Paper 69

First Steps in Building SAS® Applications

Bruce Bovill, SAS Institute, Marlow, UK

ABSTRACT

Using functionality from base SAS® and SAS/FSP® software you can build an application that is basic in its use of the power of SAS/AF® software, yet is still very useful.This paper takes a very basic approach to the creation of a simple SAS® application and assumes little or no prior knowledge of the primary tool used, SAS/AF software.

Not everyone is a professional applications developer and this paper serves as an introduction to anyone considering the benefits of creating custom applications in the light of their lack of time or budget for entering formal training.

INTRODUCTION

Well before the introduction of a SAS software implementation for the Microsoft Windows environment, the SAS user could use SAS/AF software to customize the way the end user approached using a particular application.

There has been an enormous growth in the market for PC-based office applications during the past ten years. The user now expects to see a very graphical ('GUI') look in everything they use, word processors, graphics packages, databases and statistical applications.

Since 1991, SAS/AF software has been developed to be increasingly graphical and extremely functional, and is the basis for many enhancements to SAS software itself, such as the Analyst Application, SAS/Warehouse Adminstrator™ and Enterprise Miner™ software.

The interfaces of these applications are attractive, easy-to-use and very functional. you can bet that an awful lot of time was spent by developers who know SAS/AFsoftware inside-out.

Is there anything useful the average user can implement easily with the aid of SAS/AF software, or is a very high level of knowledge and training necessary?

The answer is that the average user can very easily create an application that is immediately useful, with only a minimal knowledge of base SAS, SAS/FSP and SAS/AF software - and interface design.

BEFORE YOU START

You have to want to do something. . .

Software applications of any sort do not exist in isolation; they are built with a specific need in mind.

If there is more than one need, then all the better!

Some Facts of Life

The first thing you should realize is that you are not a professional applications developer. You don't have everything you need to know in your head, primed and ready to go! You do not have the luxury of time to learn products thoroughly before you start work.

You always learn best from things you see you have not done as well as you might have. You create things and from time to time, you un-do them or re-do them in the light of your growing understanding of what you can accomplish.

SOME BASICS

Using SAS/AF software to create a graphical user interface (GUI) means that you use object-oriented technology.

What is an object?

In simplest terms, an object is a building block. Figure 1 shows two examples of basic objects.

Figure 1. Simple Objects

On the left is a pushbutton object. When you press it then something you have programmed will happen.

On the right is a radio button object. You can select either 'TABLE' or 'FORM'. As you select the one, the other is automatically de-selected.

You could use the two together to browse a data set and to choose to look at the data as a table or one observation at a time.

Figure 2. More functional objects

This is a 'tab layout' object. Each tab can be programmed to provide a different function. You will use this object as a basis for the application.

Are objects difficult to use?

When one object starts to talk to other objects and when you build your own larger objects from several basic objects they become more complex. In basic use, objects can be really easy, especially if you use some of the ready-to-go function that is part of SAS software, and that you may already use. Screen Control Language (SCL, renamed SAS Component Language for Version 7) is the means to make objects work together.

For example, there are objects that enable you to enter and browse data as a table or by individual observation. You may already use PROC FSEDIT as the way you enter data and the good news is that you can continue to use this procedure within your application, together with any associated screen catalogs you have previously created.

Sample data

The data used for this paper has been reduced to its basics and relates to students attending courses and their performance.

The fields are:

1. Start of Academic Year
2. Student Name
3. Student ID
4. Course Name
5. Course Code
6. Course Modules Studied (up to 8 fields)
7. Course Modules results (up to 8 fields)
8. Course Modules marks (up to 8 fields)

Data already exists and was input using SAS/FSP software.

Scope of the application

You need to write an application that enables the user to:

1. Enter and edit data
2. Browse data
3. Produce reports to review the progress of students

BUILDING THE APPLICATION

The first task is to define where you want to store your new application and assign a LIBNAME:

```
LIBNAME sugi24 'f:/Bruces documents/SUGI 24';
```

Now create an empty FRAME window.

The initial window

You first need to start the SAS/AF BUILD window. The quickest way to do this is to type 'build' in the SAS Command Dialog, then press the check button to execute the command.

The BUILD window appears(Figure 4).

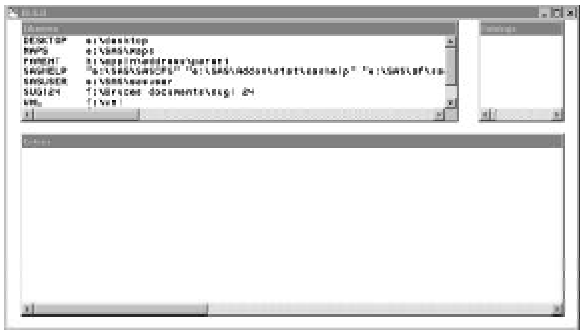

Figure 4: The BUILD window

- In LIBRARIES, press the 'SUGI24' library name to select where the application will be built.
- Right click within the ENTRIES window and select File->New->Catalog to create a new catalog. Call the new catalog 'SUGI24'.
- The new, empty catalog appears in the CATALOGS window. It should be highlighted - if it is not highlighted, click on it to select it.
- Now create a new entry in the catalog: right click within the ENTRIES window and select File->New->Entry
- A NEW ENTRY dialog opens. The default entry type is FRAME. A frame is the window where you assemble objects and build applications. This is your first frame, so call it 'start'.

The frame is empty and looks like Figure 5.

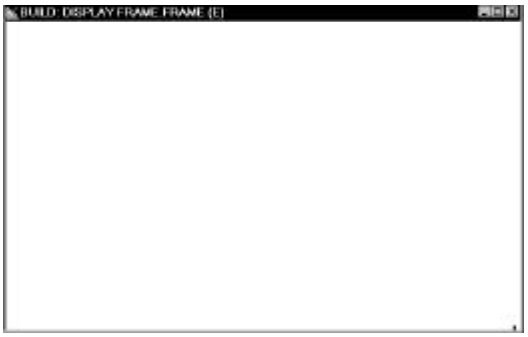

Figure 5: Empty FRAME.

Using the TAB LAYOUT OBJECT to avoid multiple frames

Many applications adopt a hierarchy of frames and it is easy to call a second frame from one you have already created. Multiple frames make for greater complexity when SCL variable values need to be carried across. The TAB LAYOUT OBJECT saves the need to carry values between frames and is an effective solution in many situations.

Using this object you can place different aspects of your application on different tabs within a single frame, as in Figure 6.

Figure 6: TAB LAYOUT object

Creating the TAB LAYOUT object

First you need to create the TAB LAYOUT OBJECT within the frame.

1) Right click and select MAKE. Find and select TAB LAYOUT OBJECT from the list that appears. Click within the frame to place the object. The TAB LAYOUT OBJECT ATTRIBUTES dialog opens (Figure 7)

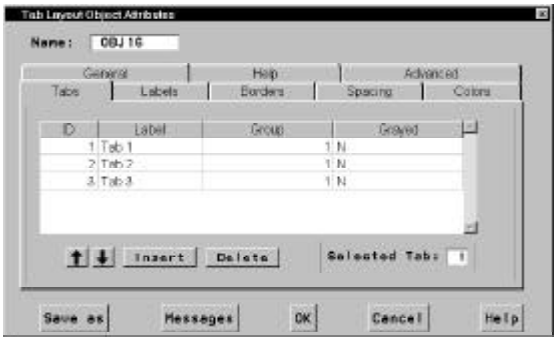

Figure 7: TAB LAYOUT OBJECT ATTRIBUTES dialog

2) You want to have 4 tabs in all. By default there are just 3. Press the INSERT button to add the extra tab. It will appear in the dialog window.
3) To name the first tab, left click on the LABEL field 'Tab1' and overtype with 'Edit/View'.
4) In turn, select the other tabs and overtype them as follows:

   tab2 Pre-built

   tab3 Assisted Ad-hoc

   tab4 Cleanup/End

5) Press OK

The TAB LAYOUT OBJECT appears and is now ready for you to use. (Figure 27)

ADDING DATA ENTRY FUNCTION

Using PROC FSEDIT and a custom data window

To use PROC FSEDIT the SAS syntax is:

```
PROC FSEDIT DATA=sugi24.students;
RUN;
```

By default PROC FSEDIT decides how to present your data. The presentation of your data may look like Figure 8.

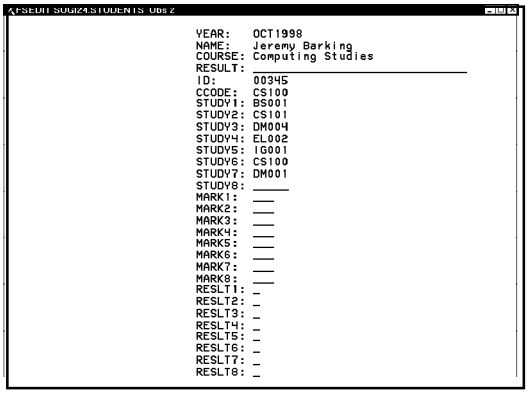

Figure 8: Default Data Entry Screen.

You may prefer to modify the data window, for ease of understanding and use.

When you use PROC FSEDIT together with a custom data window you specify a SCREEN= parameter:

```
PROC FSEDIT DATA=sugi24.students
            SCREEN=sugi24.students;
RUN;
```

A SCREEN catalog is created that contains the changes you have made to the way your data appears on screen.

Using PROC FSEDIT within a frame window

The first object to place in the EDIT/VIEW tab is one to enable you to enter and edit data for the sugi.students data set.

To create an object, right click inside the tab and a drop down list appears. (Figure 9)

Figure 9: Adding an object to the tab

Select MAKE then select IMAGE ICON from the MAKE selection list and press OK.

Figure 10: MAKE selection list

A dashed rectangle will appear in the tab and as you move the mouse, so the rectangle moves. Move the rectangle where you want it - you can move it later if you wish - and left click to drop it there.

The IMAGE ICON ATTRIBUTES dialog opens:

Figure 11: Image Icon Attributes dialog

This dialog enables you to decide how the object will look and what it does when selected.

1) Change the NAME to 'Edit'. This enables you to reference the object by name in the associated SCL code.
2) change the SIZE to 96 x 96.
3) In the LABEL field delete the default characters.
4) The image that appears on the object is shown in this dialog. To change it, press the existing IMAGE and an IMAGE SELECTOR list appears. Scroll to the 'Edit' entry and click to select it. Then press OK. The IMAGE ICON ATTRIBUTES dialog reappears with the new icon displayed.

Figure 12: Image Icon Attributes dialog with changes

3) When the application is executing and you select the IMAGE ICON, the data set needs to be opened in edit mode and with the custom data entry window displayed. This is handled by appropriate SCL statements.

INTRODUCING SCL

The object created above at the moment can do nothing. SCL is the programming language that tells it what to do.

Within the BUILD window (Figure 4) you use two kinds of entry:

1) frame entry that holds the objects
2) SCL entry that contains the SCL statements. You create SCL using the SCL window.

The SCL window

The SCL sits with its frame. To open the SCL Editor (Figure 13), select Locals->Edit SCL Source.

The SCL Editor works in a very similar fashion to the SAS Program Editor. Programming objects is different from normal DATA step programming because objects are event-driven: when you select one, that's when something needs to happen.

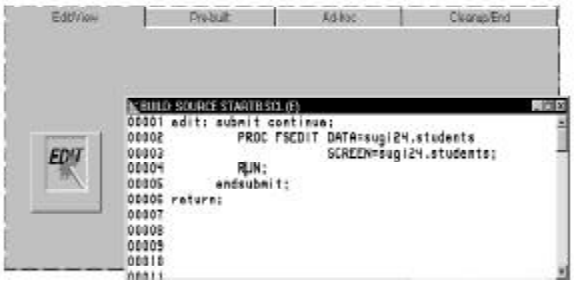

Figure 13: SCL Editor

All objects have an unique name. By default they are named OBJ1, OBJ2 and so on, but you may like to make them more easily identified within the SCL. For instance, an edit object called 'EDIT' makes the SCL that much clearer.

Figure 13 shows the small amount of SCL required to make this object function alone. This consists of a label, named for the EDIT object, the SAS procedure code for PROC FSEDIT within a SUBMIT CONTINUE/ ENDSUBMIT block (see below for a description), and finally a RETURN statement.

The application requires a number of objects to work together and this needs more detailed SCL. First you must create the other objects.

COPYING OBJECTS

Creating a browse IMAGE ICON

An IMAGE ICON that disables editing and entry of data is in most respects the same as the one just created. There are three differences:

1) The NAME of the object.
2) The image displayed on the object.
3) The object needs to use PROC FSBROWSE rather than PROC FSEDIT and SCL will reflect this.

The effective way to create the new IMAGE ICON is to COPY the existing one.

1) Place the mouse pointer over it and right click. From the resulting dropdown, select COPY. A dashed rectangle appears and you move it to where you want it then left click to position it.
2) Right click over the copied object and select OBJECT ATTRIBUTES.
3) Change NAME field to contain 'BROWSE'.
3) Click on the image and select the BROWSE item from the IMAGE SELECTOR list. Press OK to return, then OK again.

The tab now shows two different image icon objects.

Minimising numbers of objects

It is clumsy simply to keep on adding more and more objects without refining them. You could end up with an application looking like figure 14.

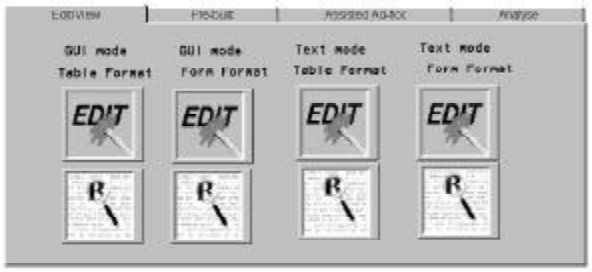

Figure 14: Needlessly Crowded tab

Rather than that, you can connect a variety of objects and enhance both the look and usability of the frame as in figure 15.

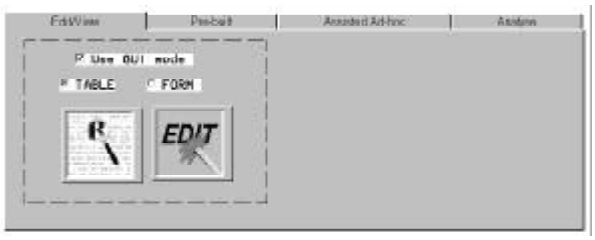

Figure 15: Enhanced Edit/Browse

The way this works is that there is a simple button for each of editing and browsing. You use the 'GUI' checkbox to select whether to have a GUI presentation of the data, or not. You select the TABLE or FORM radio button to choose a tabular or single observation view of your data.

The objects work together giving you all of the eight different options shown in figure 14.

All of this with a minimum of clutter in the window. But how much SCL does it take?

On the window you build, you will name the objects as in Figure 16.

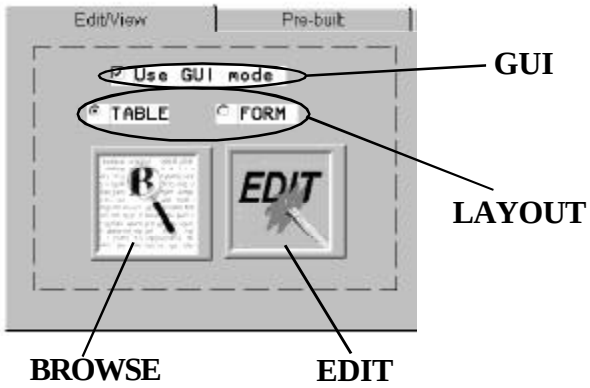

Figure 16: Enhanced Edit/Browse: Object Names

ADDING SOME NEW OBJECTS

Two of the objects in Figure 16 are new to you and you need to add them to the TAB LAYOUT OBJECT.

Adding a CHECK BOX object

1) Within the EDIT/VIEW tab, right click and select MAKE. Choose CHECK BOX from the selection list. Move the dashed object outline to where you want it and left click to place it there. A CHECK BOX ATTRIBUTES dialog opens.
2) Overtype the default NAME with 'GUI'.
3) Overtype the default LABEL with 'Use GUI mode'.
4) Change the LENGTH from the default to '12' (that's to accommodate the number of characters in 'Use GUI mode').
5) left click on the INITIALLY SELECTED check box. 'Use GUI mode' is now the default option for this object.
6) Click on VALUE ON SELECTION in the ATTRIBUTES. The resulting dialog shows a VALUE that is the same as the LABEL you typed above. Change it to 'GUI' Click on OK. If you check VALUE ON DESELECTION you will see it is empty, so the SCL can validate the contents of the check box by looking whether it is set to 'GUI' or not.
7) Press OK to close the CHECK BOX ATTRIBUTES dialog. The object is now ready to use.

Adding a RADIO BUTTON object

1) Within the frame, right click and select MAKE. Choose RADIO BUTTON from the selection list. Move the dashed object outline

to where you want it and left click to place it there. A RADIO BUTTON ATTRIBUTES dialog opens.

2) Overtype the default NAME with 'LAYOUT'.

3) Change LABEL LENGTH to 5, the number of characters in 'TABLE'.

4) In RADIO BOX LAYOUT, overtype the default NUMBER OF COLUMNS with '2'.

5) In RADIO BOX LAYOUT, overtype the default NUMBER OF ITEMS with '2'.

6) Click on ENTER VALUES in FILL TYPE.

7) In RADIO BOX ITEMS you will see two items, BUTTON1 and BUTTON2. The asterisk against BUTTON1 indicates that it is the default option.

8) Click on BUTTON1. 'BUTTON1' appears in ITEM. Change it to 'TABLE'.

9) Click on BUTTON2. 'BUTTON2' appears in ITEM. Change it to 'FORM'.

10) Press OK to return to the RADIO BUTTON ATTRIBUTES dialog.

11) PRESS OK again to return to the frame. The other object is now ready to use.

USING SCL TO MAKE THE OBJECTS WORK

Use of SCL in this paper is restricted to:

1) standard labels: INIT, MAIN and TERM: You can set values and actions to occur when an application starts up (using the INIT label), during the course of an application (using the MAIN label) and at the termination of the application (with the TERM label). Table 2 shows how to use the INIT label.

2) addressing objects. You assign each object an unique name and address an object using a label of the same name in the SCL, or using the object name as avariable.

3) SUBMIT blocks: You can use SUBMIT CONTINUE/ENDSUBMIT (Figure 13) to execute SAS language and procedures within SCL.

4) some SCL that is specific to particular objects:

- VIEWTABLE
- SAS/ASSIST® objects

Table 1 SCL code to open VIEWTABLE

```
*   dataset is an SCL variable that contains;

rc=insertc(arglist,       ,-1,'OPEN');
    editmode
*   if the data set is opened in EDIT or ;

rc=insertc(arglist,        ,-1,'OPENMODE');
    layout
*   if VIEWTABLE is used in TABLE or FORM ;

rc=insertc(arglist,      ,-1,'VIEW');
    call display invokes VIEWTABLE according;
                                          ;
          ;

display('sashelp.fsp.vtmain.frame',       );
```

Using VIEWTABLE

VIEWTABLE provides a GUI look and feel to data entry and browsing. It is neither SAS language nor a procedure and it requires SCL to invoke it.

The SCL in table 1 shows the code needed to open a SAS data set for use by VIEWTABLE. The values of the SCL variables used in table 1 are defined in the following ways:

dataset: you encode the name of the data set name into the DATASET variable when the frame window is started, using the INIT label. (Table 2)

Table 2: The INIT label

```
init: dataset='sugi24.students';
return;
```

editmode and mode: these values are set when you select the VIEW or EDIT objects. Each of these objects has an SCL label of the same name within the code. Table 3 shows the code for the VIEW object. The **LINK** statement causes execution to switch to the section of SCL labeled ***action.***

Table 3: Code for the VIEW object

```
    mode = 'BROWSEONLY';
    LINK action;
```

In table 4 there is no LINK statement and no RETURN statement. This is because the next section of SCL contains the **action** label and execution flows directly to that label as in table 5.

Table 4: Code for the EDIT object

```
edit: editmode = 'EDIT';
      mode = 'EDIT';
```

Table 5: Flow of execution from the EDIT label to the ACTION label

```
edit: editmode = 'EDIT';
      mode = 'EDIT';
action: IF gui = 'GUI' THEN DO;
        ...
        ...
```

The GUI and LAYOUT objects provide values to the SCL for the variables **editmode** and **mode**.

Elsewhere in the frame window, the GUI object is set by default to 'Use GUI mode' and this is indicated by the check in the checkbox.

The LAYOUT object is set by default to 'Table'. If you select 'Form' then that radio button will be highighted and the Table radio button will be de-selected.

The SCL labeled **action** uses the values of GUI and LAYOUT to decide how the data will look on the window, and the variables **editmode** and **mode** to decide whether to browse or edit the data.

The final segment of SCL code as shown in table 6 makes things happen.

Using the SUBMIT CONTINUE block

The SUBMIT CONTINUE/ENDSUBMIT block as used in Table 6 enables you to execute SAS DATA steps and procedures from within an SCL program.

Table 6: Editing and browsing data using the SCL code in the action label.

```
action: IF gui = 'GUI' THEN DO;
*     open the data set using VIEWTABLE  ;
*     if GUI is set to 'Use GUI mode'    ;
   arglist = makelist();
   rc=insertc(arglist,dataset,-1,'OPEN');
   rc=insertc(arglist,editmode,-1,'OPENMODE');
   rc=insertc(arglist,layout,-1,'VIEW');
   call display ('sashelp.fsp.vtmain.frame',
            arglist);
   rc = dellist(arglist);
END;
*     if GUI is not set to 'Use GUI mode' ;
ELSE DO;
   IF gui NE 'GUI' AND layout='FORM'
      THEN DO;
         submit continue;
            PROC FS&editmode DATA=&dataset
                        SCREEN=&dataset;
            RUN;
         endsubmit;
      END;
*     if LAYOUT is not set to 'Form' ;
      ELSE DO;
         submit continue;
             PROC FSVIEW DATA=&dataset &mode;
             RUN;
         endsubmit;
      END;
 END;
 return;
```

You can include SAS statements that will never change:

```
PROC FSEDIT DATA=sugi24.students;RUN;
```

Or you can embed SCL variables that are resolved at execution time:

```
PROC FS&editmode DATA=&dataset
          SCREEN=&dataset;
RUN;
```

In the example above, if the SCL variables are set such that editmode='EDIT' and dataset='sugi24.students', then the SAS procedure resolves to:

```
PROC FSEDIT DATA=sugi24.students
            SCREEN=sugi24.students;
RUN;
```

Each SUBMIT CONTINUE block must be terminated with an ENDSUBMIT statement.

The SCL that is shown in tables 2, 3, 4 and 6 comprises the entire code for the frame so far.

Compiling and testing objects that use SCL

Once the SCL is complete you can close the SCL Editor window, which returns you to the frame.

You have to compile SCL, along with the associated frame. Select Locals->Compile from the drop downs. (Figure 17)

Figure 17: Compiling SCL

Once the SCL has been successfully compiled, SAVE the frame before testing that it works. If you do not save the frame then the last saved version of the frame will be used. This will not be what you want.

Figure 18: Saving the frame

To test the application, type 'af c=sugi24.sugi24.start4.frame' in the SAS Command Dialog and run it.

Figure 19: Testing the frame

When you test the frame, the different results when pressing the VIEW button are like this:

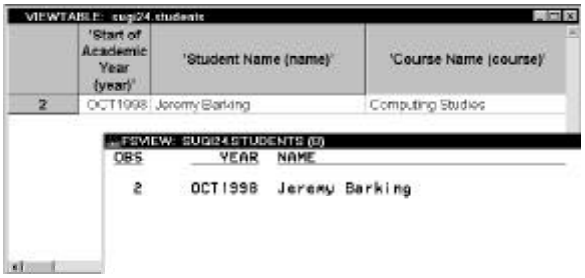

Figure 20: GUI and text format tables

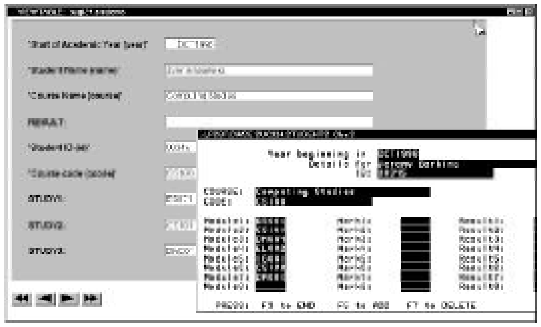

Figure 21: GUI and text format forms

REPORTING ON DATA

There are many kinds of report that you can produce from any data and for this application you will implement the following:

1) A pre-built report
2) An ad-hoc report using SAS/ASSIST software
3) Statistical reporting using the Analyst Application

Pre-built reports

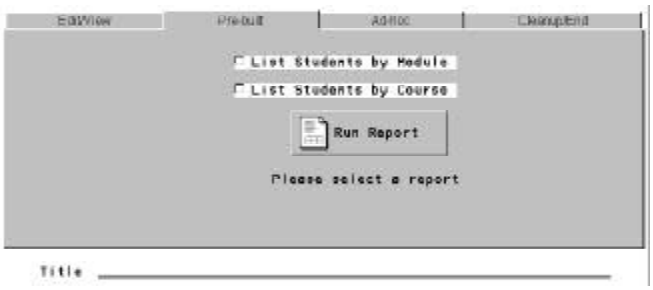

Figure 22: Objects on the PRE-BUILT tab and in the frame.

A pre-built report is one that always uses the identical SAS syntax each time it is selected, but that is run against data that regularly changes.

Pre-built reports are easy to incorporate:

1) Select the PRE-BUILT tab of the TAB LABEL OBJECT.
2) Right click in the tab, select MAKE and CHECK BOX. Place the object and the CHECK BOX ATTRIBUTES dialog appears.
3) Type 'REPT1' in the NAME field.
4) Type 'List Students by Module' in LABEL.
5) Replace the default LENGTH by a value of 24.
6) Press OK to return to the tab.

Repeat the process above for a second CHECK BOX object, but this time type 'REPT2' in NAME and 'List Students by Course' in LABEL.

These two CHECK BOX objects enable you to select a particular report. Now add a third object to run the report:

1) Still within the PRE-BUILT tab, right click, select MAKE and IMAGE ICON.
2) In the IMAGE ICON ATTRIBUTES dialog replace the default NAME with 'RUNREPT'.
3) Change the SIZE of the icon to 48 x 48.
4) Replace LABEL with 'Run Report'.
5) Select LABEL DISPLAYED TO THE RIGHT OF IMAGE.
6) Click on the image and select the REPORT icon from the IMAGE ENTRIES list. Press OK to return to the dialog and OK again to go to the tab. Re-size the object to show the full text by clicking and dragging the right side until the text all appears.

When this part of the application is visible, at first no report will be selected. If you then select RUN REPORT, nothing will happen and it will not be clear that this is because the application does not know which report to run. You want to be able to make it clear what is wrong and the best way is to display a message.

So you add a fourth object that will be purely informative.

1) In the PRE-BUILT tab, right click, select MAKE and INPUT FIELD LABEL.
2) position the object and in INPUT FIELD LABEL ATTRIBUTES, change the default NAME to 'Message'.
3) Delete all characters in LABEL.
4) Change JUSTIFICATION to CENTER.
5) Change LENGTH to 27.
6) PRESS OK.

Titles for Reports

Whatever reports you create, it is always helpful to be able to use titles, and to be able to change them easily. In this application, two tabs produce reports. It is more effective to place title objects in the frame rather than on each tab. (Figure 22)

To incorporate a title object into the frame:

1) Right click below the TAB LAYOUT OBJECT and select MAKE then TEXT ENTRY. Place the object in the frame and the TEXT ENTRY ATTRIBUTES dialog appears.
2) Overtype the default NAME with 'TITLE1'.
3) Overtype the default LENGTH with a value of 60 to enable titles of that size.
4) De-select CAPS in OPTIONS then select OK to return to the frame.
5) Right click in the frame and select MAKE and select INPUT FIELD LABEL. Place it in the frame and in the INPUT FIELD LABEL ATTRIBUTES dialog overtype the TEXT field with 'First Title'. Click OK and the object appears. Line it up with the TEXT ENTRY field. (Figure 22)

To incorporate the contents of TITLE1 into your reports, simply change the contents of the appropriate SUBMIT blocks. (Table 7)

SCL for running Pre-built reports

Table 7: incorporating titles

```
submit continue;
   TITLE &title1;RUN;
   PROC PRINT DATA=sugi24.students;
      VAR study1;
   RUN;
endsubmit;
```

The SCL you need ties together the five objects on the tab: REPT1, REPT2, RUNREPT, MESSAGE and TITLE1.

The presence of a message field requires you to ensure it displays appropriate messages all the time. This requires a change to the existing INIT block in the SCL to set an initial value of 'Please select a report' when the frame opens. (Table 8).

When you select one of the two LIST BOX objects, REPT1 and REPT2, you want the other to be de-selected. In a way these objects are

Table 8: setting an initial value for MESSAGE

```
init: dataset='sugi24.students';
      message='Please select a report';
return;
```

programmed - for this application - like a single object.
When the frame opens, neither of these objects is selected. As soon as

object is also set to be blank as a report is selected.
The final section of SCL (Table 10) submits a block of SAS code according

Table 9: Actions when check boxes REPT1 or REPT2 are selected.

List Students by Module

```
rept1: if rept1='List Students by Module' then
          rept2 = ' ';
       message = ' ';
return;
```

List Students by Course

```
rept2: if rept2='List Students by Course' then
          rept1 = ' ';
       message = ' ';
return;
```

selected before REPT1 or REPT2 have been selected, the MESSAGE object is set to 'Please select a report'.

Run Report

Table 10: Running the report style chosen.

```
runrept:
  if rept1 = 'List Students by Module' then do;
     submit continue;
        TITLE &title1;RUN;
        PROC PRINT DATA=sugi24.students;
           VAR study1;
        RUN;
     endsubmit;
    end;
  if rept2 = 'List Students by Course' then do;
     submit continue;
       TITLE &title1;
       PROC SORT DATA=sugi24.students
                 OUT=work.sorted;
          BY course;
       PROC PRINT DATA=work.sorted NOOBS;
          VAR name study1-study8;
          BY course;            RUN;
    endsubmit;
end;
if rept1=' ' and rept2=' ' then
     message='Please select a report';
else message = ' ';
return;
```

the TITLE1 object are included in a TITLE statement.
When you produce a report this tab looks like Figure 23.

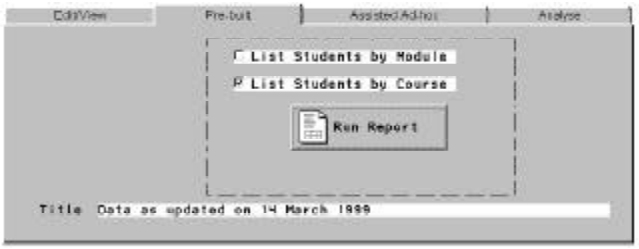

Figure 23: The PRE-BUILT report tab

The output is shown in Figure 24.

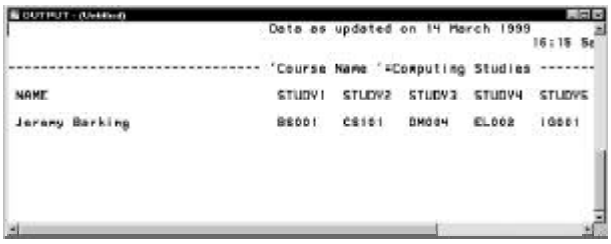

Figure 24: a PRE-BUILT report output

Ad-hoc Reports

To attempt to create an ad-hoc reporting application from scratch is beyond the scope of this presentation. Instead you use function that already exists in:

1) SAS/ASSIST software.
2) the SQL Query window in base SAS software.
3) in another SAS/AF application, the 'Analyst Application'.

The AD-HOC tab contains the objects shown in Figure 25.

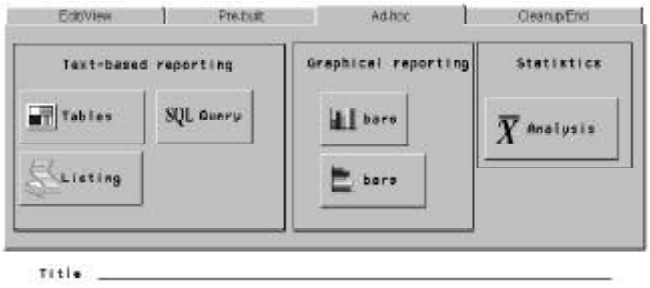

Figure 25: Ad-hoc tab

The objects on the tab are of two kinds:

1) Ones that use SAS commands (Query, Analysis)
2) Ones that use SAS/ASSIST function (Tables, Listing, bars)

SAS Commands: SAS Query Window

Objects that use SAS commands require no SCL as they are self-contained. To implement the Query object in Figure 50:

1) Right click on the tab and select IMAGE ICON from the MAKE list.
2) Type 'Query' in the LABEL field.
3) Select a suitable IMAGE.
4) Select LABEL DISPLAYED TO RIGHT OF IMAGE.
5) Select COMMAND PROCESSING in ADDITIONAL ATTRIBUTES.
6) Type 'query data=sugi24.students' in EXECUTE SAS STATEMENTS ON EXECUTION.
7) Press OK and then OK again to return to the tab.

Testing objects that use SAS commands

You can test that this object works by selecting Locals->Testaf from the drop downs. None of the objects that use SCL will work when you use testaf, but any objects that directly execute SAS statements will work.

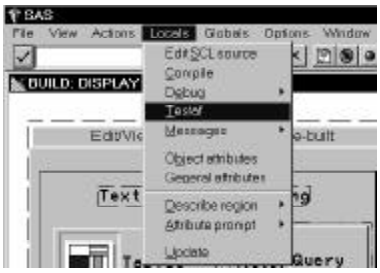

Figure 26: Invoking Testaf

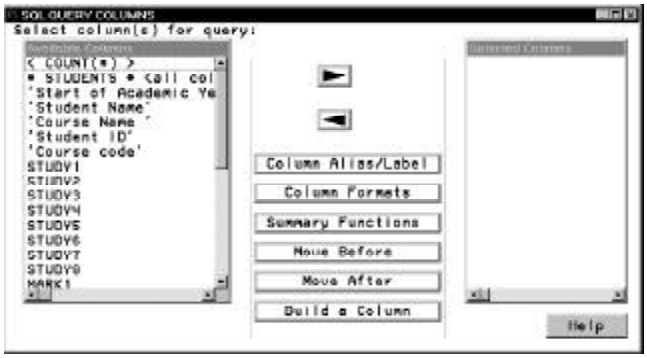

Figure 27 SQL Query Window

SAS Commands: SAS Analyst Application

The Analyst Application is a SAS/AF front-end to much of the statistical capabilities of SAS software. As such, it provides excellent ad-hoc statistical function you can use in this application

To implement the Analysis object in Figure 50:

1) Right click on the tab and select IMAGE ICON from the MAKE list.
2) Type 'Analysis' in the LABEL field.
3) Select a suitable IMAGE.
4) Select LABEL DISPLAYED TO RIGHT OF IMAGE.
5) Select COMMAND PROCESSING in ADDITIONAL ATTRIBUTES.
6) Type 'analyst data=sugi24.students' in EXECUTE SAS STATEMENTS ON EXECUTION.
7) Press OK and then OK again to return to the tab.

This time, when using TESTAF, the Analyst Application window appears. (Figure 28)

Figure 28: The Analyst Application

Using function from SAS/ASSIST software

SAS/ASSIST software provides a point-and-click front-end to most SAS procedures.To use selected SAS/ASSIST function without using the entire product requires some additional SCL to provide links between the SAS/ASSIST windows and that of your application. First, the SCL variables used by the SAS/ASSIST windows need to be defined. They come first in your SCL as in Table 11.

Table 11: SAS/ASSIST variables

```
length  menustyl  $10
        alt_menu  $35
        autosave   $3
        confirm    $3
        slist     $35
        screen     $8
        dataset   $17 ;
```

Then the INIT block requires some additional SCL. (Table 12)

Table 12: INIT block, additional statements

```
byline = " ";
menustyl = 'NEW';
alt_menu = ' ' ;
autosave = 'YES';
confirm  = 'NO';
slist    = ' ' ;
```

Placing the SAS/ASSIST objects

1) Right click on the tab and select IMAGE ICON from the MAKE list.
2) Type 'Tables' in the NAME field.
3) Type 'Tables' in the LABEL field.
4) Select a suitable IMAGE.
5) Select LABEL DISPLAYED TO RIGHT OF IMAGE.
6) Press OK to return to the tab.

SCL for the SAS/ASSIST objects

The SCL used to call the SAS/ASSIST window consists of a SCREEN variable that contains the name of the specific SAS/ASSIST window, and a CALL DISPLAY statement with a range of parameters. (Table 13)

Table 13: SCL for SAS/ASSIST object: tables.

```
tables:
screen= 'REPORT';/* Name of ASSIST object */
call display('sashelp.assist.astrun.scl',
             screen, menustyl, alt_menu,
             autosave, confirm, slist,
             dataset);
return;
```

Only the name of the object and that contained by the SCREEN variable (in figure 13 it is REPORT) is unique to an object that calls SAS/ASSIST function.

The window that appears when you select the object may be familiar. (Figure 29)

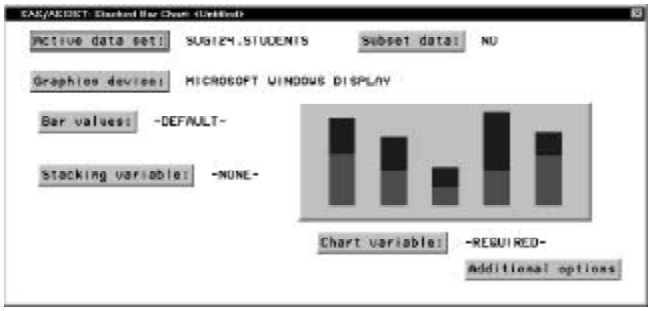

Figure 29: SAS/ASSIST window for vertical bar charts

You produce the objects for horizontal bars, tables and listings in exactly the same way; only the object name/screen name changes.

TIDYING UP

You have seen so far how to incorporate data entry and viewing, and how to provide simple reporting facilities. As a user, you also need to be able to close down the application, and from time to time, to do some housekeeping, such as clearing the OUTPUT window.

The final tab shows how to do this.

Emptying the OUTPUT window

In an application such as this you will wish to empty the OUTPUT window from time to time.

To add an object to clear the OUTPUT window (Figure 31):

1) Right click on the tab and select IMAGE ICON from the MAKE list.
2) Type 'Clear Output window' in the LABEL field.
3) Select a suitable IMAGE.
4) Select LABEL DISPLAYED TO RIGHT OF IMAGE.
5) Select COMMAND PROCESSING in ADDITIONAL ATTRIBUTES.
6) Type 'output;clear;af' in EXECUTE SAS STATEMENTS ON EXECUTION.
7) Press OK and then OK again to return to the tab.

Figure 31: Clearing the OUTPUT window

Exiting the application

To add an object to exit the application (Figure 32):

1) Right click on the tab and select IMAGE ICON from the MAKE list.
2) Delete the contents of the LABEL field.
3) Select a suitable IMAGE.
4) Select LABEL DISPLAYED TO RIGHT OF IMAGE.
5) Select COMMAND PROCESSING in ADDITIONAL ATTRIBUTES.
6) Type 'end' in EXECUTE SAS STATEMENTS ON EXECUTION.('endsas' instead of 'end' will end the application and exit SAS software entirely).
7) Press OK and then OK again to return to the tab.

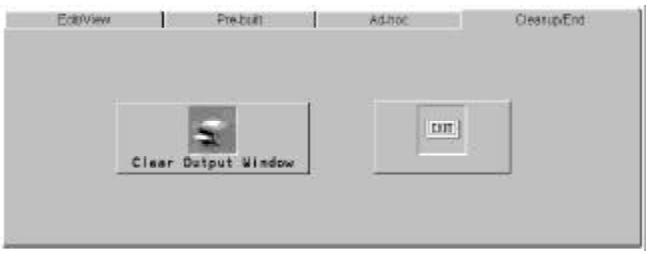

Figure 32: EXIT

The application is complete

The incorporation of the last two objects completes the application.

CONCLUSION

This paper has shown that it is entirely possible for a novice user of SAS/AF software to build a functional application using a minimum of knowledge.

You can use very simple techniques to introduce yourself to applications development and yet still provide something that is effective in use.

Naturally, there is far more to developing applications than can be shown in this simple beginning. You can learn the skills involved by attending suitable training courses and reading both documentation and books on the subject - plus, of course by attending user group conferences such as SUGI 24.

ACKNOWLEDGEMENTS

This presentation originated as a guide to a student project at the University of North London in the United Kingdom.

The project was initiated by Dr. Robert Rigby (Senior Lecturer and Researcher) and Mrs. Helen Robert (Principal Lecturer in Statistics) at the University, who wanted to include an element of applications development in a course project. I learned a lot about the practicalities of incorporating such enhancements in a tiny timeframe, with a student perspective, from Diana Asuen, whose project it was. Diana now works for OCS Consulting Ltd, a SAS Quality Partner.

AUTHOR

Bruce Bovill
SAS Institute
Wittington House
Medmenham
Marlow
Buckinghamshire
SL7 2EB
United Kingdom

Telephone: +44 (0) 1628 404347
E-mail: sukbnb@suk.sas.com