CHAPTER 3

Examples of SAS/ACCESS DATA Step Programs

Introduction

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

All of the examples in this chapter can be executed against 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 Examples

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

OPTIONS

The $IDMDBUG system option tells the SAS System 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 chapter 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 on each INFILE parameter, see “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 addressibility for a named record. In each of these examples, a null INPUT statement issues a BIND RECORD statement for each record (see “Using the Null INPUT Statement” on page 34). 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 chapter 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 on dealing with status codes, see “Checking Call Status Codes” on page 35.

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;
3       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 */

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


     /* FIND and OBTAIN NEXT DEPARTMENT record */

5    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;


6    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 Examples” on page 45 for a description of the OPTIONS statement.

❷ See “Statements Common to All Examples” on page 45 for a description of the INFILE statement.

❸ See “Statements Common to All Examples” on page 45 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 are returned to the input buffer (see “Using the Null INPUT Statement” on page 34). 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 35.) 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 Examples” on page 45 for a description of the STATERR statements.

Output 3.1 on page 49 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 re-established on the EMPLOYEE record after obtaining OFFICE information and prior to 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  */

⓬           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                              */


⓭    _error_ = 0;
    return;


⓮  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 Examples" on page 45 for a description of the OPTIONS statement.

❷ See “Statements Common to All Examples” on page 45 for a description of the INFILE statement.

❸ See “Statements Common to All Examples” on page 45 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 are 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 are moved from the input buffer to the PDV by executing the INPUT statement. The first **`INPUT @;`** statement forces the call to be submitted and allows a returned status code to be checked prior to 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 re-establish 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.

10 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.

11 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.

12 If the status code indicates end-of-set (0307) then the status variable is reset to 0000. The assignment is done to allow the DO UNTIL loop (see 6) 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.

13 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.

14 See “Statements Common to All Examples” on page 45 for a description of the STATERR statements.

Output 3.2 on page 56 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                   */

7   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                   */

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


/* OBTAIN current record using the DBKEY */

9      func       = 'OBTAIN';
       seq        = '       ';
       inset      = '        ';
       input @1 nxtrec $29. @;
       if stat ne '0000' then go to staterr;
       end;
     end;
   end;

10  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 Examples” on page 45 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 Examples” on page 45 for a description of the INFILE statement.

❹ See “Statements Common to All Examples” on page 45 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 are moved into the PDV from the input buffer, and another observation is output for the current employee.

❿ This group of statements allows 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 Examples" on page 45 for a description of the STATERR statements.

Output 3.3 on page 62 shows a portion of the output from this program.

Output 3.3 Using a SAS Data Set as a Transaction File

```
                          Positions Held by Specifed 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 Examples” on page 45 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 Examples” on page 45 for a description of the INFILE statement.

❹ See “Statements Common to All Examples” on page 45 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 allows 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 Examples” on page 45 for a description of the STATERR statements.

Output 3.4 on page 66 shows a portion of the output from this program.

Output 3.4 Locating 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 Examples” on page 45 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 Examples” on page 45 for a description of the INFILE statement.

❹ See “Statements Common to All Examples” on page 45 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 are 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.

❾ 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.

➓ 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.

⓫ See “Statements Common to All Examples” on page 45 for a description of the STATERR statements.

Output 3.5 on page 71 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
```

Re-establishing Currency on a Record

This example illustrates how a program can re-establish 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                */
```

```
⑩           func       = 'FIND';
             seq        = ' ';
             recname    = 'COVERAGE';
             dkey       = covdbkey;
             input;
             if stat ne '0000' then go to staterr;
             seq = 'NEXT';
          end;
        end;
     end;


⑪   else do;
        put 'WARNING: No EMPLOYEE record for CALC=
               'ckey;
        put 'WARNING: Execution continues with next
                EMPLOYEE.';
        _error_ = 0;
     end;

⑫   _error_ = 0;
   return;

⑬ 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 Examples” on page 45 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 Examples” on page 45 for a description of the INFILE statement.

❹ See “Statements Common to All Examples” on page 45 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.

❽ 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.

❾ 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.

❿ Currency must be re-established 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 re-established, and SEQ is assigned a value of NEXT to generate OBTAIN NEXT COVERAGE.

⓫ This group of statements allows 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.

⓬ 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.

⓭ See “Statements Common to All Examples” on page 45 for a description of the STATERR statements.

Output 3.6 on page 77 shows a portion of the output from this program.

Output 3.6 Re-establishing 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.

```
1 *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 */

8     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   */

9     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;
10    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 Examples" on page 45 for a description of the OPTIONS statement.

❷ See "Statements Common to All Examples" on page 45 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 Examples" on page 45 for a description of the STATERR statements.

❼ See “Statements Common to All Examples” on page 45 for a description of the INFILE statement.

❽ See “Statements Common to All Examples” on page 45 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 interation 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 are 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 allows 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 Examples” on page 45 for a description of the STATERR statements.

Output 3.7 on page 82 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
```

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

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

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

ISBN 1–58025–547–7

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

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

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

1st printing, October 1999

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

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

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