SAS/ACCESS® 9.2
DATA Step Interface to CA-IDMS
Reference

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2008. *SAS/ACCESS® 9.2 for the DATA Step Interface to CA-IDMS: Reference*. Cary, NC: SAS Institute Inc.

SAS/ACCESS® 9.2 for the DATA Step Interface to CA-IDMS: Reference

Copyright © 2008, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59047-931-5

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

1st electronic printing, March 2008

1st printing, March 2008

Contents

CHAPTER 1 Overview of the SAS/ACCESS Interface to CA-IDMS

Introduction to SAS/ACCESS DATA Step Interface to CA-IDMS

SAS/ACCESS software provides a DATA step interface between SAS and Computer Associates' Integrated Data Management System (CA-IDMS). Through the DATA step, you can use INPUT statements and special extensions on the INFILE statement to access or extract data from the CA-IDMS database.

Note: The DATA step interface enables only read access to CA-IDMS data. You cannot update CA-IDMS data through the SAS/ACCESS interface to CA-IDMS. △

This document describes the DATA step interface and how to write CA-IDMS INFILE and INPUT statements.

Features of the DATA Step Interface

The following list describes the major features of the DATA step interface:

- The DATA step interface enables you to access CA-IDMS data by traversing the network using DML program functions calls. You cannot access data through Computer Associate's Logical Record Facility (LRF).
- The DATA step interface is a programming interface. You do not have to create descriptor files to retrieve the CA-IDMS data requested by your application.
- Coding DATA step programs requires knowledge of the database that is being accessed and the ability to write host-level calls to retrieve CA-IDMS data. In order to provide transparent access to CA-IDMS data, you can store compiled DATA step programs as SAS DATA step views.

Prerequisites for Using This Document

This document assumes that you understand the SAS DATA step and the statements that are used in the DATA step. It also assumes that you know how to enter standard SAS INFILE and INPUT statements. For complete information about DATA steps, INFILE statements, and INPUT statements, refer to *SAS Language Reference: Dictionary*.

There are many references to CA-IDMS processing in this document, such as CA-IDMS functions and status codes. If you are not familiar with the CA-IDMS information, refer to the appropriate Computer Associates documentation. You should also read Appendix 1, "IDMS Essentials," on page 81, which gives an overview of CA-IDMS concepts that are important in writing DATA step programs for CA-IDMS.

Example Data in the Interface to CA-IDMS Document

This document contains several examples that demonstrate how to use the DATA step interface to CA-IDMS. These examples use the CA-IDMS data contained in the EMPSCHM schema of the Employee database, which is the sample database Computer Associates ships with their CA-IDMS product. The examples in this document use data contained in the subschema EMPSS01, which is part of the EMPSCHM schema. Refer to your CA-IDMS documentation for more information about the Employee database.

Note: You cannot name a fileref for a task that is the same name as the subschema. △

The SAS/ACCESS software sample library file IDMSDS contains the SAS code used in the examples in this document.

CHAPTER

2

Using the SAS/ACCESS Interface to CA-IDMS

Overview of the DATA Step Statement Extensions

Introduction to the CA-IDMS INFILE and CA-IDMS INPUT Statements

Special SAS 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 SAS the parameters that are needed to build CA-IDMS calls. The CA-IDMS INFILE statement performs the following tasks:

- names the subschema
- names SAS variables to contain the following information:
  - 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 section.

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 for 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 SAS 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 enables access to all of the DEPARTMENT records. This example uses the IDMS option in the INFILE statement, which tells SAS 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 SAS 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 is 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 enables 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 is 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 iterations 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.

The following output shows the SAS log for this example.

Output 2.1 SAS Log for Introductory DATA Step Program

```
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.
```

The following output shows the results of this example.

Note: The log shows that 11 records were read from the infile, but the following results show only 9 observations. Every time SAS 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 Results of Introductory DATA Step Program

```
                     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: You cannot 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.

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 overrides might 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.

Using the CA-IDMS INFILE Statement

Definition of 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*>;

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 SAS 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. SAS 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 34 for more information about 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. Note that the LRECL option on a statement overrides the LRECL system option. See "CA-IDMS Input Buffer" on page 5 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 SAS 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 is not found are assigned missing values. Without the default action of the MISSOVER option, SAS would issue another function call any time 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

The following table summarizes the CA-IDMS INFILE statement options.

Table 2.1 Summary of CA-IDMS INFILE Statement Options

Option	Specifies
AREA=	the variable that contains the CA-IDMS area name.
DANAME=	the variable that contains database to be accessed by the run unit.
DANODE=	the variable that contains the central version of CA-IDMS where the database resides.
DBKEY=	the variable that contains a database record's key.
DCNAME=	the variable that contains the name of the CA-IDMS dictionary where the subschema is defined.
DCNODE=	the variable that contains the DC/UCF system needed to process the database requests.
ERRSTAT=	the variable to which the CA-IDMS error status is assigned after each CA-IDMS call.
FUNC=	the variable that contains the CA-IDMS call function used when a CA-IDMS INPUT statement is executed.
IKEY=	the variable that contains the value of the CALC KEY.
IKEYLEN=	the variable that contains the length of the CALC key.
KEYOFF=	the 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>	that SAS does not read 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.

Option	Specifies
RECORD=	the variable that contains the name of the CA-IDMS record you want to access.
SEQUENCE=	the variable that contains the requested record location within the set or area, establishes currency, and determines the direction of the traversal.
SET=	the variable that contains the name of the CA-IDMS set you want to access.
SORTFLD=	the variable that contains the value of the sort-control element to be used in searching the sorted set.
STOPOVER	that SAS stops processing if the record returned to the input buffer does not contain values for all variables specified in the CA-IDMS INPUT statement.

Guidelines for Using the CA-IDMS INFILE Statement and DML Function Calls

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 of the following sections 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 Function Call

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 specify 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 Function Call

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 is retrieved. The BIND RECORD function call does not retrieve any data from CA-IDMS. A BIND function call is not necessary if no data is being retrieved, that is, if 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 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 Function Calls

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 Function

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 Statement with No Arguments" on page 40.

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 this option:

- □ 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 Function

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 if needed, use one of the following options:

- □ 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 DBKEY Function

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 specify the following option:

- □ 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 is mapped to the named variables.

FIND/OBTAIN OWNER Function

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 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 SORT KEY Function

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, currencies 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 the following option:

□ 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 is mapped from the buffer to the named variables.

FIND/OBTAIN WITHIN SET or AREA Function

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 for 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 this option:

- □ FUNC= FIND or OBTAIN

And specify one of these options:

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

If needed, specify this option:

- □ 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 is mapped from the buffer to the named variables.

GET Function Call

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 the following option:

□ FUNC= GET

If needed, specify the following option:

□ 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 Function Call

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 Function Call

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 idmss 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, NEXT, 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

The following table 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 CA-IDMS Function Calls for COBOL DML

COBOL DML Call	INFILE Statement Options
ACCEPT db-key FROM CURRENCY	*FUNC*=ACCEPT *SEQUENCE*=CURRENT *DBKEY*=Required
ACCEPT db-key FROM record-name CURRENCY	*FUNC*=ACCEPT *SEQUENCE*=CURRENT *RECORD*=Required *DBKEY*=Required
ACCEPT db-key FROM set-name CURRENCY	*FUNC*=ACCEPT *SEQUENCE*=CURRENT *SET*=Required *DBKEY*=Required
ACCEPT db-key FROM area-name CURRENCY	*FUNC*=ACCEPT *SEQUENCE*=CURRENT *AREA*=Required *DBKEY*=Required

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

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

Using the CA-IDMS INPUT Statement

Definition of 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 is 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 "The Null INPUT Statement" on page 33 and "Holding Records in the Input Buffer" on page 33 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 is 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 33 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 is retrieved from the database. These functions are described in "Specifying DML Function Calls" on page 14. See "The Null INPUT Statement" on page 33 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 33 for more information.

Note: Every time SAS 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 can code several CA-IDMS INPUT statements that do not retrieve data, this count might 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.

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 33 for information.

Holding Records in the Input Buffer

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

SAS 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 "The Null INPUT Statement" on page 33.

Checking Call Status Codes

For each function call issued, CA-IDMS returns a call status code that indicates whether 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.

The following table 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 36 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 is 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 one of both of _ERROR_ and ERRSTAT= before determining what action to take. For more information, see “Holding Records in the Input Buffer” on page 33.

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 write your DATA step code 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 the 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 SAS 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 needs to be issued only once per record type before retrieval. The automatic SAS 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 is moved into the program data vector because no variables are defined on the **`INPUT @;`** statement. This function call enables 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 is moved from the SAS buffer into the program data vector.

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 is 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.

The following output shows the SAS log for this example.

Output 2.3 SAS Log for Traversing a Set

```
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.
```

The following output shows a portion of the results of this example.

Output 2.4 Traversing a 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 Statement 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 that is 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 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 Using the Trailing @ and Null INPUT Statement

```
                                 All the EMPLOYEES in the BRAINSTORMING Department
   employee_
 Obs   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
```

CHAPTER 3

Examples of SAS/ACCESS DATA Step Programs

Introduction to Examples of SAS/ACCESS DATA Step Programs

This section contains several example programs designed to introduce and illustrate the SAS/ACCESS DATA step interface to CA-IDMS.

All of the examples in this section can be executed using the sample EMPLOYEE database provided by Computer Associates. These examples illustrate syntax and call formats as well as logic tips for sequential and direct access of DBMS records and transaction-oriented applications. Each example is described using numbered comments that correspond to numbered lines of code. The output is shown for each example, but the log files are not included. For an example of a log file, see "Introductory Example of a DATA Step Program" on page 5. All of the examples have several statements in common, as described in the following section.

Statements Common to All SAS/ACCESS DATA Step Examples

All of the examples in this section contain or generate the following statements:

OPTIONS
The $IDMDBUG system option tells SAS to write information to the SAS log regarding call parameter values and the formatted calls submitted to CA-IDMS. You can use this information to debug your application and to inspect or verify the DML calls generated by the DATA step interface. Each of the examples in this section begin with an OPTIONS statement that specifies the $IDMDBUG option, but these OPTIONS statements are commented out with an asterisk. To execute the OPTIONS statement (and activate the $IDMDBUG system option), remove the asterisk.

INFILE
The INFILE statements used in these examples specify a subschema and the IDMS keyword, which indicates that the task will be accessing CA-IDMS records. The parameters on the INFILE statements create SAS variables whose values are used to format DML calls and check error status codes after those calls have been issued. None of the parameters have default values and, therefore, each variable must be assigned a valid value or blank before each call. None of the defined variables are included in the output data set. For specific information about each INFILE parameter, see "Using the CA-IDMS INFILE Statement" on page 10.

BIND RECORD
A BIND function call must be issued for each record whose data will be retrieved during execution of the DATA step. The BIND RECORD statement establishes addressability for a named record. In each of these examples, a null INPUT statement issues a BIND RECORD statement for each record (see "The Null INPUT Statement" on page 33). After the call is issued, the programs check the status code returned by CA-IDMS to be sure the call was successful. If the call is successful, the DATA step continues. If the call is unsuccessful, execution branches to the STATERR label, error information is written to the SAS log, and the DATA step terminates.

STATERR statements
For each call to CA-IDMS, the examples in this section check the status code that is returned by CA-IDMS. When CA-IDMS returns an unexpected status code, these examples execute the statements associated with the STATERR label. These statements

- □ issue an ERROR message to the SAS log describing the unexpected condition
- □ reset _ERROR_ to 0 to prevent the contents of the PDV (program data vector) from being written to the SAS log
- □ issue a STOP statement to immediately terminate the DATA step.

For more information about dealing with status codes, see "Checking Call Status Codes" on page 34.

Performing an Area Sweep

This example performs an area sweep of all DEPARTMENT records in the ORG-DEMO-REGION, and for each DEPARTMENT record, obtains all the EMPLOYEE records within the DEPT-EMPLOYEE set. An area sweep makes a sequential pass based on the physical location of a defined area for a specified record type. Records are accessed using the OBTAIN FIRST and OBTAIN NEXT DML calls. The example illustrates the concept of flattening out network record occurrences in an owner-member relationship. Owner (DEPARTMENT) information is repeated for each member (EMPLOYEE) in the set for observations written to the output SAS data set. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;
  data work.dept_employee;
❷ infile empss01 idms func=func
```

```
    record=recname area=iarea sequence=seq
    errstat=stat set=inset;

    /* BIND records to be accessed */

    if _n_ = 1 then do;
❸       func     = 'BIND';
        recname  = 'DEPARTMENT';
        input;
        if stat ne '0000' then go to staterr;

        recname  = 'EMPLOYEE';
        input;
        if stat ne '0000' then go to staterr;


        /* OBTAIN FIRST DEPARTMENT record */

❹       seq     = 'FIRST';
        func    = 'OBTAIN';
        recname = 'DEPARTMENT';
        iarea   = 'ORG-DEMO-REGION';
    end;


    /* FIND and OBTAIN NEXT DEPARTMENT record */

❺   if _n_ ge 2 then do;
        func  = 'FIND';
        seq   = 'OWNER';
        input;
        if stat ne '0000' then go to staterr;

        func     = 'OBTAIN';
        seq      = 'NEXT';
        recname  = 'DEPARTMENT';
        iarea    = 'ORG-DEMO-REGION';
        inset    = ' ';
    end;


❻   input @;
    if stat not in ('0000', '0307') then go
        to staterr;


    /* Stop DATA step when all DEPARTMENT records */
    /* have been accessed                         */

    if stat = '0307' then do;
       _error_ = 0;
       stop;
    end;
```

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


     /* OBTAIN EMPLOYEE records in set DEPT- */
     /* EMPLOYEE for CURRENT DEPARTMENT      */

❼    seq     = 'FIRST';
     recname = 'EMPLOYEE';
     inset   = 'DEPT-EMPLOYEE';
     iarea   = ' ';

     do until (stat ne '0000');
        input @;
        if stat not in ('0000', '0307') then go
            to staterr;
        if stat = '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        9.0
                 @96  startdate       yymmdd6.
                 @102 termdate        6.0
                 @108 birthdate       yymmdd6.;
                 output;
         seq = 'NEXT';
       end;
    end;
❽   _error_ = 0;
   return;


❾  staterr:
       put @1 'ERROR: ' @10 func @17 'RETURNED
                 STATUS =' @37 stat ;
       put @1 'ERROR: INFILE parameter values are: ';
       put @1 'ERROR: ' recname= iarea= seq=
                 inset=;
       put @1 'ERROR: DATA step execution
                 terminating.';
       _error_ = 0;
       stop;
   run;

   proc print data=work.dept_employee;
       format startdate birthdate date9.;
       title1 'This is an Area Sweep of the DEPT-
```

```
            EMPLOYEE Set';
run;
```

❶ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the OPTIONS statement.

❷ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the INFILE statement.

❸ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the BIND RECORD statement.

❹ For the first iteration of the DATA step, initialize the call parameters to obtain the FIRST DEPARTMENT record in the ORG-DEMO-REGION area.

❺ For subsequent iterations of the DATA step, initialize the call parameters to find the OWNER of the current EMPLOYEE record so that the program can obtain the NEXT DEPARTMENT record in the area. The null INPUT statement forces the call to be generated and submitted, but no data is returned to the input buffer (see "The Null INPUT Statement" on page 33). The status code returned by the FIND call is checked before proceeding to the next call.

❻ The **`INPUT @;`** statement holds the contents of the input buffer so the program can check the status code returned by CA-IDMS. (See "Holding Records in the Input Buffer" on page 33.) For a successful call, the next INPUT statement moves DEPARTMENT information from the input buffer to the named variables in the PDV.

When all records in the area have been accessed, CA-IDMS returns a 0307 status code (end-of-area). The program then issues a STOP statement to terminate the DATA step. Because there is no other end-of-file condition to normally terminate the DATA step, the STOP statement must be issued to avoid a looping condition. Because non-blank status codes set the automatic DATA step variable _ERROR_ to 1, _ERROR_ is reset to 0 to prevent the contents of the PDV from being written to the SAS log.

❼ After a DEPARTMENT record has been obtained, issue an OBTAIN for all EMPLOYEES that occur within the current DEPT-EMPLOYEE set. The DO UNTIL loop issues OBTAIN calls, verifies the status code, and moves employee information from the input buffer to the named variables in the PDV. For each successful OBTAIN, the **`INPUT @;`** statement holds onto the current input buffer contents until the status code is checked. After all EMPLOYEE records in the set have been accessed, CA-IDMS returns a status code of 0307, which terminates the DO UNTIL loop.

❽ At this point, the STAT variable must have a value of 0307. Because this code is non-zero, _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

❾ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.1 Performing an Area Sweep

```
                   This is an Area Sweep of the DEPT-EMPLOYEE Set

        department_                         department_  employee_
   Obs       id      department_name            head         id     firstname

     1      2000     ACCOUNTING AND PAYROLL       11          69     JUNE
     2      2000     ACCOUNTING AND PAYROLL       11         100     EDWARD
     3      2000     ACCOUNTING AND PAYROLL       11          11     RUPERT
     4      2000     ACCOUNTING AND PAYROLL       11          67     MARIANNE
     5      2000     ACCOUNTING AND PAYROLL       11         106     DORIS
     6      2000     ACCOUNTING AND PAYROLL       11         101     BRIAN
     7      3200     COMPUTER OPERATIONS           4           4     HERBERT
     8      3200     COMPUTER OPERATIONS           4          32     JANE


   Obs  lastname    street               city        state   zip     phone

     1  BLOOMER     14 ZITHER TERR       LEXINGTON    MA    01675  617555554
     2  HUTTON      781 CROSS ST         MELROSE      MA    02176  617665101
     3  JENSON      999 HARVEY ST        MELROSE      MA    02176  617665555
     4  KIMBALL     561 LEXINGTON AVE    LITTLETON    MA    01239  617492121
     5  KING        716 MORRIS ST        MELROSE      MA    02176  617665616
     6  NICEMAN     60 FLORENCE AVE      MELROSE      MA    02176  617665431
     7  CRANE       30 HERON AVE         KINGSTON     NJ    21341  201334143
     8  FERNDALE    60 FOREST AVE        NEWTON       MA    02576  617888811


   Obs  status     ssnumber    startdate    termdate    birthdate

     1    40      103955781      880050      500000       60042
     2    00      101122333      377090      700000       41030
     3    60      102234789      180092      900000       48081
     4    20      102277887      878091      900000       49042
     5    10      106784551      680081      600000       60091
     6    50      103345611       80050      600000       55121
     7    30      101677745      177051      400000       42032
     8    20      103456789      179090      900000       58011
```

Navigating Multiple Set Relationships

This example shows how to navigate multiple set relationships and use direct access methods involving database record keys. The output consists of observations containing related employee, office, and dental claim information. Observations are only output for employees that have dental claim record occurrences. To gather the information, the program performs an area sweep for the DEPARTMENT records and uses the FIND command to establish currency and navigate the DEPT-EMPLOYEE, OFFICE-EMPLOYEE, EMP-COVERAGE, and COVERAGE-CLAIMS sets. By accepting and storing database keys, currency can be reestablished on the EMPLOYEE record after obtaining OFFICE information and before gathering COVERAGE and DENTAL CLAIM information. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;
  data work.dental_records;
    drop tempkey;

❷   infile empss01 idms func=func record=recname
```

```
           dbkey=dkey errstat=stat sequence=seq
           set=inset area=subarea;


    /* BIND the records to be accessed */

❸   if _n_ = 1 then do;
        func       = 'BIND';
        recname    = 'EMPLOYEE';
        input;
        if stat ne '0000' then go to staterr;

        recname    = 'DEPARTMENT';
        input;
        if stat ne '0000' then go to staterr;

        recname    = 'COVERAGE';
        input;
        if stat ne '0000' then go to staterr;

        recname    = 'DENTAL-CLAIM';
        input;
        if stat ne '0000' then go to staterr;

        recname    = 'OFFICE';
        input;
        if stat ne '0000' then go to staterr;
    end;


    /* FIND FIRST/NEXT DEPARTMENT record in   */
    /* area ORG-DEMO-REGION                   */

❹   seq        = 'NEXT';
    if _n_ = 1 then seq = 'FIRST';
    func       = 'FIND';
    recname    = 'DEPARTMENT';
    subarea    = 'ORG-DEMO-REGION';
    inset      = ' ';
    input;
    if stat not in ('0000', '0307') then go to
        staterr;


    /* STOP DATA step execution if no more */
    /* DEPARTMENT records                  */

❺   if stat = '0307' then do;
        _error_ = 0;
        stop;
    end;


❻   do until (stat ne '0000');
```

```
/* OBTAIN NEXT EMPLOYEE record */

 func       = 'OBTAIN';
 seq        = 'NEXT';
 recname    = 'EMPLOYEE';
 inset      = 'DEPT-EMPLOYEE';
 input @;
 if stat not in ('0000','0307') then go to
     staterr;
 if stat = '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.
         @76  phone          10.0
         @86  status         $char2.
         @88  ssnumber       $char9.
         @109 birthdate      yymmdd6.;


/* ACCEPT DBKEY for current EMPLOYEE and */
/* store in tempkey                      */

❼  func       = 'ACCEPT';
   seq        = 'CURRENT';
   dkey       = '    ';
   inset      = '             ';
   input;
   if stat ne '0000' then go to staterr;
   tempkey=dkey;


 /* OBTAIN OFFICE record for current  */
 /* EMPLOYEE                          */

❽  func       = 'OBTAIN';
   seq        = 'OWNER';
   dkey       = '    ';
   inset      = 'OFFICE-EMPLOYEE';
   input @;
   if stat ne '0000' then go to staterr;
   input @1   office_code    $char3.
         @4   office_street  $char20.
         @24  office_city    $char15.
         @39  office_state   $char2.
         @41  office_zip     $char9.;

 /* FIND EMPLOYEE using DBKEY stored in */
```

```
      /* tempkey                               */

❾       func       = 'FIND';
        recname    = '            ';
        dkey       = tempkey;
        seq        = '       ';
        inset      = '                ';
        input;
        if stat ne '0000' then go to staterr;


      /* FIND FIRST COVERAGE record for  */
      /* current EMPLOYEE                */

❿       func       = 'FIND';
        recname    = 'COVERAGE';
        dkey       = '    ';
        seq        = 'FIRST';
        inset      = 'EMP-COVERAGE';
        input;
        if stat ne '0000' then go to staterr;


      /* OBTAIN LAST DENTAL-CLAIM record      */
      /* within COVERAGE-CLAIMS               */
      /* Observations are only OUTPUT for     */
      /* employees with dental claim records  */

⓫       func       = 'OBTAIN';
        recname    = 'DENTAL-CLAIM';
        seq        = 'LAST';
        inset      = 'COVERAGE-CLAIMS';
        input @;
        if stat not in ('0000','0307') then go to
            staterr;
        do while (stat eq '0000');
           input @1   claim_year      $2.
                 @3   claim_month     $2.
                 @5   claim_day       $2.
                 @7   claim_firstname $10.
                 @17  claim_lastname  $15.
                 @32  birthyear       $2.
                 @34  birthmonth      $2.
                 @36  birthday        $2.
                 @38  sex             $1.
                 @39  relation        $10.
                 @49  dds_firstname   $10.
                 @59  dds_lastname    $15.
                 @74  ddsstreet       $20.
                 @94  ddscity         $15.
                 @109 ddsstate        $2.
                 @111 ddszip          $9.
                 @120 license         $6.
                 @126 num_procedure   ib2.
```

```
                     @131 tooth_number    $2.
                     @133 service_year    $2.
                     @135 service_month   $2.
                     @137 service_day     $2.
                     @139 procedure_code  $4.
                     @143 descservice     $60.
                     @203 fee             pd5.2;
               output;

             /* OBTAIN PRIOR DENTAL-CLAIM record */

               seq        = 'PRIOR';
               input @;
            end;


            /* When DENTAL-CLAIM records have been */
            /* processed, release INPUT buffer and */
            /* reset STAT to OBTAIN NEXT EMPLOYEE  */

12          if stat = '0307' then do;
               stat = '0000';
               input;
            end;
            else go to staterr;
          end;
        end;

        /* When all EMPLOYEEs have been processed, */
        /* reset ERROR flag and continue with next */
        /* DEPARTMENT                              */


13    _error_ = 0;
    return;


14  STATERR:
      put @1 'ERROR: ' @10 func @17 'RETURNED
                STATUS =' @37 stat;
      put @1 'ERROR: INFILE parameter values are: ';
      put @1 'ERROR: ' recname= seq= inset= dkey=
                subarea=;
      put @1 'ERROR: DATA step execution
                terminating.';
      _error_ = 0;
      stop;
    run;

    proc print data=work.dental_records;
      format birthdate date9.;
      title1 'Dental Claim Information';
    run;
```

❶ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the OPTIONS statement.

❷ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the INFILE statement.

❸ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the BIND RECORD statement.

❹ The first time the DATA step executes, the FIND command locates the FIRST DEPARTMENT record in the area. For subsequent DATA step iterations, initialize the call parameters to find the NEXT DEPARTMENT record in the area. The null INPUT statement generates and submits the call, but no data is returned to the input buffer. A SAS IF statement checks the status code returned by the FIND call.

❺ As DEPARTMENT records are located, the program checks the status code returned by CA-IDMS. When all records in the area have been accessed, CA-IDMS returns a 0307 status code (end-of-area). The program then issues a STOP statement to terminate the DATA step. Since there is no other end-of-file condition to normally terminate the DATA step, the STOP statement must be issued to avoid a looping condition. Also, non-blank status codes set the automatic DATA step variable _ERROR_ to 1, so _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

❻ For the current DEPARTMENT, the program must access all EMPLOYEE records in the DEPT-EMPLOYEE set. The DO UNTIL loop executes until the status code that is returned from CA-IDMS is not equal to 0000. For unexpected status codes, the statements associated with the STATERR label are executed, and the loop terminates when the end-of-set status code (0307) is encountered. An OBTAIN is used to retrieve the EMPLOYEE records. After the status code is verified to be successful, data is moved from the input buffer to the PDV by executing the INPUT statement. The first **`INPUT @;`** statement forces the call to be submitted and enables a returned status code to be checked before any attempt to move data from the input buffer to the PDV. This process eliminates any possibility of moving invalid data into the PDV and avoids unnecessary data conversions when the call fails.

❼ After an EMPLOYEE record has been obtained, the ACCEPT command takes the record’s database key and stores it in DKEY, the variable defined by the DBKEY= INFILE parameter. The value is then stored in a variable called TEMPKEY because the DKEY variable must be set to blanks to generate the next call correctly. By saving the record’s database key, the program can reestablish currency on the EMPLOYEE record after obtaining OWNER information from the OFFICE record in the OFFICE-EMPLOYEE set.

❽ OFFICE records are retrieved by issuing an OBTAIN OWNER within the OFFICE-EMPLOYEE set. The **`INPUT @;`** statement generates and submits the call. For a successful OBTAIN, OFFICE information is moved from the held input buffer to the PDV.

❾ The program is now ready to establish currency back to the EMPLOYEE record current in the DEPT-EMPLOYEE set. The database key value stored in TEMPKEY is used to format a FIND DBKEY command. The null INPUT statement submits the call and the status code is checked to be sure it was successful. Any status code other than 0000 routes execution to the STATERR label.

❿ Now current on EMPLOYEE, a FIND is issued to locate the FIRST COVERAGE record in the EMP-COVERAGE set. For any status code not equal to 0000, execution is routed to the STATERR label.

⓫ The goal is to process all the DENTAL-CLAIM records in the COVERAGE-CLAIMS set for the current COVERAGE record. An OBTAIN LAST is submitted by the **`INPUT @;`** statement, and if DENTAL-CLAIM records exist in the set, then the subsequent INPUT statement maps the returned data from the input buffer to the PDV. At this point, a complete observation–one containing EMPLOYEE, OFFICE and DENTAL-CLAIM data–is output to the SAS data set. The sequence variable SEQ is assigned a value of PRIOR so that subsequent iterations of the DO WHILE loop submit an OBTAIN PRIOR call. The DO WHILE continues executing until the OBTAIN PRIOR returns a status code not equal to 0000.

⓬ If the status code indicates end-of-set (0307) then the status variable is reset to 0000. The assignment is done to enable the DO UNTIL loop (see ❻) to continue executing and issuing OBTAIN calls for employees in the current department. The null INPUT statement is issued to release the buffer held by the **`INPUT @;`** statement within the DO WHILE loop. In this example, because there was a held buffer, the null INPUT statement does not attempt to generate and submit a DML call. The buffer must be released so the next DML call, the OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE, can be generated. For any other status code, execution branches to the STATERR label.

⓭ At this point, the STAT variable must have a value of 0307. Since this code is non-zero, _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

⓮ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.2 Navigating Multiple Set Relationships

```
                              Dental Claim Information

          employee_
    Obs      id      firstname   lastname     street       city     state   zip

     1        4      HERBERT     CRANE    30 HERON AVE  KINGSTON    NJ    21341
     2       30      HENRIETTA   HENDON   16 HENDON DR  WELLESLEY   MA    02198

                                                    office_
    Obs    phone     status  ssnumber   birthdate   code      office_street

     1  2013341433    01    016777451    420321     001    20 W BLOOMFIELD ST
     2  6178881212    01    011334444    331006     002    567 BOYLSTON ST

                   office_ office_ claim_ claim_ claim_  claim_    claim_
    Obs office_city  state    zip    year  month   day   firstname lastname

     1  SPRINGFIELD   MA     02076    80     10     04    JESSICA   CRANE
     2  BOSTON        MA     02243    77     05     23    HELOISE   HENDON

                                                          dds_     dds_
    Obs birthyear  birthmonth  birthday  sex  relation  firstname  lastname

     1     57          01         11      F   WIFE         DR      PEPPER
     2     68          03         15      F   DAUGHTER     SAL     SARDONICUS

                                                              num_   tooth_
    Obs ddsstreet         ddscity  ddsstate ddszip license procedure number

     1  78 COLA RD       PRINCETON    NJ    01762  877073      2       08
     2  402 NATURE'S WAY NEEDHAM      MA    02243  459631      1       14

        service_    service_    service_    procedure_
    Obs   year       month        day          code       descservice    fee

     1     80          09          16          0076         FILLING       14
     2     77          05          02          0076         FILLING       14
```

Using a SAS Data Set as a Transaction File

This example illustrates how to use an input SAS data set as a transaction file to supply parameter values for direct access DML calls. These calls obtain CA-IDMS records using CALC key values. The transaction data set WORK.EMP supplies CALC key values for EMPLOYEE records. The program then accesses EMPOSITION records in the EMP-EMPOSITION set to create an output SAS data set that contains all of the position information for the employees named in WORK.EMP. The DATA step terminates after all observations from WORK.EMP have been read. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;

❷ data work.emp;
   input id $4.;
```

```
datalines;
0471
0301
0004
0091
1002
;
data work.emp_empos;
  drop id chkrec nxtrec;
  length chkrec $ 29;

❸ infile empss01 idms func=func record=recname
        ikeylen=keyl errstat=stat sequence=seq
        set=inset ikey=ckey dbkey=dkey;

  /* BIND the records to be accessed */

❹ if _n_ = 1 then do;
     func       = 'BIND';
     recname    = 'EMPLOYEE';
     input;
     if stat ne '0000' then go to staterr;

     recname    = 'EMPOSITION';
     input;
     if stat ne '0000' then go to staterr;
  end;


  /* OBTAIN EMPLOYEE records using CALC key  */
  /* from EMP data set */

❺ set work.emp;
  func       = 'OBTAIN';
  ckey       = id;
  keyl       = 4;
  recname    = 'EMPLOYEE';
  input @;
  if stat not in ('0000', '0326') then go to
     staterr;
  if stat = '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.
          @76  phone           10.0
          @86  status          $char2.
          @88  ssnumber        $char9.
          @97  emp_start       yymmdd6.
          @103 emp_term        6.0
          @109 birthdate       yymmdd6.;
     /* OBTAIN LAST EMPOSITION record in */
     /* EMP-EMPOSITION set               */

❻    func       = 'OBTAIN';
     seq        = 'LAST';
     ckey       = '    ';
     keyl       = 0;
     dkey      = ' ';
     recname    = 'EMPOSITION';
     inset      = 'EMP-EMPOSITION';
     input @;
     if stat not in ('0000', '0326') then go to
         staterr;
     if stat = '0000' then do;
       chkrec = put(employee_id,z4.) ||firstname ||
          lastname;


    /* Process all EMPOSITION records for */
    /* current EMPLOYEE                   */

❼     do until (nxtrec = chkrec);
         input @1   pos_start    yymmdd6.
               @7   pos_finish   6.0
               @13  salarygrade  2.0
               @15  salary       pd5.2
               @20  bonus        pd2.0
               @22  commission   pd2.0
               @24  overtime     pd2.0;
         output;


     /* ACCEPT CURRENCY for PRIOR record in  */
     /* EMP-EMPOSITION set                   */

❽      func       = 'ACCEPT';
       dkey       = '    ';
       seq        = 'PRIOR  ';
       recname    = '          ';
       inset      = 'EMP-EMPOSITION';
       input;
       if stat eq '0000' then do;


  /* OBTAIN current record using the DBKEY */

❾        func       = 'OBTAIN';
         seq        = '       ';
         inset      = '        ';
         input @1 nxtrec $29. @;
```

```
if stat ne '0000' then go to staterr;
           end;
         end;
       end;
⓾     else do;
           put 'WARNING: No EMPOSITION record for
                  EMPID= ' id;
           put 'WARNING: Execution continues with
                  next EMPID.';
           _error_ = 0;
       end;
     end;
     else do;
          put 'WARNING: No EMPLOYEE record for EMPID= '
                  id;
          put 'WARNING: Execution continues with next
                  EMPID.';
          _error_ = 0;
     end;
   return;


⓫ staterr:
     put @1 'ERROR: ' @10 func @17 'RETURNED
              STATUS =' @37 stat;
     put @1 'ERROR: INFILE parameter values are: ';
     put @1 'ERROR: ' recname= ckey= seq= inset=
              keyl= dkey=;
     put @1 'ERROR: DATA step execution
              terminating.';
     _error_ = 0;
     stop;
   run;

   proc print data=work.emp_empos;
      format emp_start birthdate pos_start
          date9. salary dollar12.2
      title1 'Positions Held by Specified
          Employees';
      title2 'Listed in Ascending Order by
          Initdate/Termdate';
   run;
```

❶ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the OPTIONS statement.

❷ This DATA step execution creates the transaction data set WORK.EMP. The 4-byte character variable ID contains CALC key values that will be used to access EMPLOYEE records directly by employee ID.

❸ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the INFILE statement.

❹ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the BIND RECORD statement.

❺ An observation is read from WORK.EMP, and the current ID value is used as a CALC key for obtaining the EMPLOYEE. The length of the CALC key is specified with the IKEYLEN= variable KEYL. The **`INPUT @;`** statement submits the call and places a hold on the input buffer so that the status code can be checked. For any unexpected status code, execution branches to the STATERR label. A status code of 0000 directs execution to the INPUT statement which maps data from the held input buffer to the PDV and then releases the buffer.

❻ The program now attempts to obtain EMPOSITION records in the order of oldest (LAST) to most current (FIRST). First, an OBTAIN LAST call is issued for the EMPOSITION record in set EMP-EMPOSITION. The **`INPUT @;`** statement submits the call and holds the buffer so the status code can be checked. Execution branches to the STATERR label for any unexpected status code. For status code 0000, a variable called CHKREC is assigned a value that is composed of the current employee's CALC key, first name, and last name. CHKREC is used in the condition of the DO UNTIL loop described in the next step.

❼ The DO UNTIL loop navigates the EMP-EMPOSITION set occurrences in reverse order. The condition on a DO UNTIL loop is evaluated at the bottom of the loop after the statements in the loop have been executed (see ❾).

The input buffer already contains an EMPOSITION record. The INPUT statement maps EMPOSITION data from the held buffer into the variables in the PDV. At this point, a complete observation exists and is output to the WORK.EMP_EMPOS data set. No observation is written when no EMPOSITION records exist for a specified employee.

❽ To move in reverse order, the ACCEPT PRIOR call is generated and issued within the EMP-EMPOSITION set to return the database key of the prior record in the current set occurrence. The database key is stored in the variable defined by the DBKEY= parameter on the INFILE statement, DKEY. The null INPUT statement submits the call. For any status code not equal to 0000, execution branches to the STATERR label.

❾ For a successful ACCEPT call, an OBTAIN is issued using the database key stored in DKEY. Using this method to navigate the set implies that no end-of-set status code is set. To determine whether an end-of-set condition exists, the INPUT statement submits the OBTAIN, moves the first 29 bytes of data into a character variable called NXTREC and places a hold on the buffer contents. For a successful OBTAIN, execution resumes with the evaluation of the DO UNTIL condition. If CHKREC equals NXTREC, then the program is current on the EMPLOYEE (owner of the set) so the loop terminates. If the variables are not equal, then the record in the buffer is an EMPOSITION record, so data is moved into the PDV from the input buffer, and another observation is output for the current employee.

❿ This group of statements enables execution to continue when either no EMPOSITION records exist for the specified employee or no EMPLOYEE record exists for the CALC value specified in the transaction data set. In both cases, informative WARNING messages are written to the SAS log, and _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

⓫ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.3 Using a SAS Data Set as a Transaction File

```
                    Positions Held by Specified Employees
                 Listed in Ascending Order by Initdate/Termdate

     employee_
Obs     id     firstname  lastname         street        city       state

 1     471     THEMIS     PAPAZEUS    234 TRANSWORLD ST  NORTHBORO   MA
 2     471     THEMIS     PAPAZEUS    234 TRANSWORLD ST  NORTHBORO   MA
 3     301     BURT       LANCHESTER  45 PINKERTON AVE   WALTHAM     MA
 4     301     BURT       LANCHESTER  45 PINKERTON AVE   WALTHAM     MA
 5     301     BURT       LANCHESTER  45 PINKERTON AVE   WALTHAM     MA
 6       4     HERBERT    CRANE       30 HERON AVE       KINGSTON    NJ
 7       4     HERBERT    CRANE       30 HERON AVE       KINGSTON    NJ
 8       4     HERBERT    CRANE       30 HERON AVE       KINGSTON    NJ
 9      91     MADELINE   ORGRATZI    67 RAINBOW DR      KENDON      MA

Obs  zip     phone    status ssnumber  emp_start emp_term birthdate pos_start

 1  03256 6174561277   01   022887770 07SEP1978     0    04MAR1935 07SEP1978
 2  03256 6174561277   01   022887770 07SEP1978     0    04MAR1935 01JAN1982
 3  01476 6175341109   01   129040506 03FEB1975     0    19APR1932 03FEB1975
 4  01476 6175341109   01   129040506 03FEB1975     0    19APR1932 03FEB1977
 5  01476 6175341109   01   129040506 03FEB1975     0    19APR1932 03FEB1980
 6  21341 2013341433   01   016777451 14MAY1977     0    21MAR1942 14MAY1977
 7  21341 2013341433   01   016777451 14MAY1977     0    21MAR1942 15NOV1979
 8  21341 2013341433   01   016777451 14MAY1977     0    21MAR1942 14MAY1982
 9  06182 6174311919   01   231067878 10OCT1980     0    16OCT1951 10OCT1980

      pos_
Obs  finish   salarygrade         salary   bonus   commission   overtime

 1  811231         72         $90,000.00     10         0           0
 2       0         82        $100,000.00     10         0           0
 3  770202         52         $39,000.00      7         0           0
 4  800202         52         $45,000.00      7         0           0
 5       0         53         $54,500.00      7         0           0
 6  791114         71         $60,000.00     10         0           0
 7  820513         71         $70,000.00     10         0           0
 8       0         71         $75,000.00     10         0           0
 9       0         43         $39,000.00      7         0           0
```

Using Information in a SAS Data Set to Locate Records

This example, like the previous example, uses the information stored in a SAS data set to locate records in the CA-IDMS database. In this case, not only do the observations in the transaction data set WORK.OFFICE provide CALC information for the OFFICE record, they supply sort key information as well for the EMPLOYEE record. Therefore, the program uses both pieces of information to locate a specific occurrence of the OFFICE record, followed by a specific occurrence of the EMPLOYEE record in the OFFICE-EMPLOYEE set occurrence. If any of the transaction information is incorrect, a WARNING message is issued and no observation is output to WORK.EMP. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;

❷ data work.office;
   input offkey $3. emp $25.;
   datalines;
   001GARFIELD       JENNIFER
   002BLOOMER        JUNE
```

```
005JOE            SMITH
008WAGNER         RICHARD
010ANDALE         ROY
;
data work.emp;
   drop offkey emp;

❸ infile empss01 idms func=func record=recname
         ikey=ckey ikeylen=keyl errstat=stat
         sequence=seq set=inset sortfld=skey;


   /* BIND the records to be accessed */

❹ if _n_ = 1 then do;
     func       = 'BIND';
     recname    = 'EMPLOYEE';
     input;
     if stat ne '0000' then go to staterr;

     recname    = 'OFFICE';
     input;
     if stat ne '0000' then go to staterr;
  end;


/* OBTAIN OFFICE record based on CALC key */

❺ set work.office;
  func       = 'OBTAIN';
  ckey       = offkey;
  keyl       = 3;
  recname    = 'OFFICE';
  inset      = ' ';
  skey       = ' ';
  input @;
  if stat not in ('0000', '0326') then go to
      staterr;
  if stat = '0000' then do;
     input @1   office_code       $char3.
           @4   office_street     $char20.
           @24  office_city       $char15.
           @39  office_state      $char2.
           @41  office_zip        $char9.
           @50  officephone1     9.0
           @59  officephone2     9.0
           @68  officephone3     9.0
           @77  areacode         $char3.
           @80  speeddial        $char3.;


   /* FIND EMPLOYEE record within set  */
   /* using SORT key                   */
```

```
❻       func       = 'FIND';
        skey       = emp;
        ckey       = '   ';
        keyl       = 25;
        recname    = 'EMPLOYEE';
        inset      = 'OFFICE-EMPLOYEE ';
        input;
        if stat not in ('0000', '0326') then
            go to staterr;
        if stat = '0000' then do;


           /* OBTAIN CURRENT record */

❼          func       = 'OBTAIN';
           seq        = 'CURRENT';
           skey       = '                         ';
           keyl       = 0;
           inset      = '                ';
           input @;
           if stat ne '0000' then go to staterr;
           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      yymmdd6.
                 @103 termdate       6.0
                 @109 birthdate      yymmdd6.;
           output;
        end;

❽       else do;
           put 'WARNING: No EMPLOYEE record for
                  SORT key= ' emp '.';
           put 'WARNING: Execution continues with
                  next OFFICE CALC.';
           put;
           _error_ = 0;
        end;
     end;
     else do;
        put 'WARNING: No OFFICE record for CALC
               key= 'offkey '.';
        put 'WARNING: Execution continues with
               next OFFICE CALC.';
```

```
put;
        _error_ = 0;
      end;
   return;

❾ STATERR:
      put @1 'ERROR: ' @10 func @17 'RETURNED
               STATUS =' @37 stat;
      put @1 'ERROR: INFILE parameter values are: ';
      put @1 'ERROR: ' recname= ckey= keyl= seq=
                inset= skey=;
      put @1 'ERROR: DATA step execution
                terminating.';
      _error_ = 0;
      stop;
   run;

   proc print data=work.emp;
     format startdate birthdate date9.;
     title1 'Office and Employee Information';
     title2 'as Specified in Transaction Data Set';
   run;
```

❶ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the OPTIONS statement.

❷ This DATA step execution creates the transaction data set WORK.OFFICE. The 3-byte character variable OFFKEY contains CALC key values that will be used to access OFFICE records directly by office code. The 25-byte character variable EMP contains SORT key values that will be used to access EMPLOYEE records directly using the EMP-NAME-NDX.

❸ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the INFILE statement.

❹ See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the BIND RECORD statement.

❺ An observation is read from WORK.OFFICE, and the current OFFKEY value is used as a CALC value to obtain the OFFICE record. The length of the CALC key is specified by the IKEYLEN= variable KEYL. The **`INPUT @;`** statement submits the call and places a hold on the input buffer so that the status code can be checked. Any unexpected status code branches execution to the STATERR label. A status code of 0000 directs execution to the INPUT statement, which maps data from the held input buffer to the PDV, then releases the buffer.

❻ The program must now locate a specific occurrence of EMPLOYEE within the current OFFICE-EMPLOYEE set. A FIND EMPLOYEE WITHIN OFFICE-EMPLOYEE call is generated using the sort key information in the EMP variable read from WORK.OFFICE. The sort key length is set to 25. (The previous length of 3 applied to the OFFICE CALC key.) The null INPUT statement submits the call but does not place a hold on the buffer. FIND does not return any data. For any unexpected status code, execution branches to the STATERR label. If the FIND is successful, execution continues with the next DML call.

❼ Having successfully located the EMPLOYEE using the supplied index value, an OBTAIN CURRENT call is generated so that EMPLOYEE record information can be accessed by the program. SKEY is set to blank and KEYL is set to 0 so that their values are not used for the OBTAIN call. The **`INPUT @;`** statement submits the generated call and places a hold on the input buffer so that the status code can be checked. Any status code not equal to 0000 routes execution to the STATERR label. For a successful OBTAIN, the INPUT statement maps EMPLOYEE record data from the input buffer to the specified variables in the PDV and releases the input buffer. At this point, the OUTPUT statement writes an observation to the output data set. Only observations that contain both office and employee information are output.

❽ This group of statements enables execution to continue when either no EMPLOYEE record exists for the specified sort key value or no OFFICE record exists for the specified CALC value from WORK.OFFICE. In both cases, informative WARNING messages are written to the SAS log and _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

❾ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.4 Using a Data Set to Locate Records

```
                      Office and Employee Information
                    as Specified in Transaction Data Set

       office_                                  office_ office_
   Obs  code      office_street     office_city  state    zip   officephone1

    1    001   20 W BLOOMFIELD ST    SPRINGFIELD   MA     02076    369772100
    2    002   567 BOYLSTON ST      BOSTON        MA     02243    956237795
    3    008   910 E NORTHSOUTH AVE WESTON        MA     02371    367919136

                                                        employee_
   Obs officephone2  officephone3  areacode  speeddial      id     firstname

    1           0             0                              3     JENNIFER
    2    625719562     398000000                            69     JUNE
    3    792923671     327000000                           458     RICHARD


   Obs lastname      street      city       state   zip      phone    status

    1  GARFIELD  110A FIRTH ST   STONEHAM    MA    02928  6173321967    01
    2  BLOOMER   14 ZITHER TERR  LEXINGTON   MA    01675  6175555544    01
    3  WAGNER    677 GERMANY LN  NATICK      MA    02178  6174321109    01


   Obs ssnumber     startdate    termdate    birthdate

    1  021994516    21JAN1977        0       18AUG1945
    2  039557818    05MAY1980        0       25APR1960
    3  011776663    07JUN1978        0       04MAR1934
```

Supplying Transaction Information and Navigating Set Occurrences

This example introduces alternate techniques for supplying transaction information and for navigating set occurrences. It also uses program logic to subset records that are accessed to produce output which meets specified criteria. A macro variable supplies the transaction information that produces the subset of employee data. An OBTAIN Nth EMPLOYEE WITHIN DEPT-EMPLOYEE call is used to navigate the current set occurrence.

Using macro variables is one tool for providing transaction information. SAS data set variables have been used in previous examples; another method might make use of an SCL variable. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;

❷ %let hireyear = 1977;

  data work.emp;
     format initdate date9.;
     drop i;

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

    /* BIND records to be accessed */

❹   if _n_ = 1 then do;
       func       = 'BIND';
       recname    = 'EMPLOYEE';
       input;
       if stat ne '0000' then go to staterr;

       recname    = 'DEPARTMENT';
       input;
       if stat ne '0000' then go to staterr;
    end;


  /* FIND FIRST/NEXT DEPARTMENT record in AREA */

❺   seq        = 'NEXT';
    if _n_ = 1 then seq = 'FIRST';
    func       = 'FIND';
    recname    = 'DEPARTMENT';
    subarea    = 'ORG-DEMO-REGION';
    inset      = ' ';
    input;
    if stat not in ('0000', '0307') then go
        to staterr;


    /* STOP DATA step execution if no more   */
    /* DEPARTMENT records                    */

❻   if stat = '0307' then do;
       _error_ = 0;
       stop;
    end;


    /* OBTAIN nth EMPLOYEE within
       DEPT-EMPLOYEE */
```

```
❼   i=0;
    do until (stat ne '0000');
       i + 1;
       func       = 'OBTAIN';
       seq        = trim(left(put(i,8.)));
       recname    = 'EMPLOYEE';
       inset      = 'DEPT-EMPLOYEE';
       subarea    = '                 ';
       input @;
       if stat not in ('0000', '0307') then
            go to staterr;
       if stat = '0000' then do;
          input @1   employee_id    4.0
                @5   firstname      $char10.
                @15  lastname       $char15.
                @97  initdate       yymmdd6.;


          /* For employees hired in 1977 FIND */
          /* CURRENT DEPARTMENT               */

❽         if year(initdate) = &hireyear then do;
             func       = 'FIND';
             seq        = 'CURRENT';
             recname    = 'DEPARTMENT';
             inset      = '                 ';
             input;
             if stat ne '0000' then go to staterr;


           /* OBTAIN CURRENT DEPARTMENT info */
           /* and OUTPUT                     */

❾            func       = 'OBTAIN';
             seq        = 'CURRENT';
             recname    = '                 ';
             input @;
             if stat ne '0000' then go to staterr;
             input @1   department_id   4.0
                   @5   department_name $char45.;
             output;
          end;
       end;
    end;
❿   _error_ = 0;
  return;

⓫ staterr:
     put @1 'ERROR: ' @10 func @17 'RETURNED
               STATUS =' @37 stat;
     put @1 'ERROR: INFILE parameter values are: ';
     put @1 'ERROR: ' recname= subarea= seq=
               inset=;
```

```
      put @1 'ERROR: DATA step execution
                terminating.';
      _error_ = 0;
      stop;
   run;

   proc print data=work.emp;
      title "Departments that Hired Employees in
             &hireyear";
   run;
```

❶ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the OPTIONS statement.

❷ The %LET statement assigns the value 1977 to a newly defined macro variable called HIREYEAR. This macro variable is used to supply subset criteria as part of the condition on the IF statement in step ❼.

❸ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the INFILE statement.

❹ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the BIND RECORD statement.

❺ On the first DATA step iteration, the FIND command locates the FIRST DEPARTMENT record in the area. For subsequent DATA step iterations, initialize the call parameters to find the NEXT DEPARTMENT record in the area. The null INPUT statement generates and submits the call, but no data is returned to the input buffer. The IF statement checks the status code returned by the FIND call.

❻ As DEPARTMENT records are located, the program checks the status code returned by CA-IDMS. When all records in the area have been accessed, CA-IDMS returns a 0307 status code (end-of-area). The program then issues a STOP statement to terminate the DATA step. Since there is no other end-of-file condition to normally terminate the DATA step, the STOP statement must be issued to avoid a looping condition. Also, non-blank status codes set the automatic DATA step variable _ERROR_ to 1. _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

❼ At this point, the program has currency on a DEPARTMENT record and needs to navigate the current occurrence of the DEPT-EMPLOYEE set. The DO UNTIL loop generates an OBTAIN Nth EMPLOYEE call for each EMPLOYEE record in the set. Valid N values are generated using the loop counter variable **i** and the PUT, LEFT, and TRIM functions. The N values are stored in the variable SEQ.

The **INPUT @;** statement submits the call and places a hold on the input buffer while the status code is checked. For any unexpected status codes, execution branches to the STATERR label. For a successful OBTAIN Nth call, the INPUT statement maps employee information from the input buffer to the specified variables in the PDV and releases the input buffer.

The DO UNTIL loop terminates when CA-IDMS returns an end-of-set status code (0307).

❽ The program now evaluates the condition in the IF statement and enters the DO-END block of code only if the employee INITDATE indicates a hire year of 1977. The %LET statement assigned the value 1977 to macro variable &HIREYEAR before the DATA step executed (see ❷). This variable was resolved when the DATA step was compiled. If the year portion of the employee INITDATE is 1977, then a FIND CURRENT DEPARTMENT is generated to obtain the owner of the current EMPLOYEE record. The null INPUT statement submits the call but does not place a hold on the input buffer because FIND does not return any data. If the FIND returns any status code other than 0000, execution branches to label STATERR.

9 After the owner DEPARTMENT record is located, an OBTAIN CURRENT is generated to request that the DEPARTMENT record be placed into the input buffer. The **`INPUT @;`** statement submits the call and places a hold on the input buffer while the status is checked. For any status code other than 0000, execution branches to the STATERR label. For a successful OBTAIN call, the INPUT statement maps department information from the input buffer to the specified variables in the PDV and releases the input buffer. The OUTPUT statement writes the current observation to data set WORK.EMP. To avoid unnecessary input/output for departments that contain no employees with a hire year of 1977, the program postpones the OBTAIN of DEPARTMENT until the EMPLOYEE qualification criteria have been met. If you anticipate that many employees across multiple departments were hired in &HIREYEAR, then you could either OBTAIN DEPARTMENT before navigating the DEPT-EMPLOYEE set or add additional logic to OBTAIN CURRENT only once for the current set occurrence.

10 At this point, the STAT variable must have a value of 0307. Since this code is non-zero, _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

11 See “Statements Common to All SAS/ACCESS DATA Step Examples” on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.5 Supplying Transaction Information

```
                  Departments that Hired Employees in 1977

                                                        d
                                                        e
                                                   d    p
                                                   e    a
                      e                            p    r
                      m                            a    t
                      p    f                       r    m
             i        l    i          l            t    e
             n        o    r          a            m    n
             i        y    s          s            e    t
             t        e    t          t            n    _
             d        e    n          n            t    n
   O         a        _    a          a            _    a
   b         t        i    m          m            i    m
   s         e        d    e          e            d    e

   1   07SEP1977   100   EDWARD     HUTTON     2000   ACCOUNTING AND PAYROLL
   2   14MAY1977     4   HERBERT    CRANE      3200   COMPUTER OPERATIONS
   3   04MAR1977   371   BETH       CLOUD      5300   BLUE SKIES
   4   01DEC1977   457   HARRY      ARM        5100   BRAINSTORMING
   5   23MAR1977    51   CYNTHIA    JOHNSON    1000   PERSONNEL
   6   14DEC1977   119   CHARLES    BOWER      4000   PUBLIC RELATIONS
   7   07JUL1977   158   JOCK       JACKSON    4000   PUBLIC RELATIONS
   8   08SEP1977   149   LAURA      PENMAN     4000   PUBLIC RELATIONS
   9   21JAN1977     3   JENNIFER   GARFIELD   3100   INTERNAL SOFTWARE
```

Reestablishing Currency on a Record

This example illustrates how a program can reestablish currency on a record to complete set navigation after accessing a record that is not contained in the current set occurrence.

In this example, a transaction SAS data set, WORK.EMPLOYEE, supplies a CALC key value for the OBTAIN of an EMPLOYEE record. COVERAGE records are then obtained within the current EMP-COVERAGE set occurrence. PLANCODE values from employee COVERAGE records provide links to INSURANCE-PLAN records through a CALC key. Once current on INSURANCE-PLAN, the program gathers data and uses a stored database key to return to the current COVERAGE record. At that point, the next COVERAGE record in the current set occurrence of EMP-COVERAGE can be obtained. The output data set consists of observations which contain employee, coverage, and related insurance plan data. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;

❷ data work.employee;
     input empnum $4.;
  datalines;
  0007
  0471
  0000
  0301
  0004
  ;

  data work.empplan;
     drop covdbkey empnum;

❸  infile empss01 idms func=func record=recname
          ikey=ckey ikeylen=keyl errstat=stat
          sequence=seq set=inset area=subarea
          dbkey=dkey;


   /* BIND records to be accessed */

❹  if _n_ = 1 then do;
      func       = 'BIND';
      recname    = 'EMPLOYEE';
      input;
      if stat ne '0000' then go to staterr;

      recname    = 'INSURANCE-PLAN';
      input;
      if stat ne '0000' then go to staterr;

      recname    = 'COVERAGE ;
      input;
      if stat ne '0000' then go to staterr;
   end;
```

```
    /* OBTAIN EMPLOYEE record using CALC key */
    /* value */

❺   set work.employee;
    func       = 'OBTAIN';
    seq        = ' ';
    inset      = ' ';
    ckey       = empnum;
    keyl       = 4;
    recname    = 'EMPLOYEE';
    input @;
    if stat not in ('0000', '0326') then go to
         staterr;
    if stat = '0000' then do;
       input @1   employee_id    4.0
             @5   firstname      $char10.
             @15  lastname       $char15.;


       /* OBTAIN COVERAGE records for EMPLOYEE */

❻      seq        = 'FIRST';
       do while (stat = '0000');
         func       = 'OBTAIN';
         keyl       = 0;
         ckey       = ' ';
         dkey       = ' ';
         recname    = 'COVERAGE';
         inset      = 'EMP-COVERAGE';
         input @;
         if stat not in ('0000', '0307') then go
              to staterr;
         if stat = '0000' then do;
            input @13  type     $1.
                  @14  plancode $3.;


            /* ACCEPT CURRENT database key */

❼          func       = 'ACCEPT';
            seq        = 'CURRENT';
            dkey       = ' ';
            input;
            if stat ne '0000' then go to staterr;
            covdbkey   = dkey;


            /* FIND INSURANCE-PLAN using CALC */

❽          func       = 'FIND';
            ckey       = plancode;
            keyl       = 3;
            seq        = '       ';
```

```
              recname    = 'INSURANCE-PLAN';
              inset      = ' ';
              dkey       = ' ';
              input;
              if stat ne '0000' then go to
                  staterr;


              /* OBTAIN CURRENT INSURANCE-PLAN */
              /* record                        */

❾             func       = 'OBTAIN';
              seq        = 'CURRENT';
              ckey       = ' ';
              keyl       = 0;
              recname    = ' ';
              subarea    = ' ';
              input @;
              if stat ne '0000' then go to staterr;
              input @4   company_name  $45.
                    @105 group_number  6.0
                    @111 plndeduc      PD5.2
                    @116 maxlfcst      PD5.2
                    @121 famlycst      PD5.2
                    @126 depcost       PD5.2;
              output;


              /* FIND COVERAGE using stored  */
              /* database key                */

❿             func       = 'FIND';
              seq        = ' ';
              recname    = 'COVERAGE';
              dkey       = covdbkey;
              input;
              if stat ne '0000' then go to staterr;
              seq = 'NEXT';
           end;
         end;
      end;


⓫     else do;
         put 'WARNING: No EMPLOYEE record for CALC=
                'ckey;
         put 'WARNING: Execution continues with next
                 EMPLOYEE.';
         _error_ = 0;
      end;

⓬     _error_ = 0;
    return;
```

```
⓭ staterr:
      put @1 'ERROR: ' @10 func @17 'RETURNED
                STATUS =' @37 stat;
      put @1 'ERROR: INFILE parameter values are: ';
      put @1 'ERROR: ' recname= ckey= keyl= seq=
                inset= subarea= dkey=;
      put @1 'ERROR: DATA step execution
                terminating.';
      _error_ = 0;
      stop;
   run;

   proc print data=work.empplan;
      title 'Employee Coverage and Plan Record
             Information';
   run;
```

❶ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the OPTIONS statement.

❷ This DATA step execution creates the transaction data set WORK.EMPLOYEE. The 4-byte character variable EMPNUM contains CALC key values that will be used to access EMPLOYEE records directly by employee ID.

❸ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the INFILE statement.

❹ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the BIND RECORD statement.

❺ The current EMPNUM value from WORK.EMPLOYEE is used as a CALC key to obtain an EMPLOYEE record from the database. KEYL specifies the length of the CALC key. The **`INPUT @;`** statement submits the call and places a hold on the input buffer so that the status code can be checked. For any unexpected status code, execution branches to the STATERR label. If the status code is 0000, the INPUT statement maps data from the input buffer to the PDV and then releases the buffer.

❻ The DO WHILE loop obtains COVERAGE records for the current employee in the EMP-COVERAGE set. When all COVERAGE records in the set have been obtained, the status code is set to 0307, and the loop terminates. At that point, the DATA step obtains the next EMPLOYEE as specified by the CALC value read from WORK.EMPLOYEE. The **`INPUT @;`** statement submits the OBTAIN FIRST/NEXT call and places a hold on the input buffer while the status code is checked. For any unexpected status codes, execution branches to the STATERR label. For a successful OBTAIN call, the INPUT statement maps coverage information from the input buffer to the specified variables in the PDV and releases the input buffer. The PLANCODE variable now contains a CALC key value that can be used to directly access related INSURANCE-PLAN record information.

❼ The next DML call generated is an ACCEPT CURRENT, which takes the current database key of the COVERAGE record and stores it in the variable defined by the DBKEY= INFILE parameter, DKEY. The null INPUT statement submits the ACCEPT call but does not place a hold on the input buffer because ACCEPT returns no data. For any status code other than 0000, execution branches to the STATERR label. For a successful ACCEPT call, the value returned to DKEY is moved into variable COVDBKEY to be used in a later call. By storing the database key of this record for later use, the program can regain currency on the record.

8 Now that the database key of the COVERAGE record is stored, a FIND call is generated to locate and establish currency on the related INSURANCE-PLAN record. The FIND call uses the CALC value stored in PLANCODE. To issue this call, the DKEY field is set to blank. The null INPUT statement submits the call to CA-IDMS but no hold is placed on the input buffer because FIND does not return data. For any status code other than 0000, execution branches to the STATERR label.

9 After the INSURANCE-PLAN record has been successfully located, an OBTAIN CURRENT call is generated to request that the record be retrieved. The **`INPUT @;`** statement submits the generated call and places a hold on the input buffer so that the returned status code can be checked. For any status code other than 0000, execution branches to the STATERR label. For a successful OBTAIN, the INPUT statement maps INSURANCE-PLAN data from the input buffer to the specified variables in the PDV. At this point, an observation is written to output data set WORK.EMPPLAN that contains related EMPLOYEE, COVERAGE, and INSURANCE-PLAN information.

10 Currency must be reestablished on the COVERAGE record so that the DO WHILE loop can obtain the NEXT COVERAGE record in the current set occurrence of EMP-COVERAGE. A FIND call is generated using the stored database key in COVDBKEY. This call locates the correct COVERAGE record occurrence. The null INPUT statement submits the generated call, but no hold is placed on the input buffer since FIND establishes a position in the database rather than returning data. For any status code other than 0000, execution branches to the STATERR label. If the FIND is successful, currency has been reestablished, and SEQ is assigned a value of NEXT to generate OBTAIN NEXT COVERAGE.

11 This group of statements enables execution to continue when no EMPLOYEE record exists for the CALC value specified in the transaction data set. In this case, an informative WARNING message is written to the SAS log and _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

12 At this point, the STAT variable must have a value of 0307, which indicates that all COVERAGE records for the specified EMPLOYEE have been accessed. Since this code is non-zero, _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

13 See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.6 Reestablishing Currency on a Record

```
             Employee Coverage and Plan Record Information

           employee_
   Obs         id       firstname    lastname      type    plancode

     1          7        MONTE       BANK           F        004
     2        471        THEMIS      PAPAZEUS       F        003
     3        471        THEMIS      PAPAZEUS       F        002
     4        471        THEMIS      PAPAZEUS       M        001
     5        301        BURT        LANCHESTER     D        004
     6        301        BURT        LANCHESTER     F        003
     7        301        BURT        LANCHESTER     F        002
     8        301        BURT        LANCHESTER     M        001
     9          4        HERBERT     CRANE          F        004
    10          4        HERBERT     CRANE          F        003
    11          4        HERBERT     CRANE          M        001


                                                    group_
   Obs     company_name                             number

     1     TEETH R US                               545598
     2     HOLISTIC GROUP HEALTH ASSOCIATION        329471
     3     HOMOSTASIS HEALTH MAINTENANCE PROGRAM    952867
     4     PROVIDENTIAL LIFE INSURANCE              347815
     5     TEETH R US                               545598
     6     HOLISTIC GROUP HEALTH ASSOCIATION        329471
     7     HOMOSTASIS HEALTH MAINTENANCE PROGRAM    952867
     8     PROVIDENTIAL LIFE INSURANCE              347815
     9     TEETH R US                               545598
    10     HOLISTIC GROUP HEALTH ASSOCIATION        329471
    11     PROVIDENTIAL LIFE INSURANCE              347815


   Obs     plndeduc    maxlfcst    famlycst    depcost

     1         50           0        5000        1000
     2        200           0         200         200
     3          0           0      900000      100000
     4          0      100000           0           0
     5         50           0        5000        1000
     6        200           0         200         200
     7          0           0      900000      100000
     8          0      100000           0           0
     9         50           0        5000        1000
    10        200           0         200         200
    11          0      100000           0           0
```

Using RETURN and GET Across Executions of the DATA Step

This example contains two separate DATA steps and demonstrates the use of the RETURN and GET calls across executions of the DATA step. The first DATA step creates an output data set containing index values from EMP-NAME-NDX. The RETURN command is used to navigate the index set. The index values stored in WORK.EMPSRTKY are used to locate EMPLOYEE records in the second DATA step. Once a record is located, a GET call moves the record data to the input buffer. The numbers in the program correspond to the numbered comments following the program.

```
❶ *options $idmdbug;
  data work.empsrtky;
    length namekey $ 25;
    keep namekey;

❷   infile empss01 idms func=func sequence=seq
           dbkey=dkey sortfld=skey errstat=stat
           set=inset;


    /* RETURN EMP-NAME-NDX key values to store */
    /* in EMPSRTKY data set                    */

❸   func    = 'RETURN';
    seq     = 'FIRST';
    inset   = 'EMP-NAME-NDX';
    skey    = ' ';
    dkey    = ' ';

❹   do until (stat ne '0000');
       input;
       if stat not in ('0000', '1707') then go to
           staterr;
       if stat = '0000' then do;
          namekey = skey;
          output;
          dkey  =  ' ';
          skey  =  ' ';
          seq   =  'NEXT';
       end;
    end;

❺   _error_ = 0;
    stop;

❻ staterr:
    put @1 'ERROR: ' @10 func @17 'RETURNED
              STATUS =' @37 stat ;
    put @1 'ERROR: INFILE parameter values are: ';
    put @1 'ERROR: ' seq= inset= dkey= skey=;
    put @1 'ERROR: DATA step execution
              terminating.';
    _error_ = 0;
    stop;
  run;

  proc print data=work.empsrtky;
    title1 'This is a List of Index Entries from
       EMP-NAME-NDX';
  run;

  data work.employee;
     drop namekey;
```

```
❼  infile empss01 idms func=func sortfld=skey
           ikeylen=keyl errstat=stat set=inset
           record=recname;


     /* BIND the record to be accessed */

❽  if _n_ = 1 then do;
       func  =  'BIND';
       recname  =  'EMPLOYEE';
       input;
       if stat ne '0000' then go to staterr;
    end;


    /* Read NAMEKEY values from EMPSRTKY and */
    /* FIND EMPLOYEE using the EMP-NAME-NDX  */

❾  set work.empsrtky;
    func      =  'FIND';
    recname   =  'EMPLOYEE';
    inset     =  'EMP-NAME-NDX';
    skey      =  namekey;
    keyl      =  25;
    input;
    if stat not in ('0000', '0326') then go to
        staterr;
    if stat = '0000' then do;
       func      =  'GET';
       recname   =  ' ';
       inset     =  ' ';
       skey      =  ' ';
       keyl      =  0;
       input @;
       if stat ne '0000' then go to staterr;
       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      yymmdd6.
             @103 termdate       6.0
             @109 birthdate      yymmdd6.;
       output;
    end;
❿  else do;
       put @1 'WARNING: No EMPLOYEE record with
                 name = ' namekey;
       put @1 'WARNING: Execution continues with
```

```
                next NAMEKEY';
          _error_ = 0;
       end;
    return;
⓫ staterr:
       put @1 'ERROR: ' @10 func @17 'RETURNED
                 STATUS =' @37 stat ;
       put @1 'ERROR: INFILE parameter values are: ';
       put @1 'ERROR: ' inset= skey= keyl= recname=;
       put @1 'ERROR: DATA step execution
                 terminating.';
       _error_ = 0;
       stop;
    run;
    proc print data=work.employee;
      format startdate birthdate date9.
      title1 'This is a List of Employee Information
                Obtained';
      title2 'Using a Transaction Data Set
                Containing Name Index Values';
    run;
```

❶ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the OPTIONS statement.

❷ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the INFILE statement.

❸ Parameter values are initialized to generate the RETURN CURRENCY SET call for the entries in the EMP-NAME-NDX index set. The SKEY and DKEY variables are set to blank and will be assigned the sort key and database key values returned from the call.

❹ In the DO UNTIL loop, the null INPUT statement submits the generated RETURN CURRENCY SET FIRST/NEXT call. The call returns sort key and database key values to the SKEY and DKEY variables. For any unexpected status code, execution branches to the STATERR label. For a successful call, the SKEY value is assigned to NAMEKEY, the current NAMEKEY is written to WORK.EMPSRTKY, SKEY and DKEY variables are reset to blank, and SEQ is set to NEXT. The next iteration of the DO UNTIL loop will return the next index entry.

The DO UNTIL loop executes as long as STAT equals 0000. When the index set has been traversed and all sort values returned and stored in output data set WORK.EMPSRTKY, CA-IDMS returns a 1707 status code, which terminates the loop.

❺ When the DO UNTIL loop terminates, _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log. The index set is traversed in the DO UNTIL loop during the first DATA step iteration, so a STOP statement is used to prevent the DATA step from executing again. Without the STOP statement, the DATA step would loop endlessly, traversing the same index set once for each iteration.

❻ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the STATERR statements.

❼ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the INFILE statement.

❽ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the BIND RECORD statement.

❾ The WORK.EMPSRTKY data set, which was created in the first DATA step, serves as a transaction data set. Each iteration of this DATA step reads a new sort key value, NAMEKEY, and uses it to locate an EMPLOYEE record via the EMP-NAME-NDX. The DATA step terminates when all observations have been read from WORK.EMPSRTKY. To gather employee information, INFILE parameter variables are initialized to generate the FIND EMPLOYEE WITHIN EMP-NAME-NDX call using the supplied sort key from NAMEKEY. The IKEYLEN= parameter variable KEYL is set to 25 to indicate the sort key length. The null INPUT statement submits the FIND call but places no hold on the input buffer because no record data is returned. For any unexpected status code, execution branches to the STATERR label. For a successful FIND, a GET call is generated to request that the record data be retrieved. The **`INPUT @;`** statement submits the GET call and places a hold on the input buffer so the status code can be checked. Any status code not equal to 0000 branches execution to the STATERR label. If the GET call is successful, the INPUT statement maps EMPLOYEE data from the input buffer to the specified variables in the PDV. The contents of the PDV are then written as an observation to output data set WORK.EMPLOYEE.

❿ This group of statements enables execution to continue when no EMPLOYEE record exists for the sort key value specified in the transaction data set. In this case, an informative WARNING message is written to the SAS log and _ERROR_ is reset to 0, which prevents the contents of the PDV from being written to the SAS log.

⓫ See "Statements Common to All SAS/ACCESS DATA Step Examples" on page 43 for a description of the STATERR statements.

The following output shows a portion of the output from this program.

Output 3.7 Using RETURN and GET

```
                This is a List of Index Entries from EMP-NAME-NDX

                         Obs            namekey

                           1    ANDALE         ROY
                           2    ANGELO         MICHAEL
                           3    ARM            HARRY
                           4    BANK           MONTE
                           5    BLOOMER        JUNE
                           6    BOWER          CHARLES
                           7    BREEZE         C.
                           8    CLOTH          TERRY
                           9    CLOUD          BETH
                          10    CRANE          HERBERT
                          11    CROW           CAROLYN
                          12    DONOVAN        ALAN
                          13    DOUGH          JANE
                          14    FERNDALE       JANE


                  This is a List of Employee Information Obtained
             Using a Transaction Data Set Containing Name Index Values

        employee_
    Obs     id     firstname  lastname   street               city        state

      1    466     ROY        ANDALE     44 TRIGGER RD        FRAMINGHAM   MA
      2    120     MICHAEL    ANGELO     507 CISTINE DR       WELLESLEY    MA
      3    457     HARRY      ARM        77 SUNSET STRIP      NATICK       MA
      4      7     MONTE      BANK       45 EAST GROVE DR     HANIBAL      MA
      5     69     JUNE       BLOOMER    14 ZITHER TERR       LEXINGTON    MA
      6    119     CHARLES    BOWER      30 RALPH ST          WELLESLEY    MA
      7    467     C.         BREEZE     200 NIGHTINGALE ST   FRAMINGHAM   MA
      8    479     TERRY      CLOTH      5 ASPHALT ST         EASTON       MA
      9    371     BETH       CLOUD      3456 PINKY LN        NATICK       MA
     10      4     HERBERT    CRANE      30 HERON AVE         KINGSTON     NJ
     11    334     CAROLYN    CROW       891 SUMMER ST        WESTWOOD     MA
     12    366     ALAN       DONOVAN    6781 CORNWALL AVE    MELROSE      MA
     13     24     JANE       DOUGH      15 LOCATION DR       NEWTON       MA
     14     32     JANE       FERNDALE   60 FOREST AVE        NEWTON       MA


    Obs  zip      phone    status  ssnumber   startdate  termdate  birthdate

      1 03461  6175541108    03    027601115  15JUN1978      0     04MAR1960
      2 01568  6178870235    01    127675593  08SEP1979      0     05APR1957
      3 02178  6174320923    05    028770147  01DEC1977      0     05APR1934
      4 02415  6173321933    01    022446676  30APR1978      0     01JAN1950
      5 01675  6175555544    01    039557818  05MAY1980      0     25APR1960
      6 01568  6178841212    01    092345812  14DEC1977      0     04MAR1939
      7 03461  6175542387    01    111556692  02JUN1979      0     04MAY1934
      8 05491  6177738398    01    028701666  02NOV1979      0     04MAR1945
      9 02178  6174321212    01    326710472  04MAR1977      0     09SEP1945
     10 21341  2013341433    01    016777451  14MAY1977      0     21MAR1942
     11 02090  6173291776    01    023980110  17JUN1979      0     03APR1944
     12 02176  6176655412    01    025503622  10OCT1981      0     17NOV1951
     13 02456  6174458155    01    022337878  08AUG1976      0     29MAR1951
     14 02576  6178888112    01    034567891  09SEP1979      0     17JAN1958
```

APPENDIX 1 IDMS Essentials

Introduction to IDMS Essentials

This appendix introduces SAS users to Computer Associates' Integrated Database Management System (CA-IDMS). It focuses on the terms and concepts that help you access CA-IDMS files with SAS/ACCESS software.

If you want more information about a CA-IDMS concept or term, see the documents listed in "CA-IDMS Documentation" on page 83.

Data Dictionaries and the DDS

CA-IDMS enables you to build one or more databases using a data dictionary. A *data dictionary* is itself a CA-IDMS database that contains all the data and system definitions for one or more databases.

A data dictionary is divided logically into areas. The information is organized into entity types, which correspond to the main data processing components, such as elements, records, files, programs, and users. Data dictionaries monitor most aspects of the database environment, from tracking the status of terminals, systems, and users to being a central resource of information about the system and providing security. Some large information systems use multiple dictionaries; for example, a system might have one dictionary for each division of a company.

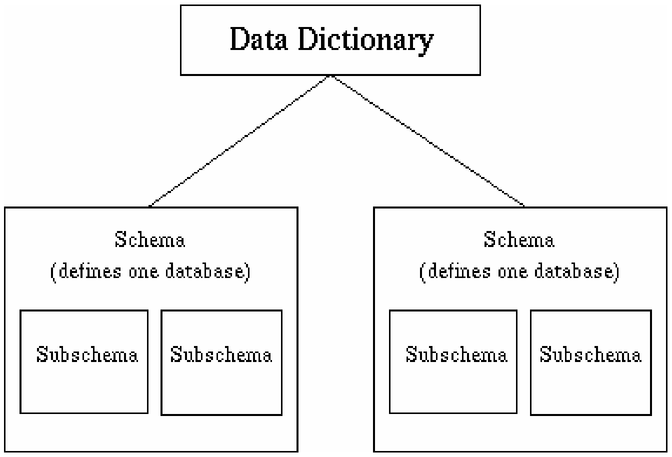
A *database administrator* (DBA) manages and maintains the data dictionaries and the entire CA-IDMS system. DBA duties often include programming systems, managing resources, monitoring the system's performance, and overseeing its security. The DBA has a key role in the SAS/ACCESS interface to CA-IDMS, which is explained in more detail in this section.

Within a CA-IDMS data dictionary are the definitions for a database's schema and subschema. A *schema* describes the contents and structure of a single database, including all of the records and sets that are necessary to define its data elements and data relationships.

A *subschema* is a subset of a schema that is used by programs at runtime. It consists of all the data elements, records, sets, and areas that are defined in the schema or a subset thereof. It includes database records and can include logical records as well as logical-record paths (defined below). The DBA defines logical records and their paths in the subschema before application programs are coded and executed.

The following figure illustrates the relationships among the data dictionary, schemas, and subschemas.

Figure A1.1 Data Dictionary, Schemas, and Subschemas



CA-IDMS provides two operating environments, or modes, for accessing data dictionaries and databases. In the *central version*, multiple concurrently executing programs access the database(s) through one shared copy of the database management system (DBMS). The central version controls concurrent updating of the database by multiple users in order to maintain database integrity.

In *local mode*, one program at a time accesses the database through a dedicated copy of the DBMS. You cannot run local mode against a database at the same time that the central version is accessing it.

A *Distributed Database System (DDS)* distributes data storage and processing functions among several systems. These systems can execute on one or more computers and at one or more sites. Each system is a node in the DDS configuration. A central version specifies which node within the DDS system to access.

CA-IDMS Networks and Sets

Each CA-IDMS database consists of database records that are grouped into record types. A *record type* consists of the record's name, all of its elements, and the elements' attributes, such as data types and sizes. These record types are linked together through different logical groups called sets. Sets are defined to the schema.

A *set* is a logical relationship established between two or more named record types. One record type is the owner of the set and the other record types are members. Record types can belong to more than one set, so a record type can be both an owner of one set and a member of another. That same record type can also be a member of more than one set. These sets and their interweaving relationships make up a *network* and give CA-IDMS its network capabilities.

To move through the database, each record type contains pointers to other record types in its set or sets. *Pointers* identify the next record in the set and link the records together in a chain. There are three kinds of pointers:

Next pointer (required pointer)
: points to the next record type in the set, regardless of whether the record type is an owner or a member of the set.

Prior pointer
: works the same way as the Next pointer except that it points to the prior record type.

Owner pointer
: points from a member record type to the owner record type.

Through these pointers, a program can navigate through the network and travel a specific path through one or many sets.

The database administrator is responsible for defining record types and sets in the schema.

CA-IDMS Documentation

You might find the following Computer Associates' Release 12.0 CA-IDMS documentation helpful while you are using the SAS/ACCESS interface to CA-IDMS. Refer to these manuals for information about your CA-IDMS system and DML application programming.

- *System Operations*
- *Database Administration*
- *Security Administration*
- *System Generation*
- *Features Summary*
- *Messages and Codes*
- *System Tasks and Operator Commands*
- *Utilities*
- *Database Design*
- *Quick Reference*
- *Programming Quick Reference*
- *Master Index*
- *Glossary*
- *Navigational DML Programming*

APPENDIX
2

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- □ *SAS Language Reference: Concepts*
- □ *SAS Language Reference: Dictionary*
- □ *Base SAS Procedures Guide*
- □ *Getting Started with the SAS System in the z/OS Environment*
- □ *SAS/CONNECT User's Guide*
- □ *SAS/GRAPH Software: Reference, Volumes 1 and 2*
- □ *SAS/STAT User's Guide, Volumes 1, 2, and 3*

For a complete list of SAS publications, go to **support.sas.com/bookstore**. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative at:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: 1-800-727-3228
Fax: 1-919-531-9439
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/bookstore**

Customers outside the United States and Canada, please contact your local SAS office for assistance.

Index

Your Turn