Moving to Version 8 in SAS/AF: An Application Developer's Perspective

Steven A. Wilson, MAJARO InfoSystems, Inc., Santa Clara CA

Abstract

Development of SAS/AF® applications in Version 8 is much different than Version 6 of the SAS System®; and much improved over Version 7. Version 8 completes the Nashville Project, representing a more robust realization of an object oriented development environment. This change will impact all SAS/AF development groups as Versions 7, 8, and 9 installed at sites throughout the globe.

MAJARO Info Systems, as a developer of SAS-based clinical software that is installed at many biotech and pharmaceutical companies, is acutely affected by these changes. Will our existing products still work? How does a new Version 8 component-based frame look in an application comprised of Version 6 widget-based frames and program entries? How much of our software will need to be rewritten to implement the new version of SAS?

This paper will attempt to summarize the changes that will be encountered as you move from SAS Version 6 Frame development to the new Version 8 development environment.

Introduction

This paper begins with an overview of the most important enhancements to grasp with V8 SAS/AF. The remaining sections of this paper discuss the author's experiences with executing and upgrading SAS /AF applications under V8.

I have also included an overview of a couple SAS/AF application development tools. These utilities, available under Version 6.12, are incredibly useful but not widely known.

The most important changes in Version 8 are these:

- Environment Changes
  The user interface for applications development is much different – and greatly improved!

- Object Properties and Communications
  Object properties have greatly streamlined the applications development process, which makes working with your objects much easier. Communication between objects on your frame is also accomplished much more easily via object attributes. These include:
  - Drag and Drop
  - Attribute Linking
  - Model / View

- SCL Coding Improvements
  Use of Object Oriented "Dot Syntax" makes coding easier in your applications, and at the same time makes the code is easier to read.

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

Opening a Version 8 Frame Entry

In Version 8 of the SAS System, a greatly improved interface is available to the application developer. This interface includes a number of new windows.

Execution of the build command opens the SAS Explorer window. You can also specify a working catalog, for example:

```
build sasuser.application
```

Note the catalog name above is greater than 8 characters. Names of SAS data sets, variables, catalogs, and catalog entries can now be up to 32 characters in length.

The build command is actually no longer necessary because the SAS Explorer window replaces the familiar V6 Build window. The SAS Explorer, much like the Windows Explorer, presents an expandable tree of directories, folders, and files. Like in the V6 Build window, use the pmenu command File...New... to create new catalogs and entries. As in V6, you can open an existing entry by double clicking on it.

These are some new frame properties of interest:

* Frames will now display scroll bars in build mode and during run mode when any 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.

* Tabbing order during execution can now be explicitly defined. Recall that in Version 6, use of the tab key to move the cursor during execution did not always result in the desired movement. Now, each frame control 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.

* In Version 8 a frame's Help attribute may reference an HTML document, an external file, or a SAS catalog entry.

* Frame attributes are assigned differently in V8. Typical frame attributes you assign are the window title and size as well as Pmenu, Keys, and Help entries for the frame to use. Since frame attributes are defined in the same manner as component attributes, you can refer to the discussion of assigning attributes that appears later in this paper to understand how frame attributes may be assigned.

Objects Have Become Components

In Version 8, we are presented with an entire new set of objects, both visual and non-visual, with which to work. In the Nashville Release, these objects are called ***components***.

The set of Version 6 objects with which you are familiar is still available in Version 8. However, as you will see, there are many advantages to using the new Version 8 components in your application.

There are two types of Version 8 components:

1) Controls
   These components are visible on the frame and are the objects formerly known as widgets. The good news is that these components are now pixel-based and use host system fonts. Thus, these new components have a much cleaner, crisp, and host-specific look.

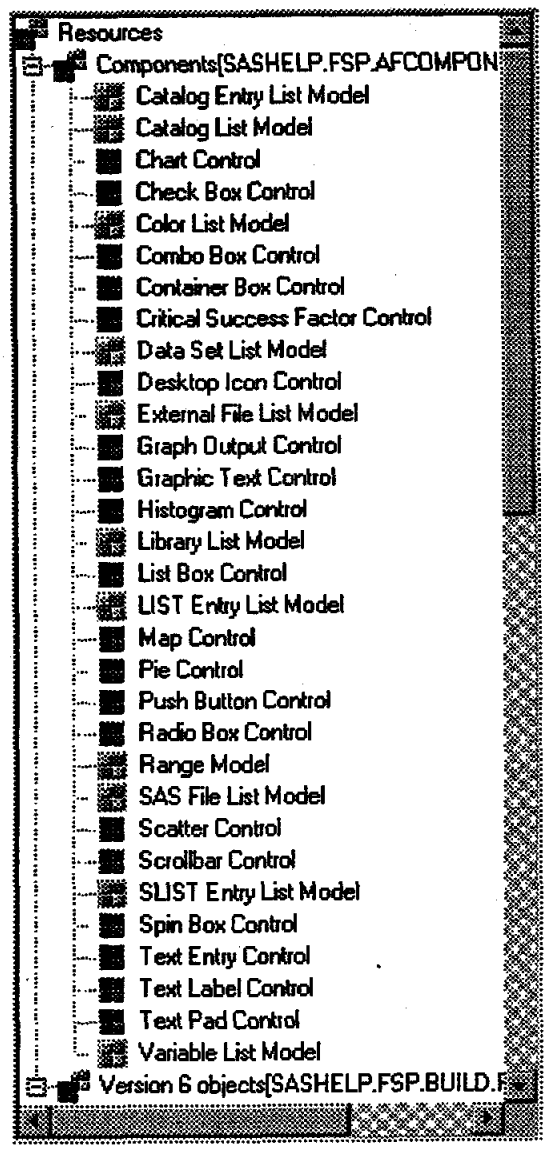
   The bad news is that because of this cleaner look, any application that mixes Version 6 frames and Version 8 frames will look schizophrenic because the SAS-created Version 6 widgets now look crude in comparison.

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. Let me just say here that the use of these objects has been greatly simplified in Version 8 and I strongly encourage you to utilize these objects in your applications.

These two types of components are represented by separate icons in the Version 8 Components window, as shown in the next column. The Components window displays the set of classes in the default resource list, SASHELP.FSP.AFCOMPONENTS.RESOURCE.

Classes may be added and/or dropped from this resource list and other resource lists may be added to this list. This is accomplished much more easily in V8. 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.

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



Most of the V8 components are simply upgraded V6 objects. In some cases V8 components make life simpler, for example, replacing the Input Field and Text Entry objects with a single Text Entry component.

The following tables are my attempt to correlate Version 6 objects with the Version 8 components. Most correspond one-to-one, but others did not evolve so simply. Also, there are a few components that are simply new constructs, while others are still lingering on as Version 6 objects.

V6 Objects Replaced with V8 Components	
Version 6 Widget	**Version 8 Component**
Block	Push Button Control
Check Box	Check Box Control
Command Push Button	Push Button Control
Container Box	Container Box Control
Control Object	Combo Box Control Spin Box Control
Critical Success Factor	Critical Success Factor Control
Extended Input Field	Text Entry Control Text Pad Control
Extended Text Entry	Text Entry Control Text Pad Control
Graphic Text	Graphic Text Control
Graphics	Chart Control Histogram Control Pie Control Scatter Control
Icon	Push Button Control
Input Field	Text Entry Control
Input Field Label	Text Label Control
List Box	List Box Control
Map	Map Control
Organizational Chart	Tree View Control*
Push Button	Push Button Control
Radio Box	Radio Box Control
SAS/GIS Map	Map Control
SAS/GRAPH Output	Graph Output Control
SAS Libname Listbox*	Library Selector*
SAS Members Listbox*	Member Selector*
Scrollbar	Scrollbar Control
Text Entry	Text Entry Control Text Pad Control
Text Label	Text Label Control

V6 Objects NOT replaced (Must continue to use V6 Object)
Catalog Entry Viewer
Data Form (can be populated with V8 controls)
Extended Table
External File Viewer
Hotspot
Image
Image Icon
OLE – Insert Object / Paste Special / Read Object
Process Flow Diagram
Slider
Tab Layout Object
Toolbar
Video Player
Work Area

New Version 8 Objects
Desktop Icon Control
Catalog Entry Selector Control*
Dual Selector Control*
List View Control*

*Experimental, unsupported objects that do not appear in the default resource catalogs. To access these controls, you must add the appropriate resource from SASHELP.FSP to the Components window:

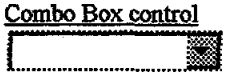
- AFCOMPONENTSPLUS.RESOURCE
- EXPERIMENTALAFCOMPONENTS.RESOURCE

Resource catalogs contain a set of objects. The default Resources are:

1) V8 components
2) V6 objects

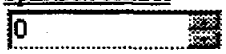
Common V6 composite objects built by many developers combined an Input Field widget with a Control widget, providing an arrow to click to obtain a selection list of acceptable values for the Input Field. These constructs exist as native Version 8 controls:

Combo Box control



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

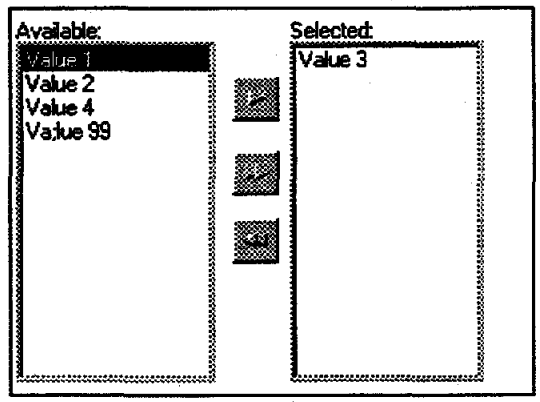
Spin Box control



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

Dual Selector control

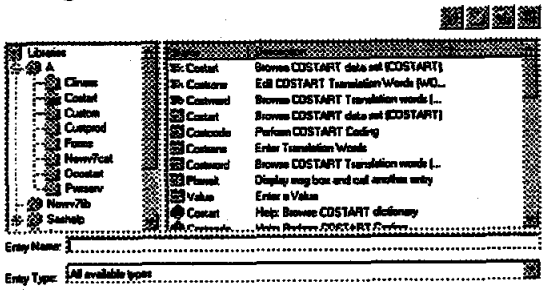
Another common composite is a selection tool with two lists – one list of available selections, the other list of selected items. A component like this is supplied as an experimental control in Version 8:



This control is stored in the following resource list: SASHELP.FSP. EXPERIMENTALAFCOMPONENTS.RESOURCE. To use this control, the resource list must be added to the components window.

Catalog Entry Selector control

Another experimental control, the Catalog Entry Selector control, will prove to be very useful. This control appears below and provides the ability to select entries from a catalog:



This control is stored in the following resource list:
SASHELP.FSP.
AFCOMPONENTSPLUS.RESOURCE.
To use this control, the resource list must be added to the components window.

Instantiating and Manipulating Components

The frame build environment in Version 8 is quite different than Version 6. First, the right mouse button is no longer used when creating (e.g. instantiating) objects on a frame. There is no make, no fill, nor can you drag out an empty region on the frame.

Instead, there are new ways to place a component on your open frame. Any of these methods will instantiate a component on your frame:

1) Double-click on a component in the Components window.
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 resized as needed.

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.

Eliminating the ability to use your mouse to draw a region on the frame allows the mouse to 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 (assigning attributes is discussed later).

Use this feature with to select and align multiple components on the frame via the Layout...Align... pmenu items or toolbar buttons. No more manually attempting to get your objects visually lined-up!

Another super enhancement is the ability to Cut and Paste controls from one frame to another. This is not intended to replace subclassing as a means to create custom components, but it does give you the ability to maintain consistency in component appearance and actions across multiple frames.

Component Attributes

Pressing the Right Mouse Button while on an open frame presents a popmenu with the item Properties. Selecting Properties opens the Properties window for the selected component, or for the frame, if no component is selected.

This Properties window displays component and frame attributes in a data table. This consistent window replaces all the individual V6 object attribute and region attribute windows. Of all the new windows in the V8 build environment, the Properties window will be the most used.

In Version 8 the behavior of an object is more likely to be controlled by attributes rather than methods. These attributes are stored with the component and essentially replace instance variables. Attributes, however, contain much more information than instance variables.

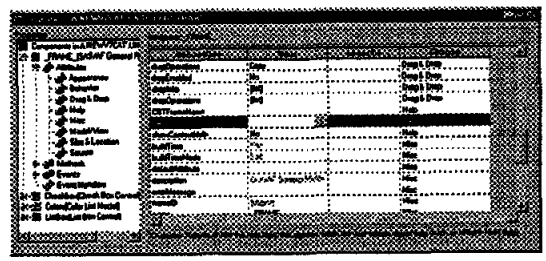
The Properties window has two parts:

1. A tree diagram displaying 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. 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.



The features I want to illustrate by this figure are the drill-down component/properties tree on the left, and that a data table is used to display the property items and settings on the right.

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 is also 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.
- When a V6 object is the active object, the Properties window will allow developers access to the old style V6 object attributes and region attributes windows.
- All attributes of a component can be accessed via SCL. This is another improvement over Version 6, where not all object attributes could be accessed programmatically.

The use of component attributes will significantly reduce the amount of SCL code you need to write for your applications. However, SCL will still be necessary. The next section details some improvements available with SCL coding.

SCL Coding Changes

The most important coding change is the availability of "Dot Notation" to set and get attribute values and to invoke methods. By providing a coding shortcut for invoking methods and for setting or querying attribute values, dot notation reduces typing and makes SCL programs easier to read. With dot syntax, there is no need to ever again write a call notify or a call send.

* You can reference attribute values directly:

```
ObjName.Text = 'cvalue' ;
field_value = ObjName.Text ;
```

instead of

```
call notify ( 'ObjName' , '_set_text_' , 'cvalue' );
call notify ( 'ObjName' , '_get_text_' , field_value);
```

You can use attribute references in SCL statements:

```
value = 5 + ObjectName.attribute ;
if ObjName.BorderColor = 'BLUE' then do;
```

* Dot notation can also be used to invoke methods:

```
ObjectName._method('parameter') ;
```

For example, the following statement uses dot notation to send the _hide_ method to the control named field:

```
field._hide () ;
```

instead of

```
call notify ( 'field' , '_hide_' ) ;
```

Notice too that all the underscores in SAS method names are now optional. Only the first underscore is now required. Thus, a method may be written as _set_border_color_ or _setBorderColor.

Attributes 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 V8 SCL.

* Use of dot notation for method calls offers the advantage of compile-time error checking, guaranteeing that you will never see the following message at run time:

ERROR: This method is not defined for the object class

* Methods may now return values, which is not possible in V6. Notice the new return= option on the method statement and the ability to return a parameter value via the return statement:

```
MYMETHOD: method return=num ;
  ... code.....
return ( 5 ) ;
endmethod ;
```

```
value = ObjectName.MyMethod() ;
```

will set value equal to the value returned by the method call, 5.

Or use method returns as values in SCL statements:

```
value = 5 + ObjectName.MyMethod() ;
if ObjectName.MyMethod() = 5 then..;
```

Error Messages
Since I mentioned error messages, let's look at this one. Do you notice anything strange about this message?

WARNING: Program halted. See LOG for messages.

Yes! We no longer have to search both the Message window and the Log window for the SCL abend messages. The Message window has been consolidated into the Log in Version 8.

An additional treat for developers: the dump of SCL variables when a program halt occurs is now a short dump. This compares favorably with the dump of the entire SCL data vector that occurs in Version 6. This will save a lot of time scrolling through the Log looking for SCL errors.

If you want the full dump of the SCL data vector at SCL abends, as is the default in Version 6, this may be requested via the SAS Registry. To access the SAS Registry, execute the command REGEDIT from the command line. Then drill down into Products...AF...Run Time to access the setting for the program halt SCL dump.

In many cases, the actual error messages written to the log are an improvement over the Version 6 error messages received at program halts. And by declaring your variables (DCL in next section), you no longer need statements like rc = rc ; to prevent compile Warnings.

New Functions
In addition to the coding changes mentioned in the previous section, there are also a number of new functions available. Some of the more useful functions for applications developers are described here:

button = call message (listid) ;
This command creates a host machine message box that will display the passed SCL list, displaying each list item in a separate line. The returned value is the button that is pressed in the message box.

call dialog (*entry*) ;
This is an important clone of the call display with the added ability to disable all other SAS windows until the called entry is closed. Note that this will prevent you from going to the Program Editor or the SAS Explorer, which may be problematic if you are accustomed to working in Build mode while your application is running.

CreatedDir = Dcreate (NewDir, ParentPath) ;
This creates a directory via your SCL code. Now we no longer have to see that unseemly DOS window when an application creates a directory.

rc = CompareList (listid1, listid2 <,options>) ;
This allows you to quickly determine whether two lists contain the same information. The rc=0 if they match.

N_obs_in_where = attrn (dsid , 'nlobsf') ;
The attrn function can now return the number of observations in a data set that meet the active WHERE clause.

rc=NameDivide(name, nparts, part1, .. part4) ;
This will separate a 2, 3, or 4 level SAS name into its individual pieces and also tell you how many pieces are in the name.

name=NameMerge (part1, part2, part3, part4) ;
This command will concatenate 2, 3, or 4 name pieces, returning a SAS file or catalog entry name.

lcCreate, lcDelete, lcValue, lcType
Integrity Constraints, new with Version 8, are a means of storing data validations with the data set definition to ensure that only valid values are loaded into the data set.

These functions allow you to create, delete and query a data set's Integrity Constraints. Customs messages to display when an Integrity Constraint fails, however, can only be defined via PROC DATASETS.

There are a number of component methods available for use with Integrity Constraints. Unfortunately, these methods require the name of the Integrity Constraint, but there is no way in SCL to obtain the name of the Integrity Constraint. This oversight will be corrected in Version 9.

rc = lock (member) ;
This function no longer requires SAS/SHARE.

Finally, the new 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 new 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.

I have found that it is necessary to declare the object ID of a component on the frame in order to invoke some methods on the object using dot notation. Otherwise, I surprisingly received a compile error.

Model Components
This brief discussion of non-visual objects is included because use of these objects should be very important to you. Developers must learn to 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.

- Raw Data
  - ✓ SAS Data Set
  - ✓ SCL List
  - ✓ Catalog SOURCE Entry
  - ✓ External File
- Metadata
  - ✓ Allocated Libraries
  - ✓ Allocated External Files
  - ✓ Data Sets
  - ✓ Variables
  - ✓ Catalogs
  - ✓ Catalog Entries
  - ✓ Colors

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 user interface that presents the data to the user. Since the viewer is data independent it may be used with different sources of data.

Use of Model objects in Version 6 was complicated to say the least. A paucity of documentation describing the use of these objects is another explanation for their lack of widespread implementation.

One of the great enhancements in Version 8 is the ability to 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 Version 8 model components are already Model/View enabled, when you drop a V8 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.

Coding is significantly reduced because there is no longer the need to initialize and setup non-visual objects via SCL. Nor are there any of the gymnastics with SCL lists so often necessary when working with V6 model objects.

This eliminates the Version 6 requirement of using the loadclass and instance SCL functions to create a non-visual object. Use of the model component attributes also replaces the need for calls to _setup_ and _get_members_ methods.

If this is incomprehensible to you, consider yourself lucky. Version 8 has removed this layer of complexity from the use of non-visual objects, resulting in much less code to work with Models. Again, this simplification of coding has resulted via the use of component attributes.

The table below lists the Version 8 Model Components that are available on the default resource list. As with the control components, there are other model components in the SASHELP library that are not contained in the default resource list.

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

Running Existing V6 Applications in Version 8

Since this is a new version of a software product, at an absolute minimum, you must re-certify your V6 application under V8. This means executing the Qualification Protocol or QA Validation Test that was originally used to certify your application under V6. This assumes that you have validated and documented the validation of your V6 application.

The results obtained from the execution of the validation under V8 must be the same as those obtained under V6 or adequate documentation of the differences must be provided.

As a test of how a V6 application would run in V8, I executed ClinAccess™, a sophisticated clinical information system developed by MAJARO InfoSystems, using V8. Being a large V6 SAS/AF application, ClinAccess seemed like it would be a challenging test of how well a V6 SAS/AF application will run under V8.

It is a simple matter to change the .exe used by an icon on the Windows desktop so that it points to SAS V8. In most cases with simple applications, this is all that you will need to do to run your existing SAS applications in V8.

However, after changing the .exe for the ClinAccess icon on my desktop, clicking the modified icon resulted in nothing!

What I discovered is that the customized config file used with ClinAccess in V6 was causing V8 to crash before SAS initialized. Some V6 config file options are no longer available.

These changes are not documented and required significant trial-and-error to diagnose. I found these differences in the custom ClinAccess config file:

-formchar	Character set must now be quoted
-wincharset	No longer supported
-helpenv	No longer supported
-$*option*	Likely replaced with a V8 option

After appropriately adjusting my config file, ClinAccess initialized just fine. However, I noticed a number of changes in my existing application. These are some common items that may require developer intervention to run a V6 application in V8:

1) When running an application in V8, you may wish to close the Results window to allow use of the entire SAS Application Workspace by your application. Use the following options in your SAS invocation in the order shown:
   -nodmsexp -dms

2) Selection lists are significantly different in appearance. These lists, generated by the liblist, dirlist, catlist, etc. functions, do not look anything like the SAS generated selection lists of V6. These have changed to be more similar to host-system windows of the same type.

   Since these windows are different sizes than their V6 counterparts, modifications will be required if you define the region size for these windows in your application.

3) Because In-cell editing within Data Tables is now permitted, you cannot simply double-click in a cell to execute the table object's labeled section of SCL (e.g. if a Data Table is used as a selection list). You must either double-click on the row number, or place the cursor in a cell before double clicking in the cell to run the label section. This can be turned off to regain V6 functionality.

4) Since V8 cannot update a V6 catalog entry, any V6 application that allows a user to edit or save a catalog entry (Source, Screen, Keys, Slist, etc.) must be modified to run under V8. In these cases, you may wish to convert the catalog to V8.

5) By default, when browsing a data set via CALL FSEDIT or PROC FSBROWSE, the cursor is not visible on the screen. This will cause problems if your application relies upon cursor positioning when using FSP screens. In V8, you may obtain V6 cursor visibility in browse via:

-hidecursor off	SAS invocation option
whidecursor off	command

6) SAS/SHARE will not initialize a server using your V6 code. A new parameter on the PROC SERVER statement is required to start a server:
   AUTHENTICATE= OPT | REQ

7) Custom pmenus created using an ID=1 on the ITEM statement of PROC PMENU will automatically gray the pmenu item. ID values in PROC PMENU must be greater than 3000.

8) Line drawing characters used on Program entry display windows using the OEM character set do not display properly because the OEM character set is no longer supported. Line drawing characters created in V6 using the Sasfont font and the ANSI character set were not tested.

This list of items is only a starting point. Rest assured that there are other items that you will find that require modification to get your application to adequately run under V8. Some obscure issue (such as the loss of the OEM character set) may seem minor, but depending on the extent used, could require substantial rework.

Considering the complex nature of the V6 ClinAccess product, I felt that this application ran well under V8.

Upgrading Existing Applications to Version 8

As discussed in the previous section, most V6 applications will require only minor enhancements to run under V8. Because of this, you may not want to devote significant resources to upgrading a V6 application to V8. The scope and life span of the application, as well as planned enhancements, should be assessed when making this decision.

If you plan to further enhance the application, you may wish to convert the entire application to V8 frames. This permits the use of the new development environment, Properties window, and long object names, as well as avoiding the unseemly appearance of an application that has both V6 and V8 objects.

Upgrading a V6 application to V8 is not as simple as replacing the V6 objects on the frame with V8 components. In addition to the items detailed in the previous section, there are more changes that are required when upgrading a V6 frame to a V8 frame:

1) Your V6 catalogs must first be copied into V8. This step is critical because once you begin making enhancements in the V8 catalog, there is no turning back without losing your changes. You cannot modify a V6 SAS/AF entry from within V8, nor can V6 modify a V8 entry.

2) When a Program entry is opened and closed in V8, the date of modification is updated, regardless of whether the entry was modified. You should be sure to use the Cancel command to close Program entries that are not modified to prevent this misleading indication. You will need to issue the Cancel command via your toolbar or a function key since it is not provided on the build environment pmenu.

3) **The issue of replacing V6 widgets with V8** components is optional, since V6 objects work in a V8 frame. However, aesthetics are also a consideration with your applications.

   All text based V6 widgets (text fields, Push button, List box, etc.) will look odd in comparison to their V8 counterparts. You may want to replace V6 text based objects with V8 components. This may be very time consuming.

   However, since the appearance of V6 graphical objects (Graphics object, Container box, etc.) do not stand out in a V8 application, you will only want to replace these objects with their V8 counterparts when specific V8 functionality is required or if you want to use object attributes.

4) Replacing V6 Text Entry/Control widget composites with a single Combo Box or Spin Box control can be tricky. In V6, two widgets are used to approximate a Windows Combo Box, so generally two labeled sections must be combined when replacing with a V8 Combo Box. Complicating matters is the fact that the selection list associated with the Combo Box is created when the component is initialized at run time on the frame.

   This precludes use of attribute linking with to control the list of items displayed in a child Combo Box according to the value selected in a parent Combo Box. This requires the use of Event Handlers and an override to the _init method of the parent Combo Box that will set the items attribute for the child Combo Box when the SelectedItemChanged event is triggered. Contat SAS Tech Support or the author for information.

5) If you have V6 objects that do not have counterparts in V8, you will need to find a work-around in order to upgrade these objects to V8. It is easy to change a V6 Toolbar object to a set of V8 push button controls. Other V6 objects may not be so easy to upgrade to V8.

   For example, you cannot upgrade a V6 Graphics object that permits the type of graph to be changed via the _set_type_ method. This is because in V8 there is no single Graphics control, but instead there are separate graphics controls for each type of chart (chart, histogram, pie chart, scatter plot). To upgrade to a V8 control, you must select which type of graphic to allow and use the appropriate control.

6) Establish a template frame (or classes) that contain a set of common standardized components that can be quickly copied into your new V8 frames. Objects can now be copied from one frame to another.

7) **It is common in V6 to use an assignment** statement to assign the default attribute to an object. For example, your V6 frame SCL may have statements like this:

   ```
   widgetx = 'value' ;
   ```

   where widgetx is the name of a Text Entry, Text Label, or some other widget that accepts character data as its default attribute.

   Such statements that reference a V8 component will fail with the following message:

   ```
   ERROR: [Line 99] widgetx is an object
   handle in V8. The value of the object handle
   for frame widgets cannot be reassigned.
   ```

   When upgrading your V6 objects to V8 components, you must modify all such SCL statements that are used to assign a value to an object. Thus, the previous statement would be re-coded for use with V8 components as:

   ```
   widgetx.Text = 'character value' ;
            for a Text Entry component, or
   widgetx.SelectedItem = 'value' ;
            for a Combo Box component.
   ```

8) Although dot notation is not required in a V8 frame, it may be desirable to re-code the call send and call notify statements with dot notation and to use component attributes. For this investment of time, you gain more maintainable code and compile-time error checking of methods. Dot notation may be used with V6 objects in V8 by clicking the option "Use object name as ID in SCL" in the Properties window for the V6 widget.

9) You must decide whether to implement the use of long variable and table names into your application. Under V8, variable names and data set names may be up to 32 characters, making the 8-character limit in your V6 application obsolete.

10) A number of concerns, mostly appearance related, are associated with the V6 Data Form object running in V8. I refer you to the SAS Notes page at www.sas.com for details. Also note, unfortunately, that a Text Pad object may not be placed on a Data Form. This is a significant oversight since the length of character variables may now be much longer than 200 characters.

11) Upgrading a V6 Program entry to a V8 frame is more difficult. You should be aware of the issues encountered with this type of upgrade if you have previous experience with converting Program entries to Frame entries in V6. For simple Program entries, you may only need to replace the Program fields with appropriate V8 controls and then follow the previously described steps for conversion.

SAS/AF Development Tools

This section discusses some AF development tools that were available in Version 6.12. If you have not already, I urge you examine these extremely useful applications development tools. I am not exaggerating when I say that the information made available by these tools is incredibly valuable.

These utilities can be classified as Static, which take a snapshot of an application, and Dynamic, which monitor the execution of an application. All the information you need to effectively use these utilities is available via online help.

Static Analyzer

My favorite, this tool collects almost every bit of information possible about an application catalog. This information includes flow charts; identification of variables, functions, formats, methods, classes, entries, method blocks, macros, options, lists, and constants used in the catalog; I/O usage; error and warning messages generated at compile time; identification of unreachable code. I'm sure I forgot something.

This information is gathered by executing the following code:

```
proc build batch ;
   crossref   project = lib.cat   outlib = outlib ;
run ;
```

After the data is collected from the snapshot of the application, execute the following command to view the collected information:

```
sclprof static lib = outlib
```

Because this information is stored in various data sets in the outlib= library, the data may be compared with data sets generated by this utility after modifications are performed to the application, allowing you to see the results of the changes to the application code.

Dynamic Performance Analyzer

This is a terrific utility to monitor an executing application and provide reports of wall time and execution time for called programs as well as individual SCL statements.

From the command line, execute either of the following commands to begin a data collection phase:

```
sclprof timer on
af c=lib.cat.entry.et sclprof = timer
```

During data collection, all actions performed in a SAS/AF application are recorded along with the time taken to perform the action. This gives a developer information concerning any execution bottlenecks that may be in the application, permitting the developer to address issues affecting execution of the application.

To end the data collection phase and immediately open the Performance Analyzer window, enter the following command or end the executing application.

```
sclprof timer off
```

Dynamic Coverage Analyzer

This QA tool monitors the execution of an application and reports on lines of code that are not executed. This gives software validators the exact location of code that is not executed. From this, the robustness of QA testing of an application can be determined.

As with the other dynamic development tools, data collection begins by issuing a command:

```
sclprof coverage on
af c=lib.cat.entry.et sclprof = coverage
```

The Coverage Analyzer window opens when the data collection phase is terminated.

Conclusion

As with all software upgrades, there will be difficulties encountered and patches that will need to be made to your applications. Although very few complex V6 applications will run seamlessly under V8, most applications will likely experience only minor irritations when running under V8.

While some of these irritations will require you to re-work some SAS/AF entries, you should be able to certify your V6 application to run under V8 without much grief. Upgrading a V6 application to V8 will be more time consuming.

Having worked in V7 and V8 for some time now, I find it very difficult to go back to the V6 SAS/AF build environment. The V8 environment is vastly superior in ease of use as well as the quality and appearance of your created frames.

Acknowledgements

I would like to extend sincere thanks to Glen Walker at the SAS Institute for the time, effort, and patience he extended helping with the preparation and review of this paper. Also, much thanks to Ann Rockett and Art Alexander at the SAS/AF technical support desk for the time spent helping to debug my applications.

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 developed in 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

SAS, SAS/AF, SAS/GIS, SAS/GRAPH, and SAS/SHARE are registered trademarks of the SAS Institute Inc., Cary, NC, USA.

ClinAccess is a registered trademark of MAJARO InfoSystems, Inc., Santa Clara, CA, USA.