible Corresponding Status Codes	Description
0	CA-IDMS 0000	Function call executed successfully
1	All CA-IDMS status codes except 0000	CA-IDMS error code returned. Contents of the input buffer and the program data vector are printed in the SAS log with the next INPUT statement or when control returns to the beginning of the DATA step, whichever comes first.
	SAS status 9999	Program cannot perform function call from options specified.

Obtaining the Value of _ERROR_

Check the SAS log to see the value of _ERROR_. If _ERROR_=1, it is printed in the SAS log along with the contents of the input buffer and the program data vector.

Obtaining the CA-IDMS Error Codes

You can obtain the status code returned by CA-IDMS by specifying a variable name with the ERRSTAT= option of the CA-IDMS INFILE statement. This variable will be assigned the CA-IDMS status after each function call to the database.

Refer to your CA-IDMS documentation for explanations of CA-IDMS error status codes.

Checking for Non-Error Conditions and Resetting _ERROR_

Some of the CA-IDMS status codes that set _ERROR_ to 1 might not represent errors in your SAS program. When this happens in your application, you should check the actual error status code returned by CA-IDMS as well as the value of _ERROR_ by the methods stated in the above sections, and possibly reset _ERROR_ to 0.

For example, suppose you are writing a program that accesses all the DEPARTMENT and EMPLOYEE records from all the DEPT-EMPLOYEE set occurrences. When an end-of-set condition (CA-IDMS status code 0307) occurs on the EMPLOYEE record, _ERROR_ is set to 1; however, you do not consider the end-of-set condition to be an error. Instead, you want your application to obtain the next owner record or DEPARTMENT record from the next DEPT-EMPLOYEE set occurrence.

If a status code sets _ERROR_ but you do not consider the condition to be an error, you should reset _ERROR_ to 0 before executing another INPUT statement or returning to the beginning of the DATA step. Otherwise, the contents of the input buffer and program data vector are printed on the SAS log. See ❻ in “Example: Traversing a Set” on page 37 for an example of how to reset _ERROR_ to 0.

Catching Errors Before Moving Data

In all programs it is important to check the values of either the _ERROR_ or ERRSTAT= variables before moving data from the input buffer into the program data vector. For example, if a GET or OBTAIN function call fails to retrieve the expected record, the input buffer might still contain data from a previous GET or OBTAIN call or be filled with missing values. You might not want to move these values to SAS variables. By checking either the ERRSTAT= or _ERROR_ variable, you can determine whether the function call was successful and decide whether to move the input buffer data to SAS variables.

When you need to issue a retrieval call but you want to check either _ERROR_ or ERRSTAT= values before moving data to SAS variables, use a CA-IDMS INPUT statement with no variables specified, but with a trailing @, to issue the call:

```
input @;
```

Because no variables are specified, no data are moved to the program data vector. The statement contains a trailing @, so the record remains in the input buffer, and your application can check the values in _ERROR_ and/or ERRSTAT= before determining what action to take. For more information, see “Holding Records in the Input Buffer” on page 35.

For example, suppose you have specified ERRSTAT=ERR and FUNC=FUNC1 on your INFILE statement, and you have assigned FUNC1= 'GET' or 'OBTAIN'. You can use the following code to check the error status before moving data:

```
❶ input @;
❷ if (err ne ’0000’ and err ne ’0307’) then
        go to staterr;
❸ if err eq ’0307’ then do;
❹    _error_ = 0;
     /* No more DEPT records so STOP */
```

```
      stop;
   end;
❺ input @1   department_id      4.0
        @5   department_name    $char45.
        @50  department_head    4.0;
```

❶ The INPUT statement retrieves a record from the database and places a hold on the input buffer but does not move data to the program data vector.

❷ A SAS IF statement checks to see if ERR is not equal to 0000 or 0307. If not, the program branches to the STATERR routine, which issues an error message and stops the DATA step.

❸ If the INPUT statement encountered the end-of-set, then the _ERROR_ variable is reset to 0 (❹) to prevent the contents of the input buffer and program data vector from being printed on the SAS log, and the DATA step stops.

❺ If the first INPUT statement (❶) was successful, then the second INPUT statement moves the data from the record being held in the input buffer to the program data vector and releases the hold.

Handling End of File

Because of the nature and design of a network database, the concept of an *end of file* does not exist. Consequently, the SAS option EOF= should not be used on a CA-IDMS INFILE statement. Instead you should either code your DATA step to stop processing when you have retrieved all the records you need or set up your code to loop, stopping only when it reaches a desired condition.

Example: Traversing a Set

The following DATA step shows how to traverse the DEPT-EMPLOYEE set using the CA-IDMS INFILE and CA-IDMS INPUT statements. The numbers in the program correspond to the numbered comments following the program.

```
❶ data work.dept_employee;

❷ infile empss01 idms func=func1 record=recname
         area=iarea sequence=iseq errstat=err
         set=iset;

   /* BIND the DEPARTMENT and EMPLOYEE       */
   /* records in the first data set          */
   /* iteration;  if successful, then        */
   /* OBTAIN FIRST DEPARTMENT WITHIN AREA    */

❸ if _n_ = 1 then do;
      func1      = 'BIND';
      recname    = 'DEPARTMENT';

❹    input;
      if (err ne '0000') then go to staterr;
```

```
      recname  = 'EMPLOYEE';
      input;
      if (err ne '0000') then go to staterr;

      /* Get a DEPARTMENT record */

      iseq       = 'FIRST';
      func1      = 'OBTAIN';
      recname    = 'DEPARTMENT';
      iarea      = 'ORG-DEMO-REGION';
   end;

   else do;
      func1      = 'FIND';
      iseq       = 'OWNER';
      input;
      if (err ne '0000') then go to staterr;
      func1      = 'OBTAIN';
      iseq       = 'NEXT';
      recname    = 'DEPARTMENT';
      iarea      = 'ORG-DEMO-REGION';
      iset       = ' ';
   end;

   /* OBTAIN DEPT record and test     */
   /* for success */

❺ input @;
❻ if (err ne '0000' and err ne '0307') then
     go to staterr;
   if err eq '0307' then do;
      _error_ = 0;
      /* No more DEPT records so STOP */
      stop;
   end;
❼ input @1   department_id    4.0
        @5   department_name  $char45.
        @50  department_head  4.0;

   /* Get the EMPLOYEE records for this DEPT */
   /* record */

   iseq       = 'FIRST';
   recname    = 'EMPLOYEE';
   iset       = 'DEPT-EMPLOYEE';
   iarea      = ' ';
   do until (err = '0307');

     /* OBTAIN EMPLOYEE records and test for */
     /* SUCCESS */

     input @;
     if (err ne '0000' and err ne '0307') then
        go to staterr;
```

```
   if err = '0000' then do;
      input @1   employee_id    4.0
            @5   firstname      $char10.
            @15  lastname       $char15.
            @30  street         $char20.
            @50  city           $char15.
            @65  state          $char2.
            @67  zip            $char9.
            @75  phone          10.0
            @85  status         $char2.
            @87  ssnumber       $char9.
            @96  startdate      8.0
            @104 termdate       8.0
            @112 birthdate      8.0;
❽           output;
❾        iseq = 'next';
    end;
end;
_error_ = 0;
return;

staterr:
   put @1 'WARNING: ' @10 func1 @17
          'RETURNED ERR ='@37 err;
   stop;
run;

❿ proc print data=work.dept_employee;
title1 'This is an Area Sweep of the
        DEPT-EMPLOYEE Set';
title2 'The Area Sweep is from Beginning to End';
run;
```

❶ The DATA statement references a temporary SAS data set called DEPT_EMPLOYEE, which is to be opened for output.

❷ The INFILE statement tells SAS to use the EMPSS01 subschema. The IDMS option tells the SAS System that EMPSS01 is a CA-IDMS subschema instead of a fileref. The statement also tells the DATA step interface to use the SAS variables as follows:

- FUNC1 to contain the function type
- RECNAME to contain the record name
- IAREA to contain the area name
- ISEQ to contain the function call sequence information
- ISET to contain the set name.

The statement also tells the interface to store the call status in ERR.

❸ All record types to be retrieved must first be bound to CA-IDMS. The BIND function call need only be issued once per record type prior to retrieval. The automatic SAS System variable _N_ is used to indicate the first iteration of the DATA step code.

❹ The INPUT statements generate and submit the function call to CA-IDMS requesting that a BIND be performed for the record type specified in RECNAME. In this example, the DEPARTMENT record type is bound first, then the EMPLOYEE record type is bound.

❺ This INPUT statement also uses the values in the SAS variables FUNC1 and RECNAME, along with the values in ISEQ and IAREA to generate an OBTAIN FIRST DEPARTMENT RECORD IN AREA ORG-DEMO-REGION DML call. However, no data are moved into the program data vector because no variables are defined on the **`INPUT @;`** statement. This function call allows the DATA step to check the status that is returned from CA-IDMS before moving data into the program data vector. This function call is issued only on the first iteration of the DATA step. On subsequent iterations, the values in these SAS variables are used to generate an OBTAIN NEXT DEPARTMENT RECORD IN AREA ORG-DEMO-REGION DML call.

❻ The program examines the status code returned by CA-IDMS. If CA-IDMS returns 0000, then the program proceeds to the next statement. If CA-IDMS returns 0307 (end of set), then there are no more department records and the DATA step stops.

❼ When this INPUT statement executes, DEPARTMENT RECORD data are moved from the program data vector into the SAS buffer.

❽ As the DATA step executes, EMPLOYEE records that are members of the DEPT-EMPLOYEE set are retrieved, and observations that contain the EMPLOYEE data are written to the DEPT_EMPLOYEE data set.

❾ The ISEQ value is changed to NEXT to generate an OBTAIN NEXT EMPLOYEE RECORD IN SET DEPT-EMPLOYEE DML call.

❿ The PRINT procedure prints the list of DEPARTMENT and EMPLOYEE records.

Output 2.3 on page 40 shows the SAS log for this example.

Output 2.3 SAS Log

```
1          data work.dept_employee(drop=filler);
2          infile empss01 idms func=func1
3                                record=recname
4                                area=iarea
5                                sequence=iseq
6                                errstat=err
7                                set=iset;
           .
           .
           .
91         run;
NOTE: The infile EMPSS01 is:
      Subschema=EMPSS01
NOTE: 86 records were read from the infile EMPSS01.
      The minimum record length was 0.
      The maximum record length was 116.
NOTE: The data set WORK.DEPT_EMPLOYEES has 56
      observations and 16 variables.
NOTE: The DATA statement used 0.37 CPU seconds
      and 2709K.
92         proc print data=work.dept_employees;
93         title1 'This is an Area Sweep of the
                  DEPT-EMPLOYEE Set';
94         title2 'The Area Sweep is from the
                  Beginning to End';
95         run;
NOTE: The PROCEDURE PRINT printed pages 1-3.
```

Output 2.4 on page 41 shows a portion of the output of this example.

Output 2.4 Area Sweep of DEPT-EMPLOYEE Set

```
                              This is an Area Sweep of the DEPT-EMPLOYEE Set
                              The Area Sweep is from the Beginning to End

      department_                             department_  employee_
Obs      id          department_name            head          id        firstname   lastname    street
  1     2000         ACCOUNTING AND PAYROLL     11           69         JUNE        BLOOMER    14 ZITHER TERR
  2     2000         ACCOUNTING AND PAYROLL     11           100        EDWARD      HUTTON     781 CROSS ST
  3     2000         ACCOUNTING AND PAYROLL     11           11         RUPERT      JENSON     999 HARVEY ST
  .        .         .                          .            .          .           .           .
  .        .         .                          .            .          .           .           .
  .        .         .                          .            .          .           .           .
 24     5100         BRAINSTORMING              15           15         RENE        MAKER      10 DROVER DR
 25     5100         BRAINSTORMING              15           341        RICHARD     MUNYON     17 BLACKHILL DR
 26     5100         BRAINSTORMING              1            458        RICHARD     WAGNER     677 GERMANY LN

Obs    city          state     zip       phone      status     ssnumber    startdate   termdate    birthdate
  1    LEXINGTON      MA      01675    617555554      40      103955781     880050      500000       60042
  2    MELROSE        MA      02176    617665101      00      101122333     377090      700000       41030
  3    MELROSE        MA      02176    617665555      60      102234789     180092      900000       48081
  .       .           .        .       .              .       .             .           .            .
  .       .           .        .       .              .       .             .           .            .
  .       .           .        .       .              .       .             .           .            .
 24    BOSTON         MA      02123    617452141      40      101067334     378010      200000       45052
 25    WESTWOOD       MA      02090    617329001      70      111100208     180111      400000       50121
 26    NATICK         MA      02178    617432110      90      101177666     378060      700000       34030


                              This is an Area Sweep of the DEPT-EMPLOYEE Set
                             The Area Sweep is from the Beginning to End

      department_                             department_  employee_
Obs      id          department_name            head          id        firstname   lastname     street
 27    1000          PERSONNEL                  13           81         TOM         FITZHUGH    450 THRUWAY ST
 28    1000          PERSONNEL                  13           51         CYNTHIA     JOHNSON     17 MANIFESTO DR
 29    1000          PERSONNEL                  13           91         MADELINE    ORGRATZI    67 RAINBOW DR
  .    .             .                          .            .          .       .           .
  .    .             .                          .            .          .       .           .
  .    .             .                          .            .          .       .           .
 50    3100         INTERNAL SOFTWARE          3            35         LARRY       LITERATA    123 SATURDAY TERR
 51    3100         INTERNAL SOFTWARE          3            23         KATHERINE   O'HEARN     12 EAST SPEEN ST
 52    3100         INTERNAL SOFTWARE          3            21         RALPH       TYRO        888 FORTITHE ST

Obs    city          state     zip       phone      status     ssnumber    startdate   termdate    birthdate
 27    MANSFIELD      MA      03458    617882012      30      111234567     881091      900000       56021
 28    WALPOLE        MA      02546    617777888      80      501134787     877032      300000       45010
 29    KENDON         MA      06182    617431191      90      123106787     880101           0       51101
  .       .           .        .          .           .          .           .           .            .
  .       .           .        .          .           .          .           .           .            .
  .       .           .        .          .           .          .           .           .            .
 50    WILMINGTON     MA      02476    617591232      30      102356783     180090      900000       55043
 51    NATICK         MA      02364    617889713      40      101955671     278050      400000       54040
 52    SINGER         MA      02254    617445919      10      101989345     680122      100000       55122
```

Example: Using the Trailing @ and the INPUT with No Arguments

This example shows the use of the trailing @ and the INPUT statement with no arguments. This DATA step creates a SAS data set, DEPT5100, from data in the EMPLOYEE records in department number 5100. The subschema used defines the DEPARTMENT and the EMPLOYEE record with all their elements.

The example starts by issuing a BIND on the DEPARTMENT record and the EMPLOYEE record. This CA-IDMS call is required for each record that will be retrieved, but the BIND function itself does not retrieve any data. To generate these calls, a null INPUT statement is used. The same thing is done with the FIND CALC

DEPARTMENT call. Once again, this call does not retrieve any data so the null INPUT statement is used.

Each OBTAIN call is issued by a CA-IDMS INPUT statement with a trailing @, so the retrieved record is placed in the buffer and held there. The ERR variable is checked. If a call results in an error, the job terminates. If a call is successful, another CA-IDMS INPUT statement moves the data to SAS variables in the program data vector, and the observation is written to the appropriate SAS data set. Output 2.5 on page 44 shows the output of this example.

```
data work.dept5100(drop=filler);
infile empss01 idms func=func1 record=recname
        sequence=iseq errstat=err ikey=ckey
        ikeylen=keylen keyoff=offset set=iset;

/* BIND the DEPARTMENT and EMPLOYEE       */
/* records; then, if successful           */
/* OBTAIN FIRST DEPARTMENT WITHIN AREA    */

func1    = 'BIND';
recname  = 'DEPARTMENT';
input;
if (err ne '0000') then go to staterr;
recname  = 'EMPLOYEE';
input;
if (err ne '0000') then go to staterr;

/* FIND DEPT record with CALC key 5100    */

func1      = 'FIND';
recname    = 'DEPARTMENT';
ckey       = '5100';
keylen     = 4;
offset     = 0;
input;
if (err ne '0000') then go to staterr;

/* Reset the options for the next call */

func1      = 'OBTAIN';
recname    = 'EMPLOYEE';
ckey       = '    ';
keylen     = 0;
offset     = 0;
iseq       = 'FIRST';
iset       = 'DEPT-EMPLOYEE';

do while (err = '0000');

   /*  OBTAIN EMPLOYEE records and test  */
   /*  for success */

   input @;
   if (err ne '0307' and err ne '0000') then
       go to staterr;
   if (err eq '0307') then do;
```

```
      _error_ = 0;
      stop;
   end;
   input @1   employee_id     4.0
         @5   firstname       $char10.
         @15  lastname        $char15.
         @30  street          $char20.
         @50  city            $char15.
         @65  state           $char2.
         @67  zip             $char9.
         @75  phone           10.0
         @85  status          $char2.
         @87  ssnumber        9.0
         @96  startdate       8.0
         @104 termdate        8.0
         @112 birthdate       8.0;
   output;
   iseq = 'NEXT';
end;
staterr:
   put @1 'ERROR: ' @10 func1 @17
          'returned err =' @37 err ;
   stop;
run;
proc print data=work.dept5100;
title1 'All the EMPLOYEES in the BRAINSTORMING
        Department';
run;
```

Output 2.5 Employee List

All the EMPLOYEES in the BRAINSTORMING Department

Obs	employee_ id	firstname	lastname	street	city	state	zip	phone	status	ssnumber	startdate	termdate	birthdate
1	466	ROY	ANDALE	44 TRIGGER RD	FRAMINGHAM	MA	03461	617554110	80	302760111	578061	500000	60030
2	457	HARRY	ARM	77 SUNSET STRIP	NATICK	MA	02178	617432092	30	502877014	777120	100000	34040
3	467	C.	BREEZE	200 NIGHTINGALE ST	FRAMINGHAM	MA	03461	617554238	70	111155669	279060	200000	34050
4	334	CAROLYN	CROW	891 SUMMER ST	WESTWOOD	MA	02090	617329177	60	102398011	79061	700000	44040
5	301	BURT	LANCHESTER	45 PINKERTON AVE	WALTHAM	MA	01476	617534110	90	112904050	675020	300000	32041
6	15	RENE	MAKER	10 DROVER DR	BOSTON	MA	02123	617452141	40	101067334	378010	200000	45052
7	341	RICHARD	MUNYON	17 BLACKHILL DR	WESTWOOD	MA	02090	617329001	70	111100208	180111	400000	50121
8	458	RICHARD	WAGNER	677 GERMANY LN	NATICK	MA	02178	617432110	90	101177666	378060	700000	34030

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Interface to CA-IDMS Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 104.

SAS/ACCESS® Interface to CA-IDMS Software: Reference, Version 8

ISBN 1–58025–547–7

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

1st printing, October 1999