SAS/AF® Features: Easy for You or Easy for Users?
Heidi Markovitz, Simply Systems, Key Biscayne, FL

Abstract:
SAS/AF includes many features that automate for developers standard data entry processing functions, such as field validation and selection lists. These functions can also be programmed in SCL using more conventional procedural techniques. Using concrete examples, this paper examines the relative merits of each approach in the areas of rapidity of development, speed of end-user training, and ease of use for both developers and end-users.

Introduction:
SAS/AF is a tool used to develop interactive applications. It provides facilities tailored to the functions of processing low-volume data as it is entered, and displaying data on demand. Unlike SAS/FSf®©, SAS/AF allows developers to make their screens behave any way they want. For instance, there are no limits on the number of data sets that can be displayed at one time. Flat files can be processed. Any number of observations can be processed as a single transaction. The programmer is the boss.

To perform these miracles SAS/AF has 3 generic techniques.

1- Screen description methods are non-procedural constructs such as Display drawing and Attributes (field-by-field definitions).

2- Screen Control Language (SCL) includes procedural commands that store and retrieve filed data, and that display and process data on the screen. Procedural commands are executed in a sequence determined by the developer. For instance, if a programmer wants to display a particular data set observation, she must know enough to read it first. Many SCL commands have non-procedural aspects. They are invoked in the order the developer specifies, but they launch complex subroutines over which the programmer has little control regarding sequence.

3- SCL interface with Base SAS may be considered a subset of SCL. It is used to process any SAS Data Steps or Procedures from within on-line programs. I use it to submit report programs and produce charts from SAS/QC®.

This paper compares the use of non-procedural methods with programmer-ordered techniques to reach the same goal. I present two examples that will bring out some of the issues involved in deciding the approach to use.

Application
The examples reviewed are inspired by an SPC (Statistical Process Control) system developed for Cordis Corporation, a manufacturer of state of the art diagnostic and interventional medical devices. The system includes SAS/AF programs for entering and reviewing the results of quality control inspections and for requesting custom reports and charts. It also has Base SAS and SAS/QC® programs to produce requested outputs. This SPC application was built using release 6.07 on a SUN Microsystems server running Unix. Users and developers display their work on dumb terminals, X terminals, Sun workstations, and PC's running terminal emulations. The examples presented here were specially produced for this paper. They are combined from actual production programs and older test versions.

Example 1 - Field Validation, Date
The first example involves validation of an entered date. Figure 1 shows a screen used to produce customized P-Charts (SAS/QC) and reports. The operator can request from it a chart or report for any range of dates. She can specify that the output include a certain part number, a particular product, or one phase of testing. The output device is also selectable - the subject of our next example. Here we concentrate on the Test Date range fields. Figure 2 shows the Display window where the layout of the Request Screen was entered. The fields CFROMDY and CTOMDY are the ones where the operator enters her dates. This first version of the screen uses automatic validation features on the Attribute window. Figure 3 shows the Attribute window providing the automatic validation for CFROMDY. Note that the Data Type is NUMERIC and the INFORMAT and FORMAT are both standard MMDDYY8. (like 03/15/93).

When the operator fills in one of the date fields SAS/AF automatically inputs the screen value into the program variable using the specified format. If the user entered a To Date "2/15/94", SAS/AF would translate that into SAS date value 12464, set CTOMDY to 12464, and proceed to the MAIN section of the program. If the user enters a non-date value (e.g. "12/35/93") the system highlights the field, possibly displays a generic error message (e.g. "Value entered does not agree with format"), and waits for a correction. It may or may not execute the MAIN section, depending on other program elements. So far you have avoided formatting the date yourself, and you perform cross-validations in the MAIN section, checking that the To Date is not earlier than the From Date (see code fragment in Figure 4).

```
MAIN:                   more code                 Restart chart or report with from-date;
                   more code
***comdy - Cross-validate To-Date for **
                   chart or report with From-Date;
                   more code
IF cromdy < cfomdy THEN
  DO; _MSG_ = 'End date is earlier than
               start date. Re-enter'
               | or leave blank.';
  ERRORON cfomdy cromdy;
  CURSOR cfomdy;
  ANYERROR = 'YES';
  END;
                   more code
RETURN: /***** end MAIN section ******
```

Figure 4 SCL Code to Cross-Validate To Date

Now you want to make this screen easier to use. Since your users are mostly running reports of the latest tests, you make the System Date the default ending date. This means that when the operator leaves the To Date field (CTOMDY) blank, your MAIN section inserts the System Date into it. This change reduces your user's normal data entry load by one field.

One day, your user wants to run a report for last December. She enters end date "12/93" - not a bona fide date. The system processes the value, see's it's not a date, highlights the erroneous field, and sets CTOMDY to . (null). Maybe it displays a generic invalid format message. If a CONTROL ERROR statement is coded into the program, the MAIN section is executed. Since CTOMDY is blank (null), the system inserts the
Enter subject to be analyzed -
Product Line: __________
OR Part Number: __________

All sizes for related parts? Y/N
Test Code: __________
Location: __________ ALL Locations

Number of printer or 'V' to view on screen: VIEW View Display
Produce report(R), Chart(C), or Both(B): R Report Only

Range of test dates to be included. From: __________
To: __________
To change chart title, type over text below:
Attribute P-Chart

Control Limits for Chart (Omit to allow automatic calculation)
Upper Limit: AUTO Mean: AUTO Lower Limit: AUTO

---

Field name: CFROMDY Frame: 1 Row: 14 Col: 49 Length: 8
Alias: CFROMDY Choice group: Pad:
Type: NUM Protect: YES NO INITIAL
Format: MMDDYYYY
Informat: MMDDYYYY.
Error color: RED attr: REVERSE Help: DATES HELP
List: __________
Initial: __________
Replace: __________
Options: CAPS CURSOR REQUIRED AUTOSKIP NOPROMPT NON-DISPLAY

---

Field name: CFROMDY Frame: 1 Row: 14 Col: 49 Length: 8
Alias: CFROMDY Choice group: Pad:
Type: CHAR Protect: YES NO INITIAL
Informat: __________
Error color: RED-- attr: REVERSE Help: DATES HELP
List: __________
Initial: __________
Replace: __________
Options: CAPS CURSOR REQUIRED AUTOSKIP NOPROMPT NON-DISPLAY
current date into the field, as prescribed in Figure 5. Now your user sees his To Date field highlighted in error, possibly an error message displayed, and the field itself containing an unfamiliar value. If CONTROL ERROR is not there, the displayed value will not be reset, but the message will be just as cryptic, the user will not be able to END out of the screen, and the validation of other fields will not be performed. If you later change the CONTROL status (ERROR, ENTER) of the program, the screen behavior will change radically. Besides, Attributes are not entirely stable. For instance, when you change the screen layout (e.g. add a new field) Attributes may be erased without notice, causing a very hard-to-find bug. Splitting the validation between the Attribute window and the Source module makes debugging and troubleshooting very complicated.

```sas
Attr: cfronian - Cross-validate To Date with From Date. If blank default to today;
IF cfronian = blank THEN
DO; cfronian = PUT(TODAY(), mmddyy8.);
* MSG = 'End date was not spe'
END;
error; Output will
END;
IF cfronian < c dönemin THEN
DO: _MSG_ = 'End date is earlier than'
| 'start date. Please'
| 're-enter or leave blank.';
ERRORON cfronian c dönemin;
CURSOR cfronian c döneminde;
ANYERROR = 'YES';
END;

... more code ...

Figure 5 SCL Code to Cross-Validate and Default To Date
```

For these reasons, when I enhanced this program by adding the default mentioned, I defined the dates as character fields and performed all the validation in my Source program. The Display window needed no changes. Figure 6 shows the new field Attributes for CFROMDY. Figure 7 is the expanded validation Source code. This approach allowed me to differentiate between genuine blanks and invalidly entered dates. The end date field is only highlighted for errors after the SCL code has evaluated it. The error message is always appropriate to the circumstance. This validation is not affected by the presence of a CONTROL ERROR statement in the program. When I used the Attribute validation described earlier, I could not access the "bad" value entered from the program. That made it impossible to permit a valid non-date entry. For instance, later we wanted "MONTH" to be a special value the operator could enter to report on the current month without entering the specific dates.

With the dates defined as character fields on the screen, it was easy in SCL to set the date range to 3/1/94 - 3/19/94 when the user entered "MONTH" in either date field on March 19.

As this example shows, validation through the Attribute window is initially easier to implement. It is excellent for getting a new screen to work. As an application develops though, procedural validation in the Source becomes more convenient. It allows the programmer to permit character or numeric data in any field and control the order of validation. She can more easily allow the user to exit on demand, set up default schemes that reflect other data entered, and localize possible errors. Further, she can eliminate the need to work around the frequently puzzling SAS/AF Attribute processing sequence.

Example 2 - Selection Lists

Selection Lists are those handy "pop-up" displays that show what the choices are for filling in some field. We will study the same P-Chart Request program as in the first example (see Figure 1). The latest edition makes available to users Selection Lists for product line, part number, test code, test location, and output device. This example examines the output device Selection List, showing three different approaches.

1. Attribute Window List.
2. SCL List commands, such as DATA LIST and DATA LISTC.

Attribute Window Lists are the easiest to implement as they require no SCL coding. They are specified entirely on the field Attribute window. An early version of the P-Chart Request screen provided only one Selection List. It allowed the operator to display her output on the screen or route it to one of a choice of printers. On Figure 2, the Display window for the program, variable VOR is the one the operator can prompt for the printer Selection List was implemented. It was big enough to contain only an output device system ID, not a description. This program was originally intended as a quick front-end for two users who shared a printer and an office. It would extract data from a non-SAS data base and offer three output choices for charts and reports. Figure 8 shows the Attribute window where the Selection List was implemented. VOR (alias VORP, for view-or-print) has the CAPS Attribute, so that when lower case letters are entered the system converts them to upper case. The List Attribute shows three values VIEW, 6 for the MIS Printer, and 27 for the users' printer. Since our users typically review several charts daily but print them only when needed for meetings, the screen is displayed with an initial value of VIEW in the printer field. If the user wants to see the Selection List of all valid choices she enters 7 in the printer field (example in Figure 9). She selects her preference by moving the cursor to the correct line and pressing ENTER (or clicking with her mouse). She leaves without making a selection, by pressing the END key. When the Selection List screen disappears, the Request Screen features the chosen value. If the VOR field was left blank, the system inserts the initial value, VIEW. Since the SCL program does not contain a CONTROL ERROR statement, the System validates the entered value against the Attribute List. If the user types in an invalid value (i.e. not in the list) or no value, the system highlights the field, shows a generic message, such as 'ERROR: XXX is not in list of valid values. Please reenter', and skips the MAIN section of the SCL program. Since the REQUIRED Attribute was selected on Figure 8, the operator cannot END out of the screen until she enters a non-blank, valid value in the field. The List Attribute technique provided a Selection List and validation for our users without our writing a single line of SCL code.

```
Select Data

Make one selection

| Make one selection | VIEW | 6 |

<FINI) <OK) <CANCEL) <HELP>

Figure 9 Attribute List Selection List
```

Once the program was working, I discovered that our users actually moved between 2 different printers and had a boss who...
wanted charts printed at her desk. At the same time, other
departments (with other printers) wanted to test drive the
system. To avoid confusion, we needed a description next to
each printer number. I considered expanding the Attribute List
values with descriptions. For instance, instead of 6 the list
would include 6 - MIS Printer. This simple approach required no
SCL changes for the Selection List and no compilation.
However, while 6 was a system printer ID, 6 - MIS Printer was
not. To access the selected printer, I would have to include SCL
code to convert the displayed value to the actual printer ID. I
also looked at constructing a SAS Format, linking each printer ID
with its description. That would require writing and running a
separate Base SAS Proc Format and accessing it from either my
SCL program or the Attribute window. In either case, a change
to a system printer would require changing both the Proc Format
and the List on the Attribute window, recreating the format, and
possibly recompiling the SCL program. That was too many
tings to remember for something that would surely need doing
every few months. To avoid spending the coming months
keeping up with printers, I progressed to SCL List commands.

SCL provides a family of List commands that are single
statement tools for displaying a variety of Selection Lists.
Selection Lists of variable names (VARLIST), catalog entries
(KATLIST), and values from a data set (DATALIST, DATALISTC)
are among those available. List commands are non-procedural
SCL commands, since they invoke sub-routines which the
developer cannot control. For the printer Selection List I used
DATALISTC, displaying the printer ID’s and names from a SAS
data set created for this purpose.

The company had, at the time, over 30 printers attached to the
Unix network. They were all defined in a Unix system file called
printcap. Every time a printer was hooked up to the network or
changed (every couple of months, generally), the system
administrator had to change the printcap file. This then was the
most reliable source of up-to-date printer info. I wrote a batch
SAS program to create a SAS data set prtnames that would
include an observation for each printer (NOTE: This was the
weak spot in the arrangement, since the SAS data set would not
be automatically updated when the printcap file was. However,
the batch program could be set up to run automatically once a
week or it could be run manually when any user complained).

With the data set in place, I returned to the Request program
and inserted an SCL CONTROL ERROR statement into the INIT
section of the program. The Display was upgraded to include
separate fields for the printer ID (VOR) and name (PRNTNAM),
as shown in Figure 1. This time, I selected the NOPROMPT
Attribute for VOR (alias VORP) and emptied the List Attribute.
NOPROMPT inhibits system prompting (as with a List Attribute)
and passes the prompt character (1) to the SCL program.
PRNTNAM is defined as a display (protected) field with no
case conversion. The SCL source is where the action is. Figure 10
shows the code fragment that handles the printer selection. The
prtnames data set will have been opened earlier in the session
with a file ID PRNTFID. When the user prompts for the Selection
List the DATALISTC command opens a window with the title
"Select view ([display] or printer", followed by a list of printer
ID’s and names from the prtnames data set (here identified by
PRNTFID). The operator can scroll through the list if it’s a long
one, as in Figure 11. She can select a value by pointing with
the cursor and pressing ENTER, or leave without making a
selection by pressing the END key. When the Selection List
window closes, the printer ID from the selected line is inserted
into variable VORP and the name is inserted into PRNTNAM.
VORP is validated against the same prtnames data set, in case
the user typed in a value instead of selecting from the list.

The DATALISTC approach had many advantages over the

<table>
<thead>
<tr>
<th>Select Data</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Select view (display) or printer</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7H Product Develo</td>
</tr>
<tr>
<td>3</td>
<td>7H Documentation</td>
</tr>
<tr>
<td>4</td>
<td>8B3 Quality Assur</td>
</tr>
<tr>
<td>5</td>
<td>7A Tool Design</td>
</tr>
<tr>
<td>6</td>
<td>7-K M.I.S</td>
</tr>
<tr>
<td>7</td>
<td>8B Production</td>
</tr>
<tr>
<td>8</td>
<td>Sun Sparcprinter</td>
</tr>
<tr>
<td>9</td>
<td>7H Documentation</td>
</tr>
<tr>
<td>11</td>
<td>7-J-E Common Area</td>
</tr>
<tr>
<td>12</td>
<td>8C Receiving Insp</td>
</tr>
<tr>
<td>13</td>
<td>COMMON AREA 7-L</td>
</tr>
<tr>
<td>14</td>
<td>7H Documentation</td>
</tr>
</tbody>
</table>

Figure 11 DATALISTC Selection List

Attribute List. SAS provides built-in Help and a Find Text
function for its List command screens. Any changes to the
prtnames data set were immediately effective in the program.
The list worked (mostly) regardless of the CONTROL (ENTER
ERROR) status of the program. Also, I had control over when
the list was displayed and validation performed. For instance,
if there was a validation error elsewhere on the screen, the list
processing could still be performed. By including a CONTROL
ENTER ERROR statement I could force the MAIN section to
always be executed, allowing the user to END out of the screen
even when it contains erroneous data. Please note that by
taking this control, a developer is prevented from using any
Attribute window validation (validation specified in the Attribute
window will be ignored during execution). If the developer does
want to use a combination of Attribute and SCL program
validation, she must omit the CONTROL ERROR statement. In
that case, the Attribute window validation is performed first. If
it flags an error, the SCL program validation is not performed
- a confusing situation for users (i.e., after they fix all the errors
flagged by the Attribute specifications, the program can
suddenly appear with another error on the same field).

On the negative side, the size of a List command window is
determined by factors other than the size of the data being
shown (note that Figure 11 appears to be truncated). The
programmer can’t specify field justification or other formatting.
The developer cannot provide field-specific Help. The window
cannot include subsets of observations, data combined from
multiple data sets, or data not in a data set at all. The rate of
scrolling (page, half page, n lines) cannot be controlled.

At this point, our users were very satisfied. The application
easily beat out its predecessor in functionality ("You want
charts? Take these SAS manuals."). Word spread, and soon
several bigger departments wanted an all-SAS SPC system,
including data entry, security, and reporting.

The full-blown system that evolved included a somewhat
"normalized" relational data base of SAS data sets. It
incorporated several combined "Browse' and Menu" screen
programs that allowed users to look through Extended Table lists
of recorded data and take action on them. For instance, a user
could scroll through the list of defined Part Numbers, determine
that the one she needs is not there, and go immediately to an
Add screen. Or if the part was on the list with the wrong name,
she could jump to the Update screen by pressing a function key.
The system also included the now-familiar P-Chart Request screen, made famous by Figures 1 and 2. This mature version provided Selection Lists for product lines, part numbers, test codes, locations, and of course output devices. The new Selection Lists were implemented using calls to the Browse and Menu programs instead of DATALIST(C) commands. That meant that when an operator entered 7 (prompt) in the Product Line field, for example, the Product Line Browse and Menu screen would be displayed, instead of an SCL-generated List window. This technique avoided some of the limitations enumerated earlier in this paper and provided several other serious advantages. Since the Browse and Menu screens are the gateways to the data entry and inquiry screens, all users learn how to use them. By recycling the same programs, even infrequent users of Selection Lists would be comfortable navigating them. Users could type in the values (or partial values) they were seeking instead of paging through long lists. They could exit the system from Selection List windows using the standard defined keystroke or Pmenu selection (such exit requests are always verified with the user). User-accessible fields could have relevant Help available. Single programs used for all instances of a particular list needed to be fairly complex, to take into account their different uses. However, such lists could combine fields from different data sets - a real boon with a database that used separate data sets for looking up long names. The items that appeared on the Selection List could be restricted based on the user's authorization. In addition, most Selection Lists were invoked from several programs in the system. If DATALIST(C) commands were used, the programmer would need to change the commands in all programs displaying the Selection List whenever a displayed field changed. With this technique, only the one Browse and Menu program would need to be changed.

The advantage of having to deal with only one Source program at a time for most data base changes was compelling. But the DATALISTC printer Selection List already had fans on the smaller system. Printer ID's did not need the sophistication of our Browse and Menu programs (e.g. they could not be updated on-line). However, we did finally elect to adopt a standard Browse and Menu program for output devices. We were convinced by the advantages inherent in interface consistency. We also realized that in this system a printer Selection List would have broader applicability. For instance, to effectively offer screen printing (with the SAS/AFSPrint command), we had to allow our users to select a printer from any screen. The new Printer Selection program was used for this purpose too. Figure 12 shows the Selection List presented by this independent program. Figure 13 shows the revisions made to the code in Figure 10 to invoke the Printer Selection program and process the value selected.

```
   Figure 13 SCL Code to Invoke Printer Selection Program

   ** vorp - Select printer or screen for output. If user enters 7 program calls printer selection program. Selected or entered value is checked against the same line used by the selection program:

   IF lookup(?,vorp) ^= 0 THEN DO; CALL DISPLAY('br020sp,program','1);
      %getmsg; **If any message returned, display;
      vorp = SYMGET('mvorp')!
      IF SYMGET('metatus') = 'C'
         THEN RETURN;
      END;
      IF SUBSTR(vorp,1,1) = 'V'
         ........ more code .......
```

** Figure 13 SCL Code to Invoke Printer Selection Program **

Conclusion

There are many ways to perform data processing tasks in SAS/AF. In choosing a method to accomplish a particular task, remember to consider the speed of initial development, ease of maintenance, and ease of use. You and your users may have to live with your choices for a long time.

Acknowledgements

Thanks to Carlos Flores, who was the perfect audience and proofreader for many versions. Thanks also to Jorge Arango, Juan Diaz, and Cordis Corporation, for their cooperation which enabled me to complete this paper.

References


SAS, SAS/AF, SAS/FSP, and SAS/QC 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.

Contact:

Heidi Markowitz - Simply Systems
161 Crandon Blvd. Ste 325 - Key Biscayne, FL 33149
305-361-9772
**tomdy - Validate To Date for chart/report, from character screen variable.**

*If blank, default to today;*

```lisp
IF cfromdy = BLANK THEN
  DO; cfromdy = PUT(TODAY(), mmddyy8.);
  *MSG_ = 'End date was not specified. Output will end with latest date.';
END;
ELSE IF INPUT(cfromdy, mmddyy8.) = . THEN
  DO; _MSG_ = 'End date must be blank or a valid date in the form mm/dd/yy.';
  ERRORON cfromdy;
  CURSOR cfromdy;
  ANYERROR = 'YES';
END;
ELSE cfromdy = PUT(INPUT(cfromdy, mmddyy8.), mmddyy8.);
IF (NOT ERROR(cfromdy) AND NOT ERROR(ctomdy)
AND INPUT(cfromdy, mmddyy8.) < INPUT(ctomdy, mmddyy8.)) THEN
  DO; _MSG_ = 'End date is earlier than start date. Re-enter or leave blank.';
  ERRORON cfromdy ctomdy;
  CURSOR cfromdy;
  ANYERROR = 'YES';
END;
RETURN; /********* end of MAIN **********/
```

Figure 7 SCL Code to Fully Validate To Date

---

**VOR Frame: 1 Row: 10 Col: 51 Length: 4**

Field name: VOR

<table>
<thead>
<tr>
<th>Field</th>
<th>Alias</th>
<th>Type</th>
<th>Format</th>
<th>Informat</th>
<th>Error color</th>
<th>Attr</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOR</td>
<td>VORP</td>
<td>CHAR</td>
<td>----</td>
<td>---</td>
<td>RED</td>
<td>REVERSE</td>
<td>PRTNAMES HELP</td>
</tr>
</tbody>
</table>

Figure 8 Printer Field Attributes Using Attribute List

---

**vorp - Select printer or screen for output. If user enters ? and printer file already opened, program opens a Selection List window. Selected or entered value is checked against the same file used for the selection window;**

```lisp
IF lookup('?', vorp) ^ 0 THEN
  DO; IF prtfid GT 0 THEN
    vorp = DATALISTC(prtfid, 'prtid prtnam', 'Select view (display) or printer:','Y');
  ELSE _MSG_ = 'List of printers not available. Please report message to MIS';
END;
IF SUBSTR(vorp, 1, 1) = 'V' THEN vorp = 'VIEW';
ELSE IF vorp = BLANK | vorp = '0' THEN
  DO; vorp = 'VIEW';
  _MSG_ = 'No printer specified. Chart will be sent to VIEW';
END;
IF prtfid > 0 THEN /*printer file open*/
  DO;
    prtrc = LOCATEC(prtfid, VARNUM(prtfid,'prtid'), vorp);
    IF prtrc > 0 THEN prtnam = GETVARC(prtfid, VARNUM(prtfid,'prtnam'));
    ELSE DO;
      prtnam = ' '; ERRORON vorp;
      anyerror = 'YES';
      _MSG_ = 'Printer |vorp| is not valid. Please enter a valid printer ID';
    END;
  END; /*********end validate against printer file ****/
```

Figure 10 SCL Code for DATALISTC Printer List