Driving to Better Credit Policies : The Risk Strategy Instrument Panel

Michael Davis, Bassett Consulting Services, North Haven, Connecticut
Adam Terr, GE Capital Card Services, Stamford, Connecticut

ABSTRACT

The Risk Strategy Instrument Panel is a custom SAS/AF® FRAME Entry application created for the Risk Management Group, Retailer Financial Services, GE Capital Services. The Instrument Panel launches an existing library of MVS (OS/390) batch SAS programs from desktop personal computers. It was designed to let risk analysts and others create and evaluate credit line and authorization strategies more easily.

To make the application easier to use, some innovative design strategies were employed, such as the use of non-visual object classes to provide common functions such as supplying lists of available tape data sets to the GUI. Another interesting feature is the use of Data Tables as list selection widgets. The combination of careful planning of the application menus with an attractive GUI design have resulted in an application that some analysts find fun to use.

INTRODUCTION

GE Capital, with assets of more than US$300 billion, is a global, diversified financial services company with 28 specialized businesses. GE Capital, based in Stamford, CT, provides equipment management, mid-market and specialized financing, specialty insurance and a variety of consumer services, such as car leasing, home mortgages and credit cards, to businesses and individuals around the world.

Not long ago, the Risk Management Group of Retailer Financial Services identified the following unmet needs:

- create a standardized and consistent framework for developing account management strategies and analysis
- friendly way for users with little programming knowledge to launch reports and analyses
- a way for users to share data extracts and summary files and recreate reports without having to rerun the analyses from the start
- to summarize large amounts of data from tape files to disk for quick and easy printing of reports

BUSINESS OBJECTIVES

The Risk Strategy Instrument Panel was created to facilitate the following business objectives:

- create account management strategies (credit line and authorizations) to increase profitable sales and reduce loss on high risk accounts
- to reduce disruptions at the point of sale
- to model and predict financial and operational impacts in account management strategies

MAIN SELECTION SCREEN

In previous projects to build decision support tools for risk analysts, it was learned that users preferred navigating using icon pushbutton menu screens. The menu screens are arranged in a hierarchy.

After displaying the initial welcome screen, the Risk Strategy Instrument Panel displays the following menu:

Figure 1

From this screen, the user can select menus that control the Instrument Panel's functions:

- upload or download strategies files
- extract data from host mainframe
- merge extract data files
- summarize and analyze extract files
- print reports
- perform maintenance tasks

SOFTWARE TOOLS USED

While the Risk Strategy Instrument Panel was engineered to appear as a single seamless applications application to users, it is woven from several different component parts. They include:

- SAS/AF Frames
- non-visual SCL
- SAS macro language
- SAS/CONNECT®
- SyncSort®

The client interface, composed mainly of SAS/AF Frames and runs under Windows 95.

To query the host mainframe for information such as lists of available files, non-visual SCL (Screen Control Language) components are used. SCL catalog entries are uploaded to the host mainframe and remotely run using PROC DISPLAY.

When results consist of a list, they are sent back to the client interface by downloading SLIST entries. Otherwise %SYSRPUT is remotely submitted to move a single macro value back to the client interface.

Since the analyses usually involve working with large data sets, they are run in batch on the host mainframe. Where significant filtering and sorting of flat files is required, SyncSort steps are run ahead of the SAS steps.

INTERFACE DESIGN OBJECTIVES

Before work on the Risk Management Instrument Panel began, the following design principles were set:

- reduce the need to type on keyboard
- show only choices valid for a given situation
- minimize the need for the user to remember data set naming conventions for both input and intermediate extract and summary files
- provide a way for mainframe batch jobs to notify user of completion via e-mail

INSTRUMENT PANEL CONTROLS

SAS/AF Frame entries provide a wealth of objects that can be used to control applications and make selections. To pick which type object to use for a given task, the following criteria was used:

- icons buttons for function selection
- push buttons where screen space limited
- radio boxes for mutually exclusive selections
- check boxes for non-exclusive selections
- list boxes for single dynamic selections
- data table for multiple dynamic selections where multiple columns must be viewed
- text-entry fields that remember previous entries

DATA SELECTION EXAMPLES

A common selection task while using the Risk Strategy Instrument Panel is to select an appropriate month and year. Rather than use a text entry field to select a month, a list box was used. Figure 2 shows the month selection window.

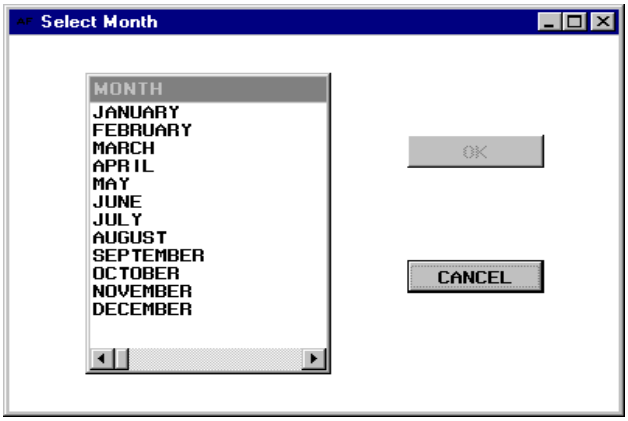

Figure 2

To insure that user selects a month, the OK button is grayed-out until a month is selected. Year selections are made using a radio box as shown in Figure 3.

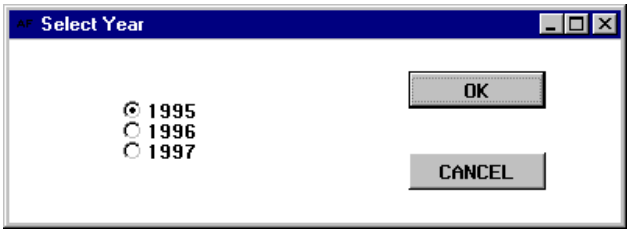

Figure 3

To help the user verify that an analysis or summary program has run correctly, the user is given the option to print out some sample observations. Rather than have the user type in the number of sample observations to be printed, the sample selection screen, shown in Figure 4, combines a check box with a radio box to make the selection. However, analysts not satisfied with the radio box

choices can select other and type in a number of observations to be printed.

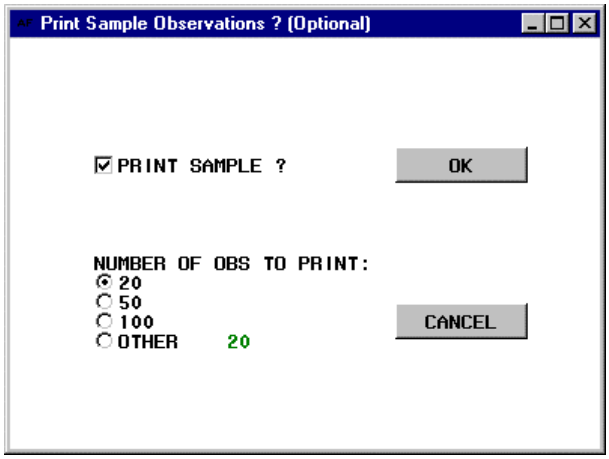

Figure 4

Sometimes it is necessary to show the user more than one item or column in order make selections.

The Data Table object is an excellent way to incorporate this feature into a selection Frame. Figure 5 illustrates a screen used to select portfolios for inclusion in an analysis. This screen illustrates how data tables may be used in Frames as a selection object.

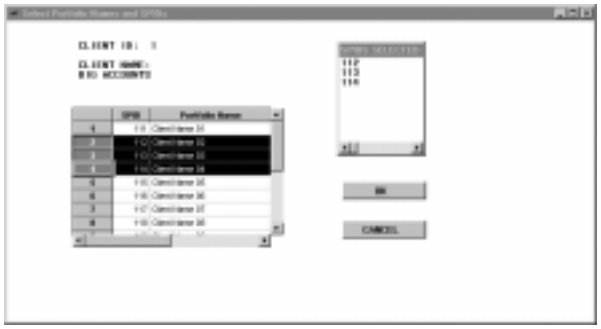

Figure 5

In the upper-right portion of this selection screen is a display only list box that shows the Portfolio ID (SPID) numbers currently selected. This box was added because the data table object is scrollable. Thus, the user may not be able to see all of the observations (rows) that have been selected without this additional aid. As with list boxes, multiple rows may be selected.

DATA TABLES AS SELECTION OBJECTS

What is involved in using a data table as a selection object? Start by opening a blank or relatively empty Frame. Select Action, then Make, and then Data Table. The Data Table Attributes window will appear as shown in Figure 6.

Figure 6

As with all Frame objects, you will probably want to change the name from the default name that begins with OBJ. There is no need to specify the table name, data mode or lock mode; this is best done in the associated SCL entry. If the data set is too big to be seen all at once, Vertical and/or Horizontal scrollbars should be checked. Click on OK to accept the selections.

In the INIT section of SCL entry for the Frame, you will need to insert SCL code similar to the following:

```
call send (_frame_, '_get_widget_',
 'data table name',tbl_id) ;
```

The Call Send statement uses the _Get_Widget_ method upon the current frame *_frame_* to get the table ID, *tbl_id,* for the data table.

```
dsostr= 'spid ( where= (clnt_id=
 ' || put(clnt_id, 2.) || '))' ;
```

The character variable *dsostr* (data set open string) contains the data set name, work.spid and applys a WHERE data set option to subset the data set for the current selection of client ID, *clnt_id*.

```
call send(tbl_id, '_set_dataset_',
 dsostr, 'browse', 'record') ;
```

The Call Send statement uses the _Set_Dataset_ method to open the data set work.spid with the WHERE clause in browse mode with record locking.

```
attr_lst= makelist() ;
rc= insertc(attr_lst,'y',
 -1,'multiple_selections') ;
```

```
rc= insertc(attr_lst,'y',
 -1,'select_rows') ;
```

The temporary attributes SCL list, *attr_lst,* is created. Two attributes, multiple_selections and select_rows, are inserted into the list.

```
call send (tbl_id,
 '_set_attributes_', attr_lst) ;
attr_lst= dellist(attr_lst) ;
```

The Call Send statement associates the attributes list with the data table. Since the attribute list *attr_lst* is no longer needed, it is deleted.

Next, the SCL program must identify the selected observations (row) in the data table. Keep in mind that multiple rows may be selected. The rows might not be contiguous! One must use SCL code similar to the following in order to identify the selected rows. This code should be inserted in the labeled section for the data table object so it will run whenever the table is selected.

```
test_lst= makelist() ;
call send (tbl_id, '_get_selections_',
 test_lst) ;
```

The Call Send statement fills the SCL list test_lst with the information we need to build a selection list

```
item_cnt= listlen(test_lst) ;
slctd_rows_lst= makelist() ;
do i= 1 to item_cnt ;
  cur_lst= getniteml(test_lst,
   left(put(i,4.))) ;
  start_lst= getniteml(cur_lst,
   'start_row') ;
  start_row= getnitemn(start_lst,'1');
  end_lst= getniteml(cur_lst,
   'end_row') ;
  end_row= getnitemn(end_lst,'1') ;
  do cur_obs= start_row to end_row ;
     rc= insertn(slctd_rows_lst,
      cur_obs,-1) ;
  end ;
end ;
test_lst= DELLIST(test_lst) ;
```

The SCL list *test_lst,* obtained using the _Get_Selections_ method is actually a series of sublists, one for each selected range. The preceding section of code gets the named items Start_Row and End_Row for each sublist and inserts each selected observation to the SCL list *slctd_rows_lst*.

NON-VISUAL OBJECT CLASSES

As previously noted, the summarization and analysis for the Risk Strategy Instrument Panel takes place on the host mainframe. One of the challenges in the design of the application was how to implement the communication of information between host and the client interface.

Part of the answer seemed obvious: employ SAS/CONNECT, using TCP/IP as the communication access method. The not so obvious coding solution was to create was to create a non-visual class to handle the myriad of situations where a list of choices or a single piece of information had to be obtained from the host.

This programming scheme was selected for two reasons. First, it seems desirable to avoid adding a number of SCL entries to the application's catalog to make the program easier to maintain. Also, creating a subclass could make easier to use inheritance and other object-oriented techniques later on.

To create a non-visual subclass, the following steps should be performed:

1. edit a new class entry in the target catalog to bring up the class editor
2. give the class a description
3. for a non-visual class, make sure the parent class is sashelp.fsp.object.class or a subclass of this class
4. save the subclass at this point

After the subclass has been created, it is time to begin adding methods. To add a method, do the following:

1. from the class editor, select methods, and the methods window will be displayed
2. select the Actions push button
3. select Add mode on
4. then enter the method's name and label
5. enter the source entry in which the SCL methods will reside
6. click on the New button to add the method to the subclass

What methods might be required for an application such as the Risk Strategy Instrument Panel? The remainder of the paper will discuss some of the methods that we implemented in our main non-visual class, the GET_ROWS Class.

GETTING THE TSO USER ID

In the Risk Strategy Instrument Panel, it seemed pointless to ask users to type in their TSO IDs since this information is known by the host mainframe when they log in. Hence, one of the first methods added was GETTSOID to obtain this value. This method also illustrates the use of %SYSRPUT macro call to move a macro value from a remote host. The method was programmed as follows:

```
gettsoid:
method userid $ 7 ;

/* test to see if already signed on */
/* sign-on to MVS if not */
/* Mynode is the name of the
   mainframe host */
isrlinkd= rlink('mynode') ;
if isrlinkd eq 0 then do ;

   /* pop-up warning about MVS sign-on
      wait (note that Popmsg.Frame
      is not supplied by SAS and has
      to be created) */
   wndtxt= 'Signing on to MVS Host' ;
   msgtxt= 'please wait for User ID?
    prompt' ;
   call display
    ('work.catname.popmsg.frame',
      wndtxt, msgtxt) ;

   control asis ;
   submit continue ;
   signon ;
   endsubmit ;

   /* test to see if signon was
      successful */
   isrlinkd= rlink('mynode') ;
   if isrlinkd eq 0 then do ;
      wndtxt= 'SIGNON Failed' ;
      msgtxt=
       'check your user id and
        password' ;
      call display
      ('work.catname.popmsg.frame',
        wndtxt, msgtxt) ;
      return ;
   end ;
end ;

submit continue ;
rsubmit ;
%sysrput userid= &sysuid ;

endrsubmit ;
endsubmit ;

/* pop up warning if null string
returned */
userid= symget('userid') ;
IF userid eq '' then do ;
   wndtxt=
    'Failed to Obtain TSO User ID' ;
   msgtxt=
    'logoff TSO, then restart
     Instrument Panel' ;
   call display
    ('work.catname.popmsg.frame',
    wndtxt, msgtxt) ;
end ;
endmethod ;
```

As the GETTSOID method illustrates, method block begin with a Method statement and are closed with an Endmethod statement. On the method statement, parameters passed to the method may be included, using the same syntax as when parameters are passed using a Call Goto or a Call Display.

To use the method, we must load the class and invoke the method. This is very simple to do as the following code illustrates:

```
init:
/* load Get Rows class */
dataobjid= instance(loadclass(
 'libref.catname.get_rows.class')) ;
return ;

main:
/* get TSO user id */
call send(dataobjid, 'GET_TSO_USER_ID',
 userid) ;
return ;
```

In this example, *dataobjid* is the numeric variable that assumes the ID number for an instance of the GET_ROWS class. This instance only needs to be loaded once during execution of the calling Frame so the code is executed in the INIT section.

The method is executed by the Call Send statement, which executes the SCL code in the method block and returns the TSO user ID in the SCL variable, *userid.* As this example demonstrates, non-visual classes and methods are an excellent way to store and execute a library of frequently used routines in SAS/AF catalog entries.

DOES A TAPE DATA SET EXIST?

SAS/AF applications developers often use the FILEEXIST function to determine if a file already exists on a disk drive. However, under MVS and TSO, if the file is on tape or has been archived,

FILEEXIST will indicate that the file does not exist, even when it really does.

However, by examining the output of the TSO LISTCAT command, an accurate determination of a data set's existence can be made. One of the methods created for the Risk Strategy Instrument Panel was TAPEXST, which reports true when a data set exists but is stored on tape or is archived.

```
tapeexst:
method prefix dsn_nm $ 44
       dsn_exst $ 1 ;

/* test to see if already signed on */
/* sign-on to MVS if not */
isrlinkd= rlink('mynode') ;
if isrlinkd eq 0 then do ;

   control asis ;
   submit continue ;
   signon ;
   endsubmit ;

   /* test to see if signon was
      successful */
   isrlinkd= rlink('mynode') ;
   if isrlinkd eq 0 then do ;
      wndtxt= 'SIGNON Failed' ;
      msgtxt= 'check your user id
       and password' ;
      call display
       ('work.catname.popmsg.frame',
         wndtxt, msgtxt) ;
      return ;
   end ;
end ;

/* qualifier for LISTCAT command */
call symput('prefix',prefix) ;
/* get DSN to test */
call symput('dsn_nm',dsn_nm) ;

submit continue ;
rsubmit ;

/* get list of DSN(s) via IDCAMS */
/* allocate fileref to contain IDCAMS
output */
%macro temp1 ;

%let tdsn_nm= &sysuid..lcatout ;
dsnexst &tdsn_nm ;

%if &sysdexst eq 1 %then %do ;
filename
 lcatout
 "&sysuid..lcatout"
 space=(625,(25,10))
 lrecl=125
 blksize=625
 recfm=v
 unit=sysda
 disp=(old,delete)
 ;
%end ;

%else %do ;
filename
 lcatout
 "&sysuid..lcatout"
 space=(625,(25,10))
 lrecl=125
 blksize=625
 recfm=v
 unit=sysda
 disp=(new,delete)
 ;
%end ;
%mend ;

%temp1 ;

tso listcat l(&prefix) outfile(lcatout)
;

/* allocate data set to contain DSNs */
%macro temp2 ;
%let tdsn_nm= &sysuid..dsnlist; ;
dsnexst &tdsn_nm ;

%if &sysdexst eq 1 %then %do ;
libname dsnlist "&sysuid..DSNLIST"
 disp= (old,delete)
 space= (trk,(1,1),rlse)
 unit=sysda
 ;
%end ;

%else %do ;
libname dsnlist
 "&sysuid..dsnlist"
 disp= (new,delete)
 space= (trk,(1,1),rlse)
 unit=sysda
 ;
%end ;
%mend ;

%temp2 ;

/* filter records to include only
   DSNs */
data dsnlist.dsnlist ;
   length dsn $ 44
    ;
   infile lcatout lrecl=80 missover pad
   ;
   input @18 dsn $char44. ;
   if substr(dsn,1,7) eq
    '<high-level qualiifier>'
    then output ;
```

```
run ;

proc download data= dsnlist.dsnlist
 out= work.dsnlist status=no ;
run ;

filename lcatout clear ;
libname dsnlist clear ;

endrsubmit ;
endsubmit ;

/* see if DSN exists */
dsn_exst= '0' ;
ds_id= open('work.dsnlist') ;
namenum= varnum(ds_id,'dsn');
call set(ds_id) ;
nobs= attrn(ds_id,'nobs') ;
do i= 1 to nobs ;
   rc= fetch(ds_id) ;
   if rc= -1 then i= nobs ;
   else do;
      dsn=  getvarc(ds_id,namenum) ;
      if dsn= dsn_nm then dsn_exst= '1'
      ;
   end ;
end ;
ds_id= close(ds_id) ;

endmethod ;
```

CREATING LOOK-UP SCL LISTS

One recurring set of needs identified while creating the Risk Strategy Instrument Panel was the need to create SCL lists from SAS data sets to serve as look-up tables. The SCL lists would then be used to populate list boxes in Frames that would display available choices.

As an example, one of the methods created to fill this need was LKUP_N2C, which would take a numeric variable as the look-up argument (SCL list item name) and use a character variable as the returned look-up value. The LKUP_N2C method follows:

```
/* LKUP_N2C - make SCL look-up list,
              numeric to character
   ds_nm    - two-level name of look-up
              dataset
   var_nm   - name of the variable to
              be looked up
   key_nm   - name of the variable to
              be used as a look-up key
   lkup_lst - SCL look-up list       */

lkup_n2c:

method ds_nm $17 var_nm key_nm $32
       lkup_lst 8 ;

ds_id= open(ds_nm) ;
namenum= varnum(ds_id,var_nm);
keynum=  varnum(ds_id,key_nm) ;
call set(ds_id) ;
nobs= attrn(ds_id,'nobs') ;
do i= 1 to nobs ;
   rc= fetch(ds_id) ;
   if rc= -1 then i= nobs ;
   else do;
      buff_c=  getvarc(ds_id,namenum) ;
      keyname=
       put(getvarn(ds_id,keynum),8.) ;
      rc= insertc(lkup_lst, buff_c,
                  -1,keyname) ;
   end ;
end ;
ds_id= close(ds_id) ;
```

CONCLUSION

The design of the Risk Strategy Instrument Panel provides a real-life example of how client-server architecture can make MVS batch applications more appealing and easier to user. The use of object-oriented programming techniques, illustrated in the sample code, is a major reason for the success of this application in meeting its design goals.

BIBLIOGRAPHY

SAS Institute Inc. (1993), *Getting Started with the FRAME Entry: Developing Object-oriented Applications, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997), *SAS® Macro Language Reference, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1994), *SAS® Screen Control Language, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGEMENTS

CONTACT INFORMATION

The authors may be contacted as follows:

Michael L. Davis
Bassett Consulting Services, Inc.
10 Pleasant Drive
North Haven CT 06473-3712
Internet: michael@bassettconsulting.com
Web: http://www.bassettconsulting.com
Telephone: (203) 562-0640
Facsimile: (203) 498-1414

Adam Terr
Retailer Financial Services
General Electric Capital Corporation
1600 Summer Street
Stamford CT 06927
Internet: Adam.Terr@gecapital.com
Telephone: (203) 961-2052
Facsimile: (203) 357-4281

Modified code is not supported by the authors.