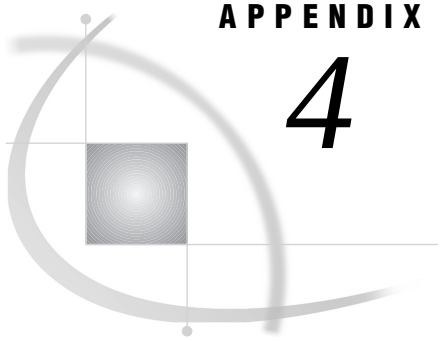

APPENDIX

4

SAS Component Language (SCL) Application

Introduction

The SAS Component Language (SCL) application that is presented in this appendix, when used with the FSEDIT procedure or the FSEDIT command, implements an order-processing and inventory-maintenance system. Because the purpose of this example is to illustrate SCL programming techniques, certain error-handling and user-friendly enhancements were omitted to keep the example as clear as possible.

This example exploits the features of SAS/SHARE software that allow several users to update a SAS data set at the same time. That capability automates maintenance of the inventory data and allows order data to be maintained centrally, which in turn can facilitate analysis of the orders that have been received.

Note: SAS Component Language is new terminology for SAS Screen Control Language. SAS Component Language and SAS Screen Control Language are synonyms. △

Audience

This example is for SAS application programmers who are familiar with SAS Component Language and with the FSEDIT procedure and FSEDIT command.

Inventory and Order System

There are three general types of data that are involved in an order-processing system:

- Customer information, which includes name, address, and a customer number that uniquely identifies the customer.
- Inventory information, which tracks the amount of each item that is in stock at any given instant.
- Orders information, which identifies items in inventory that a customer has placed an order for.

Customer Information

In order to simplify the example, this application does not include a customer data set. This SCL program, which is expanded for use in an actual order-processing system, would open the customer data set during the FSEINIT step program and close it during the FSETERM step.

The customer data set is an ideal candidate to access through a SAS/SHARE server to keep information up-to-date, such as customers' address changes, contact information, and so forth.

For the purpose of entering orders, the customer data set would usually be opened for read-only access through the server because customer information is not usually updated at the time an order is taken.

Updating the customer data would ordinarily be done through a second FSEDIT application that would also access the data set through the server. That application would be available to customer service representatives and administrative personnel.

Inventory Information

In this example, the inventory data is accessed through a SAS/SHARE server. The data is completely hidden from the user of the order entry application. The point of this example is to show how to maintain information automatically in a SAS data set.

The inventory file is also an ideal candidate for access through a SAS/SHARE server because it allows the available quantity of each item to be continually accurate.

In the order-processing example, the inventory data set is opened for update access through the server. Read-only access to the inventory data set might be given for the reporting programs (run either nightly or on-demand), which would also access the data set through the server. The inventory reporting and analysis programs are not included in this example.

In a mature inventory-management system, there would be additional access to the inventory data by employees who receive merchandise. This example shows only the order-processing side (the "inventory depletion" process), but it is important to remember that replenishing inventory must also take place. Accessing the inventory data through a SAS/SHARE server allows both the ordering and receiving operations to continually maintain accurate information in the inventory file.

Orders Information

The orders data set is a series of transactions. When a product is ordered, that event is recorded in the orders data set. This makes the orders data set a time-based record

of events, which is quite different from the type of information that is maintained in the customer or the inventory data sets.

Accessing the orders data set through a SAS/SHARE server allows all order information to be captured in a single file. This enables simplified reporting and analysis of the orders data, which can be performed on current information at any time.

In this example, the data set ORDERS is updated by the users of the application. Reporting and analysis of the orders data could be done with read-only access to the data set ORDERS through the server. Reporting and analysis of the orders data is not illustrated here.

The Inventory/Order System SCL Application

The following sample application is referred to in "SCL Programming Considerations" on page 50. See that section for more information about updating concurrently shared data in SCL applications.

```
/*-------------------------------------------------------------------

 *  Inventory/Order System SCL application for use with PROC FSEDIT
 *  when editing an orders data set.
 *
 * CAUTION:
 * + The deletion of a non-null order (quantity>0) results in an
 *   error message being written to the SAS log because the inventory
 *   data set will not have been updated to reflect the returned
 *   inventory.
 * + Do not issue a CANCEL of a new order (not yet added to the
 *   data set ORDERS) with a DELETE command. In this case the
 *   program will not detect a cancel or delete condition and will
 *   debit the inventory for the quantity in the cancelled order.
 *
 *  The SCL program included here is designed to run with the following
 *  setup and data set prototype:
 *
 *  The data set ORDERS has these variables:
 *
 *  o  PRODUCT   type=character  /* Product Code    */
 *  o  QUANTITY  type=numeric    /* Amount of Order */
 *
 *  The INVENTOR data set has these variables:
 *
 *  o  CODE      type=character  /* Product Code        */
 *  o  DESC      type=character  /* Product Description */
 *  o  INVENT    type=numeric    /* Stock on hand       */
 *
 *
 *
 *  For information about selecting a communications access method
 *  and server name, see Chapter 3.
 *  To start a SAS/SHARE server to access the data that is used by this
 *  example, execute these SAS statements in a SAS session:
 *
```

```
 * OPTIONS COMAMID=communications access method;
 *  LIBNAME DLIB 'physical name';
 *  DATA DLIB.INVENTOR;
 *   CODE='ABC'; DESC='PRODUCT ABC'; INVENT=100;
 *  RUN;
 *  DATA DLIB.ORDERS;
 *   PRODUCT='ABC'; QUANTITY=20;
 *  RUN;
 *  PROC SERVER  ID=server name;  run;
 *
 *  To create a client SAS session that you can use to execute this
 *  example, execute these SAS statements in a second SAS session:
 *
 *  OPTIONS COMAMID=communications access method;
 *  LIBNAME DLIB SERVER=optional computer name.server name;
 *  /* EDIT AND COMPILE SCL PROGRAM ON SCREEN AND RUN IT */
 *  PROC FSEDIT  DATA=DLIB.ORDERS
 *               SCREEN=DLIB.DISPLAY.ORDERS.SCREEN;  RUN;
*-------------------------------------------------------------------*/


length rc 8 ;      /* System return code storage */
length invent 8 ;  /* Current n of items inventoried*/

FSEINIT:
    /*-----------------------------------------------------------------

    / Open the product control data set and save the needed variable
    / numbers. "Control term" ensures non-null deletions can be
    / detected in TERM.
    /-----------------------------------------------------------------*/

    codeid=open('dlib.inventor','U');
    vdesc=varnum(codeid,'desc');
    vinvent=varnum(codeid,'invent');
    control term;
    return;

INIT:
    /*-----------------------------------------------------------------

    / Save initial order values for later.  For a pre-existing order,
    / get the inventory info (item description) for the display, and
    / do not forget to unlock the record.  Also prohibit *changing*
    / the product code on a pre-existing order by using the FIELD
    / function.
    /-----------------------------------------------------------------*/

    _msg_=' ';
    sav_prod=product; sav_quan=quantity;
    if (obsinfo('new')) then do;
       oldorder=0;
       rc=field('unprotect','product');
       if (product=' ') then link needcode;
```

```
      return;
     end;
    oldorder=1;
    link getrec;
    rc=unlock(codeid);
    rc=field('protect','product');
    return;

MAIN:
    /*-----------------------------------------------------------------

    / For a change in quantity or a new order, fetch (and lock) the
    / inventory record, validate the request, and update the
    / inventory data set.  In either case, if all operations succeed,
    / issue a SAVE command in the primary data set so that the data set
    / cannot be made out-of-sync with the inventory due to a
    / subsequent CANCEL command from the user.
    /-----------------------------------------------------------------*/


    if (_STATUS_='C') then return;

    else if (product=' ') then link needcode;

    else if (sav_quan^=quantity or ^oldorder) then do;

       /* Try to lock inventory record to update. */
        loop_cnt=0;
    lokloop: loop_cnt=loop_cnt+1;
        link getrec;
        if (not gotrec) then return;
        if (rc=%sysrc(_swnoupd)) then do;
           if (loop_cnt<500) then goto lokloop;
           _msg_='Error: Product was locked.';
           erroron product;
           return;
         end;

       /* Check and debit the inventory. */
        link chkquan;
        if (not quanok) then goto unlok;
        invent=invent-quantity;
        call putvarn(codeid,vinvent,invent);
        rc=update(codeid);
        if (sysrc()>0) then do;
           _msg_=sysmsg();
           erroron product;
           goto unlok;
         end;

        /* Force FSEDIT to save the observation so that */
        /* the primary data set will be up-to-date now. */
        call execcmd('save;');
```

```
        /* In case user did not leave observation,      */
        /* clarify that this order is saved.            */
        sav_prod=product; sav_quan=quantity;
        oldorder=1;
    unlok: rc=unlock(codeid);
    end;
    return;

getrec:
    /*-----------------------------------------------------------------
    / Generally, this section fetches the record of the inventory
    / data set that you want.  If it is successful, 'gotrec'
    / will have value 1; else, 0.  This section leaves the fetched
    / record locked.
    /-----------------------------------------------------------------*/

    gotrec=0;
    rc=WHERE(codeid,"code='"||product||"    '");
    if (rc>0) then do;
       _msg_='WHERE: '||sysmsg();
       erroron product; return;
     end;
    rc=FETCH(codeid);
    if (rc>0) then do; /* Error! */
       _msg_='FETCH: '||sysmsg();
       erroron product; return;
     end;
    else if (rc=-1) then do;

       /* Product not found but no error. */
       _msg_='ERROR: The product code is invalid. Please re-enter';
       erroron product; return;
     end;
    else gotrec=1;
    desc=getvarc(codeid,vdesc);
    invent=getvarn(codeid,vinvent);
    return;

chkquan:  /* Check the amount available */
    /*-----------------------------------------------------------------

    / This section checks that the available inventory is sufficient
    / for the quantity that is being requested.  If so, 'quanok' will
    / have value 1; else, 0.  This section may modify 'invent' if a
    / quantity change is being verified.
    /-----------------------------------------------------------------*/

    quanok=1;
    /* If just a quantity change, add back old quantity. */
    if (oldorder) then invent=invent+sav_quan;
    if (quantity=0) then do;
       _msg_='This order is null due to a zero quantity';
       cursor quantity;
     end;
```

```
    else if (quantity>invent) then do;
       _msg_='ERROR: Available stock is ' || put(invent,best.);
       erroron quantity;
       quanok=0;
     end;
    return;

needcode:
    /*-------------------------------------------------------------

    / Ask user to enter product code. Set ERRORON to prevent exiting
    / the observation.
    /-------------------------------------------------------------*/

    _msg_='Please enter a product code';
    desc=' ';
    erroron product;
    return;

TERM:
    /*-------------------------------------------------------------
    / For safety, check if the user accidentally deleted a non-null
    / observation, which we are leaving, and log an error message if so.
    /-------------------------------------------------------------*/

    if (oldorder & sav_quan & obsinfo('deleted')) then
       put 'ERROR: Order consisting of ' sav_quan
           'units of product number ' sav_prod 'has been deleted.';
    return;

FSETERM:
    /*-------------------------------------------------------------

    / Termination: Close the lookup data set if it was indeed
    / successfully opened.
    /-------------------------------------------------------------*/

    if (codeid>0) then rc=close(codeid);
    return;
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/SHARE User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 247.

SAS/SHARE User's Guide, Version 8

ISBN 1–58025–478–0

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

1st printing, September 1999