Paper 193-27

Getting Started with SAS/AF® Software

Steven A. Wilson, MAJARO InfoSystems, Inc., San Jose CA

ABSTRACT

This workshop will be an introductory guide to the development of interactive computer systems using Frame technology in the SAS System Version 8. This paper is intended for both current applications developers as well as SAS System programmers who wish to learn how to begin developing GUI applications using Frames in SAS/AF.

In this workshop we will examine the basics of Frame technology, including the Applications Development environment, SAS/AF components and their properties, and ways to add system functionality by using the SAS Component Language (SCL).

At the end of this workshop you will have a simple SAS/AF application which you can use as a starting point for building your own SAS/AF applications.

INTRODUCTION TO SAS/AF

SAS/AF is the SAS System product that allows developers to create interactive applications using an object-oriented approach to development.

A screen that is built to have a GUI interface is called a Frame and is stored as an entry in a SAS catalog. SAS/AF applications are typically comprised of catalogs containing many Frame entries as well as other entry types (eg. SCL, Keys, Pmenu, Formats) required by the application.

SAS/AF Frames are populated with **components** (buttons, entry fields, etc.) required for system execution. These components are controlled by specific **attributes** and perform specific actions that are defined by **methods**.

The application can invoke SAS Component Language (**SCL**) code or other windows or Frames as required to perform the necessary actions requested by the user.

During development, the SCL used in the application is compiled. This compiled code is used during execution of the application.

SAS/AF applications are client-server applications that execute on your desktop machine. This is true even though the SAS System may reside on a server machine. Thin-client applications distributed over the Internet are developed using the SAS Web/AF product in the AppDev Studio of products.

APPLICATIONS DEVELOPMENT

Following a Software Development Life Cycle (SDLC) is good application development practice. This approach identifies a series of steps that an application development project should follow. The steps most often associated with a SDLC are:

Functional specifications

It is crucial to have a detailed understanding of the actions to be accomplished by the application. As a developer, you have the skills to create an application, but you must rely on the application's user community to define the needs and functionality of the application.

Design specifications

This is where programs and their user interfaces, inputs, outputs, data storage structures, parameters and other technical details are identified. The goal here is to transform the functional requirements into the logic and design necessary to begin development. Pseudo-code may also be written as part of a detailed design specification.

Coding

Coding from design specifications is guaranteed to produce an application that requires less re-work than an application developed in an ad-hoc fashion. Coding from design specifications also reduces the actual time spent coding.

Be certain that coding standards are followed to ease the task of code review and maintenance.

Acceptance testing

QA testing should ensure that the functional specifications are met and that the application produces accurate results. Document how each functional element is tested.

This testing will result in an application that is ready for release or a request for further development.

Release

A formal sign-off for application release should not be done until a plan for providing technical support and accepting enhancement requests is in place.

Change Control

This is a crucial step for applications with a large scope or a long life span. The change control process describes how to manage the bug reports, enhancement requests and new software releases.

Rapid Prototyping is a common application development approach that fits into the SDLC. This development approach invests less time in the design and coding phases in order to get an application quickly into an abbreviated acceptance testing phase. With this approach, it is expected that speedy testing will quickly lead to refinements in the functional and design specifications. This results in a revised application returning for more acceptance testing after the design refinements have been implemented.

Eliminating application deficiencies during the design phase is much more cost effective than finding them later in the SDLC. However, this initial design time investment must be weighed against the very real need to quickly move software into production. Fortunately, application prototypes can be rapidly developed when using SAS/AF software in conjunction with a well-thought design.

SAS/AF DEVELOPMENT ENVIRONMENT

Object oriented coding techniques are used to develop SAS/AF applications. To work in this environment, you must know what objects are available for use in creating your application. All objects have specific properties that control the object and streamline the applications development process. Knowing the object properties (attributes and methods) is critical to being able to fully exploit the object in your application.

The SAS/AF development environment primarily consists of a set of four windows:

1) Build Mode of the Frame
   This is where the GUI is designed.
2) Components window
   This window lists the components available to be placed on the Frame.
3) Properties window
   This is where attributes for a component on the Frame are assigned.
4) Source window
   This is where SCL code is written and compiled for use by the Frame.

BUILD MODE

Using the SAS Explorer window, we can either request to open a new Frame entry or open an already existing Frame entry. When a Frame entry is opened via the SAS Explorer, the Frame is in Build Mode.

While in Build Mode, the other windows of the SAS/AF Development Environment are accessible. The Frame display may be enhanced and the Frame SCL may be accessed.

THE COMPONENTS WINDOW

When a Frame is open in build mode, the Components window displays the set of objects available to be placed onto the Frame. These objects are called components.

There are two types of components:

1) Controls, represented by an icon in color, are visible on the Frame. These are the objects which comprise the GUI.
2) Model components are not visible on the Frame, but instead are used to describe data, usually for display by a Control component. These components are represented by black-and-white icons.

Right-click in the Components window to access the **class dictionary** online documentation

It is important to know the components available in order to be able to produce the desired functionality of your application. There are approximately 25 different control components and about a dozen model components. While this may seem like a lot, this is actually a set of objects that are both familiar and intuitive.

MANIPULATING COMPONENTS

To create, or instantiate, a component on your Frame, drag the component from the Components window and drop it on the Frame. Once on the Frame, controls may be moved or resized as needed. You may also Copy and Paste controls from one Frame to another.

Press the Right Mouse Button on a control to open a popmenu with build time actions. These actions include component editing commands and any component-specific actions, such as Snug Fit for the Container Box control.

This popmenu also allows you to open the Properties window and the SCL Source window.

The mouse may be used to lasso multiple components and manipulate them together. You can select multiple controls to move, delete, copy, cut-and-paste, or align. Common attributes may also be assigned to multiple selected controls.

Use this lasso feature to select and align multiple components on the Frame via the Layout…Align… pmenu item or via toolbar buttons. It is a waste of your time as a developer to attempt to visually align the components on your Frame.

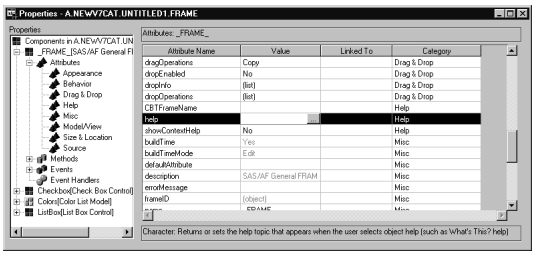
THE PROPERTIES WINDOW

Select Properties from the build time popup menu to open the Properties window. This is where component attributes are defined. Component attributes are characteristics, such as color, value, editable, borderTitle, etc.

When opened, this window displays the attributes for the currently selected component on the Frame or the attributes for the Frame itself if no component is selected.

The Properties window has two parts:

1) A drill-down component/properties tree on the left displays both visual and non-visual components on the Frame, as well as the Frame component itself.
2) A data table displaying the selected component's properties is on the right side of the Components window.



The Properties window provides the ability to assign values to most attributes via a selection list or push button. This ensures that valid values are entered for the attribute.

Attribute linking is performed in the `Linked To` attribute column. This causes an attribute value of a component to automatically change when the attribute of another component changes. For example, this can link the text in a Text Entry control with the name of a graph to display in a Graph Output control.

Since all attributes of a component can be accessed via SCL, component attributes can be queried and assigned programmatically. You may prefer to assign important attribute values via SCL, which makes the attribute assignment more visible than when it is assigned in the Properties window.

The set of available attributes is large. However, don't be intimidated by the entire set of attributes. The number of attributes that you will commonly use will be just a small subset of all the available attributes. Most attributes are intuitive and easy to assign.

THE SOURCE WINDOW

Each Frame that is created may have a separate SCL entry associated with it. SCL is a vast and mature programming language that is used to code the actions that are to be performed by your interactive application. While only a rare person will ever have a full grasp of SCL, it is easy to get started and to become productive with this extremely powerful language. These are some important topics to understand when using SCL:

1) SCL Label sections
   Each SCL entry has specific labeled sections of code. Each section executes at a specific time according to the name of the label. These sections execute in the order listed below:
   a) INIT:
      The INIT section of code executes when the Frame entry is first opened for execution. This section executes only once.

   b) ObjectName:
      Each control on the Frame has a name and this name may be used as a label for code that is to execute when the control on the Frame is modified.

   c) MAIN:
      The MAIN section of code executes each time any control on the Frame is modified. The `CONTROL` SCL statement can be used to control when the MAIN section is executed, if, for example, you would prefer that the MAIN section execute each time the Enter key is pressed.

   d) TERM:
      The TERM section executes when the Frame entry is being closed and is typically used to verify that required entries have been made and to perform clean up actions.

2) Declaring variables
   The `DECLARE` statement is used to define variables to the SCL data vector to better enforce variable scoping by providing more information to the SCL compiler than a simple `LENGTH` statement.

```
DCL       Char(20)     char_var     ,
          Num          num_var      ,
          List         listid       ,
          Object       object_id
;
```

   The data types LIST and OBJECT are used to define variables that will contain a list ID or object ID. This provides the compiler with additional information that is used to help prevent run time errors.

3) Lists
SCL Lists are temporary data storage structures that occupy memory during the execution of a SAS/AF application. Named lists contain an item value paired with a name value in the stored list.

Many component attributes are stored in lists and many SCL functions require lists, so it is important to develop a familiarity with their use.

4) Opening other windows
   a) Frames & SCL entries
   `CALL DISPLAY(`*entry*`) ;`
   The Call Display statement invokes the called entry, be it a Frame or a SCL program. When the called entry is closed, execution resumes with the statement after the Call Display.

   b) SAS Windows
   `CALL FSEDIT()` and `CALL FSVIEW()` use SAS/FSP to view or edit the contents of a SAS Data Set.

   To look at the attributes of a SAS Data Set, use the contents function:
   ```
   rc = CONTENTS ( data_set_name ) ;
   ```

   c) SAS File selection windows
   These functions open host selection dialog windows that allow users to easily navigate and select SAS files or external files:
   `OpenSASFileDialog OpenEntryDialog SaveSASFileDialog SaveEntryDialog FileDialog`

5) Processing SAS files
The OPEN function is used to open a SAS data set for use by a SCL program. Many SCL functions require a Data Set ID (dsid) that is returned by the OPEN function. Be certain to close the data set when it is no longer needed.

```
dsid = open('SASHELP.CLASS');
if dsid then
do ;
   nobs = attrn( dsid, 'NOBS') ;
   put '# obs in data set:' nobs ;
   rc = close(dsid);
end ;
```

6) Submit Blocks
Submit blocks are sections of code that contain Data and PROC Steps, or SQL statements that are submitted to the SAS System for compilation and execution. Typically these sections of code are used to invoke procedures to perform functions that are not available in SCL.

```
SUBMIT CONTINUE ;
   PROC PRINT DATA=SASHELP.CLASS ;
   TITLE 'From my AF Submit Block' ;
   RUN ;
ENDSUBMIT ;
```

7) Messages
   a) _msg_
   The value of the automatic SCL variable _msg_ is displayed at the bottom of the SAS window. Unfortunately, the user often overlooks these messages.
   ```
   _msg_ = 'This is a message';
   ```

   b) MessageBox function
   This SCL function displays the contents of a passed SCL list in a dialog box. Various options are available for specifying the message box icon, title, and the buttons that are to appear in the message box. The returned value is the button pressed in the message box:
   ```
   dcl char(3) button ,
       list    listid = makelist() ;
   rc = insertc(listid,
                'My message');
   rc = insertc(listid,
                'Message line #2',
                -1);
   button = messagebox( listid ) ;
   rc = dellist ( listid ) ;
   ```

8) SCL uses Dot Notation to set and get attribute values. This provides a coding shortcut for setting or querying attribute values.

```
textEntry.text = 'cvalue' ;
field_value = textEntry.text ;
```

Dot notation can also be used to invoke methods. Methods are actions that are performed by the component:
```
textEntry._hide () ;
```

COMPILE AND EXECUTE

Once you have written a SCL program, the program must be compiled. Use the `COMPILE` command to perform this action . Quite often errors in the SCL are identified during this compile. This will require you to debug your SCL program. Once the program has a clean compile, you can save it for execution.

Use the `TESTAF` command to test your Frame. This command executes your Frame, allowing you to verify that the desired functionality is obtained.

Finally, once the application is developed, it may be invoked by the user by using the `AF` command and identifying the entry to execute:
```
AF C=lib.cat.entry.entrytype.
```

WORKSHOP SPECIFICATIONS

The most important steps in the development of a new application are the functional and design specifications. You need to know where you are going before you start driving down the road.

For this workshop, we have been asked to develop a rapid prototype of an application that has the following functional specifications:

1) Main Menu
   a) Display "SUGI 27" as large colorful text.
   b) Allow user to change the color of the text.
   c) Allow user to advance to Examine Data Set screen.

2) Examine Data Set screen
   a) User may select an existing SAS data set.
   b) For a selected data set
      i) Allow user to edit.
      ii) Allow user to view the contents.
      iii) Allow user to print.

Remember that with rapid prototyping we spend less time up front on design with the expectation that modifications will result from the functional testing.

WORKSHOP EXERCISE

When you are able to quickly walk thru this exercise you will have a good understanding of SAS/AF basics that will allow you to begin developing more complex applications. Check off these steps with a pencil as you walk through this workshop.

CREATING THE MAIN MENU FRAME

1) Start a SAS Version 8 session.

2) Open the SAS Explorer window by entering the command `EXPLORER` on the command line, or clicking on the Explorer toolbar icon.

3) Select the WORK library in the Explorer window.

4) Create a new catalog in the WORK library by selecting File…New from the pull-down menu and selecting Catalog. You can call the catalog *New*.

5) Select the WORK.NEW catalog in the Explorer window.

6) Create a new Frame in the created catalog by selecting File…New from the pull-down menu and selecting Frame. A new Frame will be opened and the Components window displayed.

7) Grab a corner of the Frame and make it larger.

8) Drag a Combo Box control from the Components window onto the middle of the Frame. This will allow selecting a color.

9) Drag a Graphic Text control onto the Frame and place it to the right of your Combo Box control. This will hold the "SUGI 27" text.

10) Select the List Box control in the Components window.

11) Double-click on your Frame below the Combo Box control. This instantiates a List Box control on the Frame at the position where you have double-clicked. This is another object we could use to display colors.

12) Drag a Library List model onto the List Box control. This links the model (listing the allocated SAS libraries) with the List Box as a viewer. You should see the library list immediately populate the List Box (SASHELP, SASUSER, WORK).

13) Drag a Color List model onto the Combo Box control. It is **important** that the Model be dropped exactly on top of the Combo Box. For a typical arrow cursor, the tip of the pointer should be on the Combo Box when dropping the Model.

    This links the model (listing the available colors) with the Combo Box as a viewer. Unlike with the List Box, you do not see this linkage at build time. However, during execution, the Combo Box will display a selection list based on the colors returned by the Color List model.

14) Right-click on the List Box control in the Frame. Select `Delete` from the popmenu that appears to delete the List Box control. Note that this does not delete the Library List model attached to the List Box control.

15) Select the Push Button control in the Components window.

16) Double-click on your Frame below the Combo Box control to instantiate a Push Button.

17) Double-click on your Frame below the Graphic Text control to instantiate another Push Button.

18) Lasso the buttons and align them using Layout…Align…Middles from the pull-down menu.

19) Save the Frame by selecting File…Save from the pull-down menu and entering an Entry Name of *MainMenu*. You may enter a description too.

PROPERTIES FOR THE MAIN MENU

20) Right-click on the Frame and select `Properties` from the popmenu that appears. This opens the Properties window.

21) Each of the components we placed on the Frame appears in the Properties window. Notice that there is also a Frame component as well as the Library List model created in step 12, which was not deleted in step 14.

22) Delete the Library List model by Right-clicking on the Library List model in the Properties window and selecting Delete from the popmenu.

23) Verify that there is only one Color List Model component defined, Colorlist1.

24) Assign attributes to the components. Simply click on the indicated component and assign the Attribute in the data table. Notice that selection lists are presented to ensure you select a valid attribute value.
    a) _FRAME_
        i) backgroundColor — Gray
        ii) bannerType — None
    b) Combobox1
        i) name — Color
        ii) selectedItem — Blue
        iii) model — Colorlist1
            The Colorlist1 model is assigned via drag-and-drop
    c) Pushbutton1
        i) name — ok
        ii) label — Done
        iii) icon — 111
        iv) buttonStyle — Icon only
        v) commandOnClick — END
    d) Pushbutton2
        i) name — next
        ii) label — Next…
    e) Graphic Text
        i) name — display_text
        ii) text — SUGI 27
        iii) Click in the `Linked To` column of the Color Attribute, select Color as the Component, and selectedItem as the Attribute. This links the color attribute of the graphic text to the selectedItem attribute of the combo box.

25) Close the Properties window

TESTING THE MAIN MENU

26) Test the Frame entry by executing the `TESTAF` command. Note the functionality obtained via the attribute linking without the use of SCL.

    Also, the Next button does not work because we have not implemented this functionality yet.

Assume that testing resulted in the following changes to the Main Menu functional specifications:

a) Allow user to enter text to be displayed instead of a hard coded “SUGI 27”.
b) Label the color selection box “Select color”
c) Limit colors to Black, Blue, and Red.
d) Permit the user to specify the text color only if the user has entered text.

REVISING THE MAIN MENU FRAME

27) Return to the Frame in Build Mode.

28) Double-click on the Text Entry component in the Components window. This instantiates a Text Entry control on the open Frame.

29) Click the Text Entry control on the Frame and reposition the Text Entry control above the combo box control. You can move a selected control when the cursor appears as a hand icon.

30) Click and hold the mouse button. Draw a lasso around both the Text Entry control and the Combo Box control and release the mouse button to select both controls.

31) Left Align the two controls by selecting Layout…Align…Lefts from the pull-down menu.

32) Instantiate a Text Label control to the left of the Text Entry control.

33) Now open the Properties window and modify the component attributes.
    a) ColorList1 -- delete this component --
    b) Color
        i) borderStyle — Simple
        ii) borderTitle — Select color
        iii) borderTitleOffset — 6
    c) Textentry1
        i) name — userText
    d) Textlabel1
        i) label — Enter text:
        ii) justification — Right

34) Close the Properties window. Notice the difference in how labels were assigned to the user entry fields.

    Also notice that we did not use attribute linking to link the userText to the graphic text. Rather than using attribute linking, as done previously to link the color with the text, we will link the text value via SCL.

WRITING SCL

35) Right-click on the Frame and select `Frame SCL` from the popmenu that appears. This opens the SCL entry that will execute with the Frame.

36) Enter the following text into the SCL entry:

```
* SCL for MAINMENU.FRAME ;

INIT:
  /* method to hide the combo box */
  color._hide() ; ❶
  /* assign combo box list items  */
  color.items = {'BLACK' ,
                 'BLUE'  ,
                 'RED'   } ;
RETURN ;

USERTEXT: ❷
  /* Examine value in text entry  */
  if userText.text ne ' ' then
  do ;
    /* SCL for attribute linking  */
    display_text.text =
                userText.text;
    /* unhide the combo box       */
    color._unhide() ;
  end ;
  else
  do ;
    color.selectedItem = 'Blue' ;
    color._hide() ;
  end ;
RETURN ;

NEXT:  ❸
  /* Open a frame if Next clicked */
  call display('EXAMINE_DS.FRAME') ;
RETURN ;

* END of SCL for MAINMENU.FRAME ;
```

❶ This code illustrates how to invoke simple methods on a component using dot notation.

The combo box, which we have named color, is removed from the Frame display in the INIT section by using the _hide() method. This method can be used with any component.

❷ A labeled section for the userText component is also included. This code executes when the Text Entry control is modified during execution. Again, using dot notation, we check whether a value has been entered into the text entry field. If so, it is used as the graphic text value. This is exactly the same functionality obtained when using attribute linking, as was done previously with the color combo box.

37) Compile your SCL program by entering the command `COMPILE` on the command line. You should receive a message at the bottom of your SAS System window that reads:

```
NOTE: Code generated for MAINMENU.FRAME
```

38) Review and correct your SCL program if necessary. Close the SCL program entry after it compiles successfully.

39) Test the Frame entry by executing the `TESTAF` command. Note the additional functionality obtained via SCL to set attributes and invoke methods. Also note that the Next button still does not work because we have not created EXAMINE_DS.FRAME ❸.

40) Close and save MainMenu.frame.

CREATE EXAMINE_DS.FRAME

Remember from our functional specifications that this Frame requires selecting an existing SAS data set. Once selected, we need to allow users to edit, print, and view the contents of the data set.

To accomplish this, we will create a Frame with various push buttons that perform each of these functions.

41) Select the WORK.NEW catalog in the Explorer window.

42) Create a new Frame in the catalog by selecting File…New from the pull-down menu and selecting Frame.

43) Instantiate a Text Entry component in the center of this Frame.

44) Instantiate five (5) Push Button controls. Place one button to the left of the Text Entry. Place the other buttons below the Text Entry.

45) Lasso the 4 bottom buttons and align them using Layout…Align…Middles from the pull-down menu.

46) Save the Frame by selecting File…Save from the pull-down menu and entering an Entry Name of *Examine_DS*. You may enter a description too.

47) Right-click on the Frame and select Properties from the popmenu that appears. This opens the Properties window.

48) Assign attributes:
   a) Textentry1
      i) name dsn
      ii) editable NO
   b) Pushbutton1 (the button next to Textentry)
      i) name select
      ii) label Select
   c) Pushbutton2
      i) name contents
      ii) label Contents
   d) Pushbutton3
      i) name edit
      ii) label Edit
   e) Pushbutton4
      i) name print
      ii) label Print
   f) Pushbutton5
      i) label Done
      ii) commandOnClick END

49) Close the Properties window and enter the following SCL for Examine_DS. There are less than 25 executable statements in this code:

```
* SCL for EXAMINE_DS.FRAME ;
DCL  num  rc ;

SELECT:
  /* Use dot notation to assign   */
  /* the data set returned from   */
  /* the SCL function to the      */
  /* text entry field.            */
  dsn.text =
      openSASFileDialog('DATA') ;

  /* Report the # of observations */
  if dsn.text = ' ' then RETURN ;
  dsid = open(dsn.text) ;
  nobs = attrn(dsid, 'NOBS') ;
  rc = close(dsid) ;
  _msg_ = '# of rows in data set:'||
          put(nobs,best.) ;
RETURN ;

CONTENTS:
  /* When Contents is pressed,    */
  /* display the contents of the  */
  /* selected data set.           */
  if dsn.text ne ' ' then
    rc = contents(dsn.text) ;
  else link MSG ;
RETURN ;

EDIT:
  /* Use FSEDIT to edit the data  */
  if dsn.text ne ' ' then
    call fsedit(dsn.text, ' ',
                    'EDIT') ;
  else link MSG ;
RETURN ;

PRINT:
  /* When the Print button is     */
  /* pressed, submit a PROC to    */
  /* process the data set.        */
  if dsn.text ne ' ' then
  do ;
     charval = dsn.text ;
     /* SCL variable values are   */
     /* substituted for macro     */
     /* variable references in    */
     /* submit block code.        */
     submit continue ;
       PROC PRINT
            DATA = &charval ;
       RUN ;
     endsubmit ;
  end ;
  else link MSG ;
RETURN ;


MSG:
  /* Display message to the user. */
  dcl  char(1)  button ;
  if not(msg_list) then
  do ;
     msg_list = makelist() ;
     rc = insertc(msg_list,
       'Please select a data set.');
  end ;
  button = messagebox ( msg_list,
                        '!', 'O' ,
                        'Note' ) ;
RETURN ;


TERM:
  if msg_list then
     rc = dellist(msg_list) ;
RETURN ;

* END of SCL for EXAMINE_DS.FRAME ;
```

50) Compile, debug, and save Examine_DS.

51) Execute your application by via the following command:

```
AF C=work.new.Mainmenu.Frame
```

As you can see, a large amount of functionality is available from just a few SCL statements.

Now let's assume that this testing has resulted in further clarification of the functional specifications. Instead of a text-based editor (FSEDIT), the users want a graphical, tabular editor.

Fortunately, such an editor exists as a component available with SAS/AF. Using this component will result in Model / Viewer application.

GETTING STARTED WITH MODEL/VIEW

Consider your experiences as a SAS programmer. You know that data can be described by a SAS view or stored in a SAS data set. Regardless of how the data are defined, the data can only be viewed by using some sort of SAS procedure to present the data, like PROC PRINT or PROC MEANS.

Similarly, in SAS/AF we have distinct concepts of describing data versus viewing data. Instead of procedures, we have SAS/AF Viewer components that are used to display data. To access or modify data, there are SAS/AF Model components. It is important to be able to understand this Model/Viewer concept in SAS/AF.

Models identify, access, and manipulate data. Viewers simply display the data.

As seen earlier, you can place model components on your Frame just like any other object. When associating a model with a viewer, drag the Model onto the Frame and drop it on a Viewer control.

Model/View communication is defined via attributes that establish communication between a model and a viewer. Since model components are Model/View enabled by default, when you drop a model on a viewer, SAS automatically establishes the Model/View relationship.

Many controls are Model/View enabled. Simple Model/View controls include the Combo Box, Radio Box, and the List Box.

With SAS Version 8.1, there are more complex components available. The Form Viewer control provides the ability to interact with a single row of data in a SAS data set. The Table Viewer interacts with a data set in a tabular layout.

The requested modification will require changing the Examine_DS Frame to eliminate the `call fsedit` and instead call a new Frame that will display the data set. This new Frame will require the use of a Table Viewer control and a SAS Data Set Model.

MODIFY EXAMINE_DS.FRAME

52) Change the `call fsedit` statement in the `EDIT` section of the Examine_DS SCL to a `call display` statement which will invoke a new Frame, TableViewer. The new Frame will accept the data set name selected in Examine_DS as a parameter.

    The new `EDIT` section looks like this:

```
EDIT:
  if dsn.text ne ' ' then
   call display('TABLEVIEWER.FRAME',
                dsn.text ) ;
  else link MSG ;
RETURN ;
```

53) Compile and save the modified Examine_DS Frame.

CREATE TABLEVIEWER.FRAME

54) Select the WORK.NEW catalog in the Explorer window.

55) Create a new Frame in the catalog by selecting File…New from the pull-down menu and selecting Frame.

56) Drop a Table Viewer on the Frame and resize it.

57) Drag a SAS Data Set Model and drop it on top of the Table Viewer. This links the model component with the Table Viewer control.

58) Instantiate a Push Button on your Frame below the Table Viewer.

59) Save the Frame by selecting File…Save from the pull-down menu and entering an Entry Name of *TableViewer*. You may enter a description too.

60) Right-click on the Frame and select Properties from the popmenu that appears. This opens the Properties window.

61) Assign attributes:
    a) SASDataSet1
        i) name — model
    b) Pushbutton1
        i) commandOnClick — OK
        ii) label — OK

62) Close the Properties window and enter the following SCL for TableViewer:

```
* SCL for TABLEVIEWER.FRAME ;

INIT:
  /* accept the passed parameter. */
  entry dsn $ ;
  /* assign the table attribute   */
  model.table = dsn ;
  model.editMode = 'RowLevelEdit' ;
RETURN ;

* END of SCL for TABLEVIEWER.FRAME ;
```

63) Compile, debug, and save TableViewer.

64) Execute your revised application by via the following command:

```
AF C=work.new.Mainmenu.Frame
```

Use the Right Mouse Button on the Table Viewer to obtain a popmenu of commands used to manage data via the Table Viewer.

One final request is to add a new push button to Examine_DS.frame that opens a new Frame which allows frequency histograms of any character variable in the selected data set to be generated.

MODIFY EXAMINE_DS.FRAME

65) Return to Examine_DS.frame in Build Mode. Add a new push button control named CHART.

66) Add new code to invoke a new Frame, *Chart*. Compile and save the modified code.

```
CHART:
  call display('CHART.FRAME',
            dsn.text ) ;
RETURN ;
```

CREATE CHART.FRAME

67) Create a new Frame in the WORK.NEW catalog.

68) Double-click on the List Box Control to instantiate a list box on the new Frame.

69) Drag a Variable List Model and drop it on top of the list box

70) Drag a Histogram onto the Frame and drop it below the list box.

71) Instantiate a Push Button on your Frame below the histogram control.

72) Save the Frame as Chart.frame.

73) Assign attributes:

a) Variablelist1		
i)	typeFilter	Character
b) Pushbutton1		
i)	commandOnClick	OK
ii)	label	OK
c) Histogram1		
i)	XVariable	Linked To listbox1.selectedItem
ii)	XVariableType	Categorical

74) Enter, compile, debug, and save SCL for Chart:

```
* SCL for CHART.FRAME ;

INIT:
  /* accept the passed parameter. */
  entry dsn $ ;
  /* assign the dataset attribute */
  variableList1.dataset = dsn ;
  histogram1.dataset    = dsn ;
RETURN ;

* END of SCL for CHART.FRAME ;
```

75) Execute your revised application by via the following command:

```
AF C=work.new.Mainmenu.Frame
```

Be sure to select a data set with character variables to allow testing the new chart functionality.

CONCLUSION

SAS/AF is a vast and powerful product that has an almost unlimited set of functionality for implemented into an application. With such flexibility comes a steep learning curve. However, as demonstrated in this workshop, it is easy to implement simple functionality via component attributes. Use of just a few basic component method calls and SCL increases the available functionality greatly..

CONTACT INFORMATION

Steve Wilson is the Director of Clinical Applications Development at MAJARO InfoSystems, Inc., a SAS Alliance Partner that produces ClinAccess ™, a SAS-based clinical trials software package.

Steve has been developing applications in SAS for nearly 20 years and has given numerous presentations at SUGI as well as regional and local conferences.

Steve can be contacted with questions or copies of this workshop SAS code at:

SWilson@majaro.com