Paper 66-25

Getting Started with Version 8 SAS/AF® Software

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

Abstract

This paper 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 applications using Frames in SAS/AF.

In this paper 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 SCL (SAS Component Language).

Finally, this paper concludes with a step-by-step exercise that walks you through building a simple SAS/AF application.

Introduction to Frame Technology

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.

1) Structure of AF Applications
   SAS/AF Frames are displayed as windows in an interactive session. These windows are populated with objects (buttons, entry fields, etc.) required for system execution. These objects are controlled by specific **attributes** and perform specific actions that are defined by **methods**.

   The application can invoke SCL code or other windows or Frames as required to perform the necessary actions requested by the user.

2) Application Catalog
   Frames are stored in a SAS catalog as a Frame entry type. SAS/AF applications are typically comprised of multiple SAS catalogs with each catalog containing many Frame entries as well as other entry types (eg. SCL, Keys, Pmenu, Formats) that are required by the application.

3) Application Execution
   During the build process, the developer compiles the SCL used in the application. This compiled code is used during execution of the application.

   SAS/AF applications are fat-client 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.

Knowledge Goals

Learning SAS/AF can seem like a formidable task. However, if you break it down into a series of "knowledge goals", the learning curve may seem less intimidating.

These are the most important areas to develop your SAS/AF knowledge to become a successful AF application developer:

- AF Developer's Environment
  The developer's interface for performing the development and testing of your SAS/AF applications is easy to use!

- Objects and their Properties
  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.

- Object Communications
  Communication between objects on your frame is also accomplished easily via object attributes. These include:
  - Attribute Linking
    Allows any attribute value to be easily passed to another object on the Frame.
  - Model / View
    Objects on the Frame (viewers) are easily linked to Data or metadata models.
  - Drag and Drop
    Drag a value from one object to another.

- SCL Coding
  The SAS Component Language (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.

  Use of Object Oriented "Dot Syntax" as well as a standard coding style in your SCL makes coding easy to perform in your applications, and at the same time makes the code easy to maintain.

- Class Editor
  Beyond the scope of this paper, the Class Editor allows developers to easily create and maintain your own custom SAS/AF classes.

Opening a Frame Entry

In the SAS System, a superior interface is available to the application developer when in build mode. I sometimes refer to the AF development environment as the "build mode". This environment includes a number of windows, which make the development process a very GUI experience.

The SAS Explorer window, much like the Windows Explorer, presents an expandable tree of directories, folders, and files. Use the pmenu command File…New… to create new catalogs and entries within catalogs. You can open an existing entry displayed in the SAS Explorer by double clicking on it.

These are some points of interest when you open a frame entry in build mode:

✷ Frames always display scroll bars in build mode. However, during execution, scroll bars only appear when an object is outside the frame boundary. Horizontal and vertical scroll bars always appear together. This works for frames that are larger than the monitor, frames larger than the window size, and frames that are resized.

Unfortunately, text-based objects that are partially outside a frame boundary are not always displayed during execution. Since a text-based object must be completely inside the frame boundary to be displayed, be sure to keep these objects visible to the user. When you see a scroll bar on a frame during execution, this means that there is at least one object that is not being displayed or only partially displayed on the frame.

✷ The cursor movement resulting from use of the Tab key during execution can be explicitly defined. Each object on the frame is explicitly placed in a tab order list, which may be modified by the developer. This tabbing order list can be accessed during build mode from the pmenu item View…Tab order.

✷ Help for a Frame may reference an HTML document, an external file, or a Help entry in a SAS catalog.

✷ Frame attributes are defined in the same manner as object attributes. Typical frame attributes you assign are the window title and size as well as Pmenu, Keys, and Help entries for the frame to use. You can refer to the discussion of assigning attributes that appears later in this paper to understand how frame attributes may be assigned.

✷ Frame window sizes may be assigned by using the SETWSZ command. Simply size the Frame as desired and execute this command to set the size of the Frame during execution.

✷ Frame applications are portable to all SAS System platforms. Thus, development may take place on a PC environment, and the application deployed for use on a Unix network.

The Components Window

When a Frame is open in build mode, we are presented with a set of available objects in the Components window. These objects are called ***components***.

There are two types of components:

1) Controls
These components are visible on the frame. These components are pixel-based and use host system fonts. Thus, these components have a clean, crisp, host-specific look.

2) Models
These components are not visible on the frame, but instead are used to describe data, usually for display by a Control component. These components enable the Model-Viewer paradigm and are discussed later in this paper. I strongly encourage you to utilize these objects in your applications.

Separate icons in the Components window represent these two types of components. A black-and-white icon represents models, while an icon in color represents controls. The Components window displays the set of classes in the default resource list, SASHELP.FSP.AFCOMPONENTS.RESOURCE.

Each component is based on a class definition, which is stored in a catalog with a CLASS entry type. A RESOURCE entry contains a set of classes. The default Resources are:
1) Components
2) Version 6 objects

Generally, SAS supplied class entries are in SASHELP.CLASSES, while resource entries are in SASHELP.FSP. However, there are many other catalogs in the SASHELP library that contain these entry types.

Classes may be added and/or dropped and other resource entries may be added to the Components window. Simply right-click in the Components window to display a popmenu of actions. This popmenu is also how the **class dictionary** online documentation is accessed.

SAS/AF Classes

It is important to know the classes available in order to be able to easily produce the desired functionality of your application.

Don't be intimidated by the following list of classes supplied with SAS/AF. The component created by the class is exactly what you would expect from the class name, so it is quite easy to become familiar with the available set of components. You will find that your applications will mostly use just a few of these classes to accomplish most of your tasks.

1) Executing commands

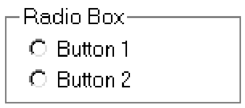
a) Desktop Icon

b) Push Button

2) Selecting items

a) Check Box

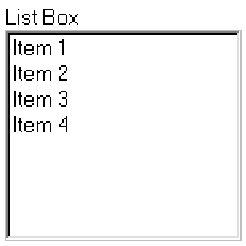
b) Radio Box



c) Combo Box
This control produces a drop-down list of acceptable choices for entry into the field.

d) Spin Box
This control allows the user to scroll through either a range of numeric values or a set of character values.

e) List Box



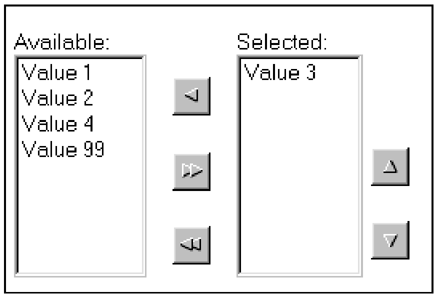
f) Scrollbar

g) Library Selector*
Control to select assigned SAS libraries.

h) Member Selector*
Control to select SAS files.

i) Catalog Entry Selector*
Control to select entries from a catalog.

j) Dual Selector *
This is a selection tool with two lists – one list of available selections, the other list of selected items.



3) Data collection

a) Text Entry
Allows simple text entry into a field.

b) Text Pad
This notepad with text wrapping capabilities is for collecting large amounts of text.

c) Data Table ⊗
A Data Table displays an entire data table in a spreadsheet layout, with columns and rows.

d) Data Form ⊗
The Data Form is an object that is used to display a row in a table for edit or browsing.

4) Annotations

a) Container Box
This is used to draw a box on the Frame

b) Graphic Text
Large, colorful text using SAS/GRAPH fonts.

c) Text Label
Simple, black & white text using host fonts.

5) Graphics
Use of these controls requires SAS/GRAPH.

a) Graph Output
This displays a generated SAS/GRAPH output.

b) Chart

c) Histogram

d) Pie

e) Scatter

f) Critical Success Factor

g) Map

6) Models
See the later section in this paper.

7) Browsing Files

a) Catalog Entry Viewer ✦

b) External File Viewer ✦

c) Video Player ✦

* Experimental, unsupported classes that do not appear in the default resources. To access these classes, you must add either the resource SAShelp.fsp.experimentalafcomponents.resource to the Components window, or add the desired classes to the Components window. These components may become available in Version 8.1 or Version 9.

⊗ These components will be available in Version 8.1. Until then, you must use the Version 6 object.

✦ There are no Version 8 components for these objects so the Version 6 objects must be used. Version 6 objects are located in the Version 6 Objects resource.

Instantiating and Manipulating Components

There are numerous ways to place a component on your frame from within the Components window:

1) Double-click on a component.
2) Drag a component onto the frame.
3) Right-click and drag a component onto the frame allows you resize the component when it is instantiated.
4) Select a component, then double-click in the frame at the position where the component is desired.

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 opening the Properties window and any component-specific actions, such as Snug Fit for the Container Box control.

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. There is no need to visually attempt to get your objects lined-up.

Component Attributes

Pressing the Right Mouse Button while on an open frame presents a popmenu with build time actions. Select Properties to open the Properties window for the selected component, or for the frame, if no component is selected.

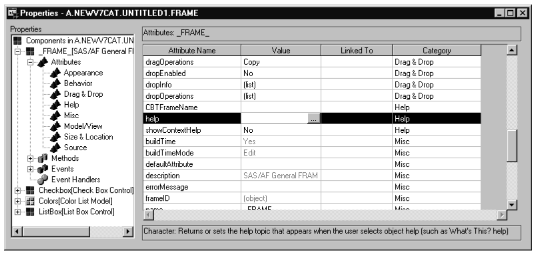
The Properties window has two parts:

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. You can click on components on this tree diagram to specify the active component. Right-click on the component or component property to open a popmenu of actions, including access to the class dictionary for the component.

   By drilling down into a component via the tree diagram, you can access the other component properties: Methods, Events, and Event Handlers. Only component Attribute properties are discussed in this paper. The other component properties are beyond the scope of this paper.

2) A data table displaying the active component's properties is on the right side of the Components window. By default the displayed properties are the component attributes.

An attribute window for a frame with multiple components is illustrated below. Please don't strain your eyes! You will soon be all too familiar with this window during applications development.



Attribute Name	Value	Linked To	Category
dragOperations	Copy		Drag & Drop
dropEnabled	No		Drag & Drop
dropInfo	(list)		Drag & Drop
dropOperations	(list)		Drag & Drop
CBTFrameName			Help
help	...		Help
showContextHelp	No		Help
buildTime	Yes		Misc
buildTimeMode	Edit		Misc
defaultAttribute			Misc
description	SAS/AF General FRAM		Misc
errorMessage			Misc
frameID	(object)		Misc

Character: Returns or sets the help topic that appears when the user selects object help (such as What's This? help)

The Properties window provides the following:

- Access to the object attributes, region attributes, and other properties, in a consistent format.
- 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 categorization allows you to sort the attributes within category (Appearance, Behavior, Help, etc.) or subset the list of attributes to see only a desired category of attributes.
- Attribute linking is performed in the Linked To attribute column. This allows one component to change an attribute value when the attribute on another component is changed. For example, this can link the name of a graphic in a text entry control with the graph to display in the graph output control. No SCL is needed!
- Drag and Drop is easily implemented by assigning the DragEnabled attribute of the object to drag from and the DropEnabled attribute of the object to drop on. The DragInfo and DropInfo attributes determine the value to take when dragging and where it is placed when dropped. Again, no SCL needed!
- Model/View communication can be specified via the attributes. This is discussed later.
- Ability to manipulate the shared attributes of multiple selected components. For example, selecting both a Text Entry control and a CheckBox control would allows you to specify common attributes (BorderColor etc.) for both of the controls at the same time.

The use of component attributes will significantly reduce the amount of SCL code you need to write for your applications.

Note that all attributes of a component can be accessed via SCL. This allows component attributes to be accessed and assigned programmatically.

Commonly used attributes are listed on the next page. Again, don't be intimidated by this list, as these attributes are intuitive and easy to assign.

Commonly Used Control Attributes

1) **Attributes used by Many Controls**
   a) BackgroundColor

   b) BorderColor
   c) BorderLightSource
   d) BorderStyle
   e) BorderTitle
   f) BorderTitleColor
   g) BorderTitleFont
   h) BorderTitleFontScaling
   i) BorderTitleJustification
   j) BorderTitleOffset
   k) BorderWidth

   l) Font
   m) Justification

   n) CommandOnClick
      Command executed when clicked.

   o) Help
   p) HelpText
   q) ToolTipText

   r) LabelColor
      Not available under Windows.
   s) TextColor
      Not available under Windows.

2) **Frame attributes**
   a) Appearance attributes
      i) BannerColor
      ii) BannerType
      iii) CommandAttribute
      iv) ForegroundColor
      v) Title
   b) Other Attributes
      i) KeysEntry
      ii) ForcePmenuOn
      iii) PmenuEntry
      iv) Help
      v) AutomaticCompile
      vi) Type
      vii) ShowBlockCursor
      viii) Icon
      ix) CursorPlacement
      x) DefaultPushButton
      xi) ConfirmOnExit

3) **Text Label**
   a) Label

4) **Push Button**
   a) ButtonStyle
      Text and/or Icon
   b) Label

5) **Graphic text**
   a) Color
   b) Text

6) **Text Entry, Combo Box, Spin Box**
   a) Common attributes
      i) Editable
      ii) MaximumCharacters
      iii) UpperCase
      iv) DataType
      v) Marked Text
      vi) Format
      vii) Informat
   b) Other attributes for Text Entry
      i) Text
   c) Other attributes for Combo Box
      i) Items
      ii) AllowNewItems
      iii) SelectedItem
   d) Other attributes for Spin Box
      i) DataSource
      ii) Items
      iii) Maximum, Minimum
      iv) Text

7) **Check Box**
   a) Label
   b) Selected

8) **Radio Box**
   a) Items
   b) SelectedItem

9) **List Box**
   a) Title
   b) Selection Mode
      Single, Multiple selections allowed.
   c) Items
      Items in the list
   d) Rows
   e) SelectedCount
   f) SelectedItem
   g) SelectedItems
   h) Model

10) **Text Pad**
    a) Editable
    b) ScrollBars
    c) UpperCase
    d) Marked Text
    e) Text

11) **Chart, Histogram, Pie, Scatter**
    The attributes of these graphics controls are customized to the type of control, but a general functionality exists for all of these controls.
    a) ActionMode
       Move, Rotate, Tilt, Spin, etc.
    b) DataSet
    c) HeightVariable
    d) HeightVariableStatistic
    e) ColorVariable
    f) ColorVariableStatistic
    g) Footnote1
    h) Title1

SCL Coding
Each Frame that is created may have a separate SCL entry associated with it. This SCL entry is used to contain SCL code that executes as the Frame is used.

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. It does not execute again when control returns to the Frame after calling another Frame.

   b) Label sections
   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. This is especially useful when you wish a specific action to occur only when a certain control on the Frame is used (eg. Call another Frame entry when a Push Button is clicked.)

   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            mylistid ,
           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.

The Local Environment List is a special list that is automatically maintained by the SAS System. This list exists while you are in an AF application and can be used to pass information within your application. For example, I could place the User ID into the local environment list and then be able to obtain the User ID from the local environment list at any place in the application.

Place the value of the variable userid into the local list with a name of USERID:
```
rc = setnitemc(envlist(‘L’),userid,’USERID’);
```

Read the userid from the local list:
```
userid = getnitemc(envlist(‘L’),’USERID’) ;
```

4) Opening other windows
   a) Frames & SCL entries
   There are a number of SCL statements available for calling other Frames or SCL programs.
   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 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) Value Selection lists
   These host selection windows, generated by the liblist, dirlist, catlist, datalistn, datalistc, etc. functions, allow users to select from lists of metadata supplied in the function call (Library, Directory, Catalog, Data Set, etc.).

   d) SAS File selection windows
   These functions open host selection dialog windows that can be used to allow users to easily navigate and select SAS files or external files:
   OpenSASFileDialog, OpenEntryDialog, SaveSASFileDialog, SaveEntryDialog, FileDialog.

5) Processing SAS files
   a) Open and Close a Data Set
      i) OPEN
         The SCL OPEN function is used to open a SAS data set for access by a SCL program. The Data Set ID returned (dsid) is used to reference the data set in most SCL functions.
         `dsid = open('SASHELP.CLASS');`
      ii) CLOSE
         This function is used to close an open data set when it is no longer needed
         `if dsid then rc = close(dsid);`

   b) Reading & Writing observations
      i) CALL SET (dsid) ;
         This important statement is used to link the variables in the open SAS data set (dsid) with the SCL variables in the SCL program. Thus, when rows are read from a SAS Data Set, the values in the Data Set variables are automatically loaded into any SCL variables of the same names.
      ii) rc = WHERE (dsid , where_clause) ;
         This statement is used to subset the open Data Set using the passed Where clause.
      iii) rc = FETCH(dsid) ;
         This statement is used to read a row from the open SAS Data Set. The return code, rc, indicates whether a row was successfully read.
      iv) rc = APPEND(dsid) ;
         This statement is used to add a new row to the open SAS Data Set.
      v) rc = UPDATE(dsid) ;
         This statement is used to update a row in the open SAS Data Set with the values in the SCL variables when linked via a CALL SET.

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

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

7) Messages
   I cannot stress the importance of providing messages to the user during the execution of your SAS/AF application. I spend huge amounts of my time as a developer coding the delivery of messages to the application user.

   a) _msg_
      The _msg_ is a message line that appears at the bottom of the SAS window. Placing messages in this line is very easy:

      `_msg_ = 'Message to appear on the screen' ;`

      Unfortunately, this message line is often overlooked by users. Thus, messages of importance should not be placed here.

   b) Message label
      I frequently place a Text Label component on the Frame and use this component to display messages. This is a better solution because the message appears in the Frame window and is less likely to be overlooked:

      `msg.label = 'Message in a Text Label.';`

      Dot notation in the SCL is to assign a value to the Label attribute of the Text Label component.

   c) MessageBox function
      This SCL function creates a host message box that will display a passed SCL list, displaying each item in the list as a separate line in the message 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:

```
listid = makelist() ;
rc = insertc(listid,'My message');
button = messagebox( listid ) ;
```

      This is the best solution for displaying important messages to the user. However, this solution requires more lines of code because you must build the list to be passed into the function.

   d) Messages to the SAS Log
      When extensive messages or diagnostics are needed, I often write these to the SAS Log. You can write as much as you want to the SAS Log and even use the Log to record a user's movement and actions while using your application.

      `put 'This message goes to the Log' ;`

8) SCL uses “Dot Notation” to set and get attribute values. This common coding convention in object oriented programming languages provides a coding shortcut for setting or querying attribute values.
   a) You can reference attribute values directly:
      ```
      ObjName.Text = ‘cvalue’ ;
      field_value = ObjName.Text ;
      ```
   b) You can use attribute references in SCL statements:
      ```
      value = 5 + ObjectName.attribute ;
      if ObjName.Text = ‘NO’ then do;
      ```
   c) Dot notation can also be used to invoke methods. Methods are actions that are performed by the component:
      ```
      ObjectName._method(‘parameter’) ;
      ```
      For example, the following statement uses dot notation to send the _hide method to the control named field:
      ```
      field._hide () ;
      ```

Note that attribute and method names are not case sensitive. However, the use of mixed cases is a coding convention that is quite useful for increasing readability of your programs. Use this convention of initial caps on words when coding your SCL. (This is a convention that is also used in Java and other object oriented coding.)

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 test 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
```

Model Components

This brief discussion of non-visual objects is included because use of these objects should be very important to you. Developers must use these objects in order to code effective SAS/AF applications that access SAS data sets or SAS metadata.

Model Components provide a way to access some type of data. The data to be accessed may be raw data, which you may permit to be modified, or metadata, data that describes other data.

Typically these model components are used in conjunction with a Viewer Component to provide a visual representation of the data being accessed by the model. The Viewer Component is the control that presents the data to the user. Since the viewer is data independent it may be used with different sources of data.

You instantiate a Model Component in your frame in the same manner as visual components. In other words, you can place models in your frame just like any other object. When associating a model with a viewer, you can drag the Model Component onto the frame and drop it on a Viewer Control.

Model/View communication, as previously mentioned, is defined via component attributes that establish communication between a model and a viewer. Since model components are already Model/View enabled, when you drop a model on a viewer, SAS automatically sets attributes to establish the Model/View relationship.

Although they are not visible on the frame display, the instantiated Model Component appears in the Property window object tree, where it may be selected, deleted, or similarly manipulated.

The table below lists the Model Components that are available in the default resource catalog. As with the control components, there are other model components in the SASHELP library that are not contained in the default resource catalog.

Version 8 Default Model Components	
Model	**Data**
Catalog Entry List Model	Catalog entry
Catalog List Model	Catalog in a data set
Color List Model	Available system colors
Data Set List Model	Data set metadata
External File List Model	External file
Library List Model	Librefs
LIST Entry Model	LIST catalog entry
Range Model	Min, Max, Increment for numeric range
SAS File List Model	Any type of SAS file
SLIST Entry List Model	SLIST catalog entry
Variable List Model	Variable
Variable Values List Model	Variable Values

A Simple Example

While space prohibits my including illustrations, you can follow the steps below to walk thru an exercise in building a simple SAS/AF application that illustrates most of the functionality described in this paper.

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.

Use a pencil to check off the steps as you complete them in this exercise.

Creating the Frame Entry

1) Start a SAS Version 8 session
2) Open the SAS Explorer window by entering the command EXPLORER on the command line.
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 will be displayed.
7) Grab a corner of the Frame and make it larger.
8) Double-click on the Text Entry component in the Components window. This instantiates a Text Entry control on the open Frame.
9) Click on the Text Entry control on the Frame
10) Reposition the Text Entry control to the center of the Frame. You can move a selected control when the cursor appears as a hand icon.
11) Drag a Combo Box control from the Components window onto the Frame below the Text Entry control.
12) Click and hold the right mouse button. Draw a lasso around both the Text Entry control and the Combo Box control. Release the mouse button when the lasso contains both controls. This selects both controls.
13) Left Align the two controls by selecting Layout…Align…Lefts from the pull-down menu.
14) Drag a Graphic Text control onto the Frame and place it to the right of your Text Entry and Combo Box controls.
15) Select the List Box control in the Components window.
16) 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.
17) 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).
18) Drag a Color List model onto the Combo Box control. 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.
19) Right-click on the List Box control in the Frame. Select Delete from the popmenu that appears to delete the List Box control.
20) Select the Push Button control in the Components window.
21) Double-click on your Frame below the Combo Box control to instantiate a Push Button.
22) Double-click on your Frame below the Graphic Text control to instantiate another Push Button.
23) Lasso the buttons and align them using Layout…Align…Middles from the pull-down menu.
24) Save the Frame by selecting File…Save from the pull-down menu and entering an Entry Name of Frame1. You may enter a description too.
25) Right-click on the Frame and select Properties from the popmenu that appears. This opens the Properties window.
26) Maximize the Properties window by entering the command ZOOM on the command line. Each of the components we placed on the Frame appear in the Properties window. Notice that there is also a Frame component as well as the Library List model created in step 16, which was not deleted in step 18.
27) Let's get rid of the Library List model by Right-clicking on the Library List model in the Properties window and selecting Delete from the popmenu that appears.

Assigning Attributes

28) Now we're going to assign some attributes to our 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)	font	size = 10
	iii)	selectedItem	Blue
c)	Textentry1		
	i)	borderStyle	Simple
	ii)	borderTitle	Enter text here
	iii)	borderTitleFont	size = 10
	iv)	borderTitleOffset	6
d)	Pushbutton1		
	i)	name	ok
	ii)	label	Done
	iii)	icon	111
	iv)	buttonStyle	Icon only
	v)	commandOnClick	END
e)	Pushbutton2		
	i)	name	next
	ii)	label	Next…
f)	Graphic Text		
	i)	name	display_text

   ii) Click in the Linked To column of the Text Attribute, select Textentry1 as the Component, and text as the Attribute.
   iii) Click in the Linked To column of the Color Attribute, select Color as the Component, and selectedItem as the Attribute.

29) Close the Properties window

Writing SCL

30) 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.
31) Enter the following text into the SCL entry:

```
* SCL for FRAME1.FRAME ;
INIT:
  display_text._hide() ;
RETURN ;
TEXTENTRY1:
  if textentry1.text ne ' ' then
    display_text._unhide() ;
RETURN ;
NEXT:
  call display('FRAME2.FRAME',
                    textentry1.text ) ;
RETURN ;
```

This code illustrates how to invoke simple methods on a component using dot notation. The graphic text object, which we have named display_text, is removed from the Frame display in the INIT section by using the _hide() method. This method can be used with all components.

A labeled section for the TextEntry component is also included. This code executes when the Text Entry control is modified during execution and causes the graphic text to become visible on the Frame via the _unhide() method.

32) 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 FRAME1.FRAME
```

33) Close the SCL program entry after it compiles successfully.
34) Test the Frame entry by executing the TESTAF command. Note the functionality obtained via the attribute linking and the use of methods. Also note that the Next button does not work because we have not created FRAME2.FRAME.
35) Close and save Frame1.Frame.

Create Frame2.Frame

36) Following the same procedure as in steps 6–9 and steps 19–24, create FRAME2.FRAME that contains:
    a) one Text Entry control
    b) five Push Button controls
37) Assign attributes:

a) Textentry1		
	i) name	dsn
	ii) editable	NO
b) Pushbutton1		
	i) name	select
	ii) label	Select
c) Pushbutton2		
	i) name	contents
	ii) label	Contents
d) Pushbutton3		
	i) name	browse
	ii) label	Browse
e) Pushbutton4		
	i) name	print
	ii) label	Print
f) Pushbutton5		
	i) label	Done
	ii) commandOnClick	END

38) Write the SCL for Frame2:

```
SELECT:
  dsn.text = openSASFileDialog('DATA') ;
RETURN ;
CONTENTS:
  if dsn.text ne ' ' then
    rc = contents(dsn.text) ;
RETURN ;
BROWSE:
  if dsn.text ne ' ' then
    call fsedit(dsn.text, ' ', 'BROWSE') ;
RETURN ;
PRINT:
  if dsn.text ne ' ' then
  do ;
    charval = dsn.text ;
    submit continue ;
      PROC PRINT DATA = &charval ;  RUN ;
    endsubmit ;
  end ;
RETURN ;
```

39) Compile, debug, and save Frame2.
40) Execute your application by via the following command:
    AF C=work.new.frame1.frame

Conclusion

There is a huge amount of information here for a Beginning Tutorial. Don't try to absorb it all at once, but refer to this paper over time and work towards your knowledge goals. Enjoy your SAS/AF experiences!

About the Author

Steve Wilson is the Director of Clinical Applications Development at MAJARO InfoSystems, Inc., a consulting and software development company for the pharmaceutical and biotech industries. MAJARO develops and markets ClinAccess ™, an integrated clinical trials system available for Version 8 SAS/AF software.

Steve has been developing applications in SAS for over 15 years and has given numerous presentations at SUGI as well as regional and local conferences.
Steve can be contacted at: Swilson@majaro.com