Paper 7-28

Developing SAS/AF® Applications with Form Viewers and Table Viewers

Bernd E. Imken, Patented Medicine Prices Review Board, Ottawa, Canada
Steven A. Wilson, MAJARO InfoSystems, Inc., San Jose CA

Abstract

This paper will give SAS/AF developers the basic knowledge required to build an interface to a SAS® data set using the Form Viewer and the Table Viewer controls. These controls allow end users to view or edit a SAS data set using a graphical user interface (GUI).

To properly use these controls with SAS data sets, a SAS Data Set Model Component must be used to identify the SAS data set in use to the Form or Table Viewer. The communication between these components is referred to as Model/Viewer communication.

In this paper we will create frames that employ Model/Viewer communication to browse and edit data. This paper uses a step-by-step approach for developing frames with these components.

To make this document easier to view, we suggest that you try enlarging (Zoom) the PDF Adobe version available on the CD supplied at the conference or from the web site at http://www.sas.com/usergroups/sugi/proceedings/index.html.

Introduction

Among the standard set of SAS/AF controls are two controls that are specialized for viewing data in a SAS data set or view. The Form Viewer control provides the ability to interact with a single row of data in a SAS data set while the Table Viewer interacts with a data set in a tabular layout. These are the Version 8 upgrades for the Version 6 Data Form and Data Table objects.

Since these components employ Model/Viewer communication, this paper will provide an introduction to the Model/Viewer concepts as well as specifics of the SAS Data Set model component that is used with these components.

Overview of SAS/AF

Extensive overviews of SAS/AF have been given by the authors at previous SUGI conferences. See the list at the back of this paper for additional references. These papers give detailed descriptions of how to develop simple frames.

This paper will focus on the advanced development of SAS/AF applications using Form and Table Viewers. These viewers are among the most sophisticated and powerful visual Control Components available within SAS/AF.

Overview of Model/View Relationships

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

Models - identify, access, and manipulate data.
Viewers - display and collect the data.

Many controls are Model/View enabled. These components are ready-made to be linked with a model component.

Simple Model/View controls are the Combo Box, Radio Box, and the List Box. Imken's paper on Applications Made Easy showed how to populate a Radio Box Control using data as specified in the Variable Values List Model.

Similarly, the Form Viewer control provides the ability to view a single row of data in a SAS data set. The Table Viewer displays a data set in a tabular layout.

To create, or instantiate, a Form Viewer or a Table Viewer on your frame, drag the desired control from the Components window and drop it on the frame. Once on the frame, these controls may be moved or resized as needed.

Model components are placed on a frame using drag-and-drop, just like any other object. When associating a model with a viewer, drag the model component onto the frame and drop it on top of the desired viewer control.

Since model components are Model/View enabled by default, when you drop a model on a viewer, SAS automatically establishes the Model/View relationship. Model/View communication is defined when the **model** attribute of a viewer identifies an instantiated model component.

Although they are not visible on the frame display, the instantiated model component appears in the Properties window, where it may be selected, deleted, or similarly manipulated.

It is important that the model be dropped exactly on top of the viewer. Otherwise the **model** attribute is not automatically assigned, requiring you to manually assign a value to the **model** attribute of the viewer. For a standard arrow cursor style, the tip of the pointer should be on the viewer object when dropping the model.

Creating a Table Viewer

Let's create a frame with a Table Viewer that is used to browse the SASHELP.SHOES data set. Follow these steps to create this simple frame.

1) Open a new frame in a catalog named Sasuser.New.
2) Instantiate (i.e. create) and resize a Table Viewer on the frame.
3) Drag a SAS Data Set Model and drop it on top of the Table Viewer as shown in Figure 1. This links the model component with the viewer control.

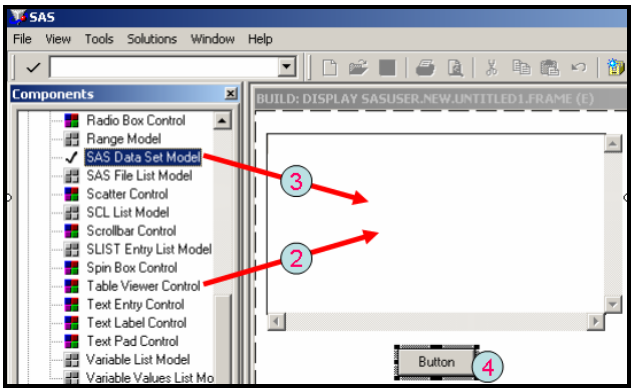
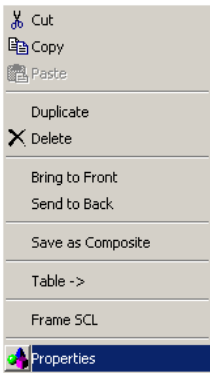


Figure 1 Creating the Table Viewer Control

4) Create a Push Button Control Component on your new frame below the Table Viewer.



5) Click the Right Mouse Button (RMB) on the frame and select "Properties" from the popmenu that appears. This opens the Properties window.

6) Verify that there is only one SAS Data Set Model component defined (see Figure 2) and that it is listed as the value of the viewer's **model** attribute. (called Sasdataset1)

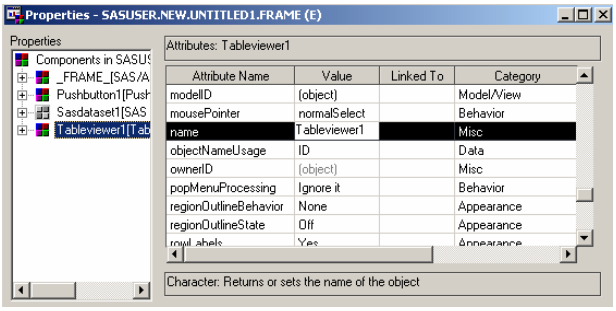


Figure 2 Modifying Viewer Properties

7) **Assign the following attributes:**
   a) **TableViewer1**
      i) name **table**
   b) **SASDataSet1**
      i) name **model**
      ii) table **sashelp.shoes**
      iii) wrapLabelText **Yes**
   c) **Pushbutton1**
      i) commandOnClick **OK**
      ii) label **OK**
8) **Close the Properties window. The specified SAS data set is now displayed in the Table Viewer control on the frame.**
9) **Adjust the width of the columns on the Table Viewer.**

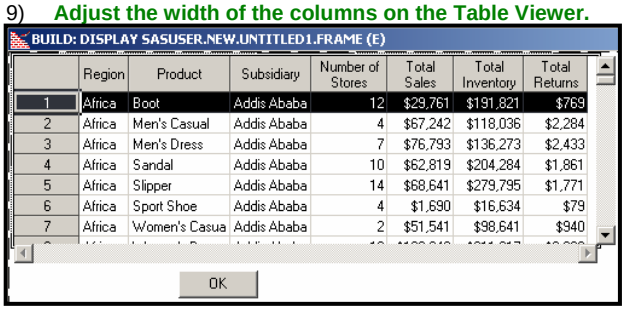
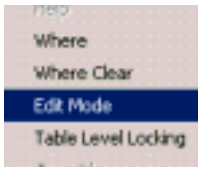


Figure 3 The Completed Table Viewer

10) **Adjust the position of the columns on the Table Viewer. To move a column, click the column header to highlight the column. Then drag the column to its desired location.**
11) **Save the frame by selecting File > Save from the pull-down menu and entering an Entry Name of tableViewer. The column position and width customizations are saved with the Table Viewer when the frame is saved.**
12) **Test your frame.**

During execution, use the RMB on the Table Viewer to obtain a popmenu of commands used to browse data via the Table Viewer.



The popmenu item “Edit Mode” allows you to toggle between “Browse Mode” and “Edit Mode”. The default value of the model attribute **editMode** is “Browse”. To prevent the user from entering the edit mode, set the model attribute

```
model.editMode = 'BrowseOnly';
```

Customizing the Table Viewer

Try assigning other attributes to the Table Viewer to modify the appearance of your frame. Attributes like **rowLabels**, **grid**, or **borderTitle** can substantially change the appearance of your Table Viewer. The SAS/AF developer’s environment makes it easy to modify a frame and quickly see the changes.

The **heldColumns** attribute is used to hold a column in place when scrolling left and right on the Table Viewer.

To define which columns and the order of columns displayed via the Table Viewer, use the **columns** and **columnOrder** attributes of the model. Yes, that’s correct – these are attributes of the model component!

Although the viewer attributes control how the data is displayed, these particular attributes are associated with the model because it is the model that is passing the data to the viewer. These attributes control which columns of the model data set and the order in which they are passed to the viewer.

The **columns** attribute controls a vast amount of functionality available when working with the SAS Data Set Model component. This single attribute defines column display attributes like font, color, justification, format, and uppercase, as well as min, max, and initial values, and required and protected settings. Computed columns can also be defined via this attribute. Dot notation can also be used to programmatically assign any **columns** value. For example:

```
model._getColumnNumber('NAME', colnum) ;
model.columns{colnum}.uppercase = 'No' ;
model.columns{colnum}.required  = 'Yes' ;
```

Selecting or Subsetting Data

In the example below we have created a frame containing a List Box and a Table Viewer.

A Variable Values List Model populates the list box with values of the PRODUCT variable in the data set SASHELP.SHOES.

The table viewer is the one we just created. The product column has been dragged to the left and the **rowLabels** attribute has been set to 'No' to improve the appearance.

When the frame first is displayed during execution, the list box does not have any items marked and all products are displayed in the Table Viewer.

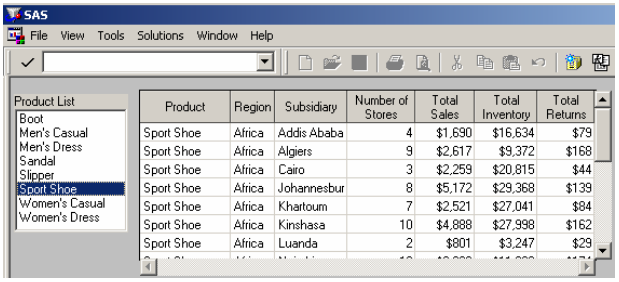


Figure 4 List Box Controlling a Table Viewer

The SCL code below is activated when a selection is made in the list box.

```
LISTBOX1:
  model.where =
   'product='||quote(listbox1.selecteditem);
return;
```

Clicking an item in the List box causes the **where** attribute for the model to be changed so that only the selected product is displayed.

A Table Viewer can be easily used as a selection list by setting a couple of viewer attributes and passing the selected data via a frame parameter.

```
dcl  char(14) selected_product ;
INIT:
  table.rowSelectionMode = 'RowLabelOrData' ;
  table.runObjectLabel   = 'SingleClick' ;
RETURN;

TABLE:
   model._getColumnText ( 'PRODUCT' ,
                selected_product ) ;
RETURN;
```

This code illustrates using the method **_getColumnText** to retrieve a character data value from the selected row on the Table Viewer. The method **_getColumnValue** is used to retrieve numeric data.

Creating a Form Viewer

Now develop a frame with a Form Viewer. Go back and follow the same steps detailed on the previous pages to create a frame named *formviewer* using a Form Viewer control named *form*. Next assign SASHELP.SHOES to the **table** attribute as below.

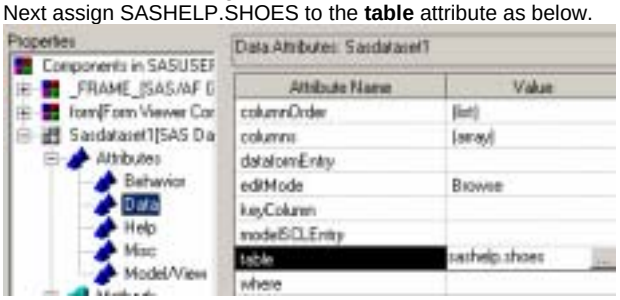


Figure 5 Creating a Form Viewer

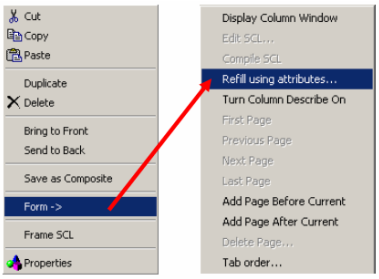
As soon as the table has been identified and you press the ENTER key you will notice that the Properties window changes considerably. The original components for *form* and *Sasdataset1* are still there but now there are also new ones for each variable in SASHELP.SHOES eg, Product, Region etc. In addition Text Label components have been generated for each variable.

Although each of these new components could be modified you will see in the next couple of paragraphs that there is an easier way of modifying Form Viewer components, rather than doing it at the frame level.

Let's first look at the results on the frame as shown in the diagram below. Notice that each of the variables in the SASHELP.SHOES data set is shown in the Form Viewer with the matching Label to the left of the Text Entry Control.

Try assigning attributes to the Form Viewer to modify its appearance. Notice that not all attribute changes are immediately reflected on the Form Viewer.

To apply attribute changes to the Form Viewer, click the RMB on the Form Viewer and select "Form →" to open the Form Viewer build-time popmenu.



Select the "Refill Using Attributes" option to re-draw the viewer using the current attribute values. Because of this necessary refill, have your form design well thought out before you position or otherwise manage the controls on the Form Viewer. You do not want to be in the position of having to refill your Form Viewer after you have invested a lot of time drawing the viewer

The concept of pages is used for Form Viewers in cases where a row has too many columns than can fit onto the viewer control. In this situation, the viewer displays the row using either 1) multiple viewer pages, or 2) one viewer page with scrollbars. This is specified via the **useSinglePage** viewer attribute.

Scrolling between pages can be accomplished using keystrokes or by using the mouse to click on the viewer earmarks. Using a single page with scrollbars will require the user to navigate the viewer with the mouse.

When creating a multi-page viewer display, you must cut-and-paste controls to move columns from one page to another.

Customizing the Form Viewer

Generally, in the first stage of customization you simply change the attributes of the variables displayed within the Form Viewer and then ask for the Form to be redesigned using the "Refill Using Attributes" as was discussed above. Remember every time you press "refill" the screen is redrawn.

This first stage of customization involves the model attribute **columns**. The value is set to {array}, indicating a special properties array is used to store elements of this attribute.

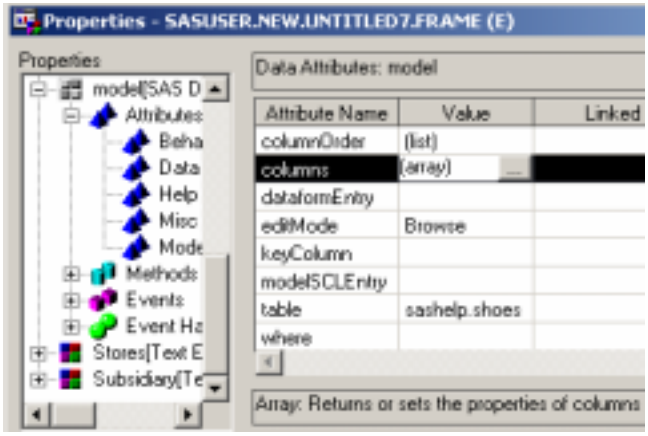


Figure 6 Accessing the "Columns" Atribute

To change any of the attributes of the variables being displayed in the Form Viewer simply click on the button to the right of {array} and the screen below will open up. Multiple columns can be modified simultaneously by pressing on Ctrl and holding it while clicking your column selections.

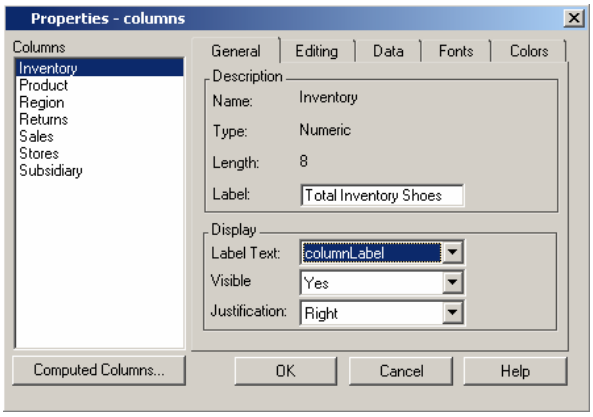


Figure 7 Modifying Column Properties

On the Columns Properties screen you will notice that the tabbed object on the right of the screen has the General Tab displayed allowing modifications of Label Text, Visible and Justification. The other Tabs are described at the top of the next page.

TAB	Attribute Type	Attribute Value
Editing	Values	Initial
		Minimum
		Maximum
	Formats	Format
		Informat
		Uppercase
Data	Options	Require Value
		Protect Value
	In Cell Editing	Cell Editor
		Cell Control
		Values List
Fonts	Fonts	Label
		Data
Colors	Background Colors	Label
		Data
		Error
	Foreground Colors	Label
		Data
		Error

You can change the appearance as is shown in the figure 8 below for Stage 1 Customization by modifying the attributes. Here the label for "Total Inventory" was replaced with "Total Inventory Shoes". If you had moved fields to different locations like was done in Stage 2 Customization and then had performed the refill you would have been returned to the alignment as shown in Stage 1. It is possible however to return to the **columns** attribute at any time to make modifications - just be careful with the "Refill Using Attributes" command.

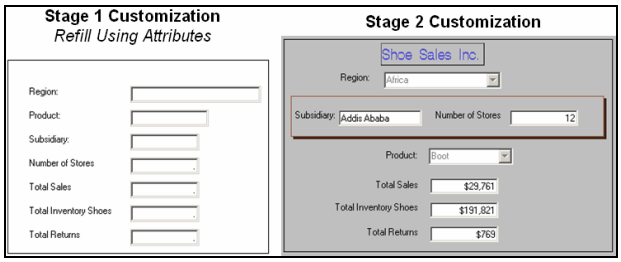


Figure 8 Further Form Viewer Customization

In Stage 2 Customization you can see that we have made a number of changes.

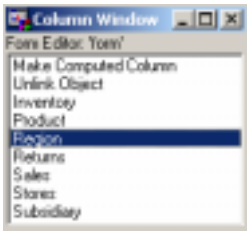
- A Graphic Text control was added to the top of the Form Viewer. The attributes for **color**, **borderStyle** and **font** were modified on this control.
- Subsidiary and Number of stores were positioned horizontally and a Container Box was placed around them.
- The Text Entry Controls for Region and Product were replaced by Combo Box Controls to improve data entry.
- The frame **backgroundColor** was modified to gray.

IMPORTANT: after making these modifications **NEVER** press the **"Refill Using Attributes"** selection or it will return to the Stage 1 level as shown on the left. It is useful to save your work periodically during development.

Many of the changes to the above Form Viewer were easily done by modifying attributes. One major area however is different. To change a variable's input/display field from a Text Entry Control to a Combo Box Control must be done in the following way. We will demonstrate the process by converting the PRODUCT field to a Combo Box.

1) Click on the Text Entry control for Product, click RMB and select DELETE to remove the existing Text Entry Control.

2) Drag the Combo Box Control from the Component window and drop it in the desired location.



3) Click RMB on the Form Viewer, on the Selection list displayed select "Form ->", on the next selection list click "Display Column Window", this will open up the Column Window as is shown on the left.

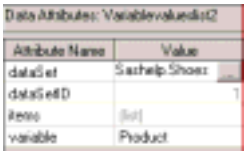
4) Click on **Product** in the list and drag it to the Form Viewer, dropping it over the newly created Combo Box Control.

5) This associates the new Combo Box with the variable for Product in the SAS data set.

6) To verify at any time that the field is associated with a variable in the data set, click RMB on the Form Viewer and again select "Form ->" , from the next selection list select "Turn Column Describe On", then click on the Combo Box you want to check and read the LOG message. If the association is properly done the message should read "The object 'Combo Box' is linked to column 'Product' ".

7) You will notice that if you test your application that the Combo Box initially is empty, that is because the region to be displayed is not yet available as data to the Combo Box Component. There are two ways to populate the Combo Box Control, both are shown below, the second approach is the most dynamic.

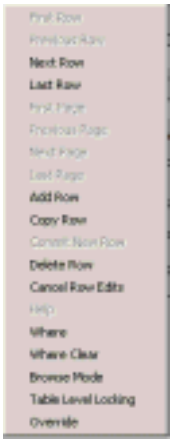
a) Enter the data manually by clicking on the **items** attribute and adding each item in turn. -or-



b) Click on the Variable Values List Model component and drop it over the Combo Box Control. Click RMB on the Form Viewer and select Properties. In the Properties window select the newly created variableValuesList model component. Enter the data set name and the variable to be used.

8) Return to the Form Viewer and perform another test, the Product values are now displayed.

While in test or production mode, at any time you can also click RMB on the Form Viewer and bring up the Form Viewer pop-up menu. This menu is displayed on the right. It is important to note that any processing which can be done by this pop-up menu can also be done within SCL code which can be added to your application. Using SCL can give you far more control than would be available with the pop-up menu. Using this menu you can move from Browse to Edit mode, add rows, and navigate throughout the data set being displayed in the Form Viewer.



It is a very simple process to change these frames to allow the user to edit the data instead of just browsing the data. As previously mentioned, this is controlled via the model attribute **editMode**.

This attribute can be set in the frame SCL by using dot notation:

```
model.editMode = 'rowLevelEdit' ;
```

The model attributes **allowAdditionDuplication** and **allowDeletion** define functionality of the model during execution of the frame when the **editMode** attribute permits editing.

The implications of allowing data edits are enormous. It now becomes critical to understand how and when data are read and written to the data set. The concepts of relative and absolute row numbers are also important to be able to work correctly with many model methods.

Adding Model SCL

Normally when we think of SCL in a SAS/AF application we think of the SCL program which is attached to the frame. The location of the SCL code attached to the frame is specified by the frame attribute **SCLEntry**.

Each model component also has its own SCL program. In order to add SCL to the model, you must first identify where the SCL will be stored. This is identified via the **modelSCLEntry** attribute of the model component, as shown in Figure 9.

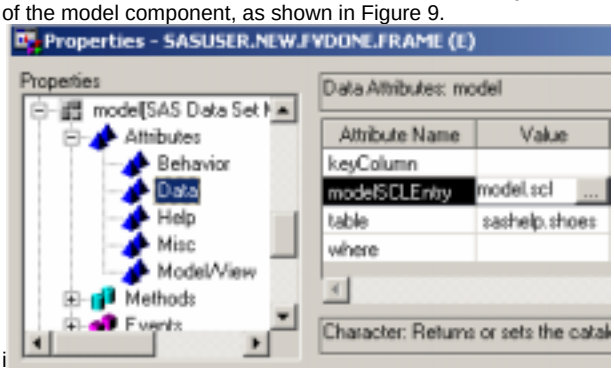


Figure 9 Adding Model SCL to the Viewer Component

Once the location has been identified you can add code by right clicking the form and then selecting "Form ->" and then "Edit SCL…" from the popmenu. The screen displayed should start off blank. It is important to call your frame SCL and your Model SCL by different names because they must be stored in separate locations.

Model SCL is commonly used to perform actions when data is read or saved as the user interacts with the data set via the viewer. Values in the data set and computed column values can be assigned via Model SCL.

Model SCL is *absolutely* required when you wish to execute code when moving between rows in a data set. This is because the PageUp and PageDown keys will scroll between rows in the data set when the viewer is the active control. The frame SCL cannot determine when the PageUp and PageDown keys are used to move among rows. This is a unique consideration since all other means for scrolling between rows can be identified in the frame SCL.

Since the model executes the Model SCL, a separate execution environment exists for the Model SCL. Variables that exist in the frame SCL cannot be referenced by the Model SCL and vice-versa. However, you do have the ability to reference any object on the frame or on the viewer by obtaining and using the component identifier within the Model SCL.

These statements illustrate how to obtain useful component identifiers when writing Model SCL:

- `_viewer_`
  This automatic variable contains the component ID of the viewer control.
- `DCL object modelid ;`
  `modelid = _viewer_.modelid ;`
  Obtains the component ID of the Model. Model attributes and methods may be referenced from within the Model SCL.
- `DCL object id ;`
  `_viewer_._getComponentID ( 'PRODUCT', id );`
  Obtains the component ID of a component on a Form Viewer. Attributes and methods of components on the Form Viewer may be referenced from within the Model SCL.
- `DCL object frameid ;`
  `frameid = _viewer_.frameid ;`
  Obtains the ID of the frame. You can assign frame attributes within the Model SCL.
- `DCL object controlid ;`
  `frameid._getWidget('pushButton1',controlid);`
  Obtains the ID of a control component on the frame, allowing you to manage control components that are not on the viewer. This cannot be used for model components.

Model SCL has labeled sections of code that are executed automatically. These special label sections are:

Section Heading	Usage
DFINIT:	This section of code executes when the frame is opened and the viewer is initialized. This executes after the frame INIT code has executed.
INIT:	Executes when a row in the data set described by the model is displayed. With a Table Viewer, INIT runs for each displayed row when the viewer is populated, when scrolling, or when a row is locked.
column_name:	Labeled sections that correspond to variables in the data set described by the model are executed when the value of the variable is modified via the viewer. This section does not execute if the value is changed programmatically via SCL.
MAIN:	This executes when any data in the data set is modified, prior to the execution of any frame SCL.
TERM:	This executes as the user leaves the current row only when in edit mode.
DFTERM:	This final section of code executes when the frame is closed, prior to the Frame TERM section.

Computed Columns

A computed column is a calculated value that does not exist in the SAS data set, but is instead computed and displayed on the Form Viewer during execution. To define a computed column:

1) Drag a Text Entry control that will display the computed value from the Components window and drop it onto the Form Viewer. This may be some other control, such as a Combo Box control.

2) Open the Properties window and assign the attribute name to the control. This will be the name of the computed column. Also define the data type.
   a) TextEntry1
      i) name newvalue
      ii) dataType Numeric

3) Drag the item "Make Computed Column" from the Column window and drop it exactly on top of the new control. This allows the new control on the viewer to be able to display a computed value.

4) Edit and compile the Model SCL to assign a value to the computed column as needed. This example computation

would be needed when a row is initially displayed and also when the value of the Sales column is modified:

```
INIT:
SALES:
      newvalue = sales * 100 ;
RETURN ;
```

Let's look at another example. We want to create a push button control and a Text Entry component on the Form Viewer such that when we push the new button the Text Entry field should get set to today's date.

1) Create the two new control components on the Form Viewer. If the Text Entry component represents a SAS variable "Date" which has been added to the data set behind the Form Viewer, you must first return to the Column window and drag date from this window and drop it exactly over the new Text Entry component.

2) Click "Make Computed Column" in the Column window, drag it and drop it on top of pushbutton1. You will notice that pushbutton1 is added to the list in the Column window.

3) Add the following code to the Model SCL. This assigns the button label in the INIT section and executes the labeled section of code when the button is clicked. The model must be in edit mode to allow clicking the Push Button on the Form Viewer.

```
INIT:
   pushbutton1 = 'Date' ;
RETURN ;
PUSHBUTTON1:
   date=today();
RETURN ;
```

Remember to compile the Model SCL code. When you next go into test mode, you must first enter EDIT mode, this can be done by clicking RMB and bringing up the form viewer's popmenu. Of course, you can also assign the edit mode programmatically:


```
model.editMode = 'Edit';
```

Once in edit mode click the date button and see the desired results. Notice that the SAS data set variable name is the same as the SCL variable. When we set the `date=today()`, we are setting the SAS data set variable date which is linked to the date Text Entry Component. Do not set `date.text=today()` because SAS data set variables in Model SCL are not referenced using dot notation.

As a final note regarding development of Model SCL, it can be convenient to automatically compile the Model SCL when the frame is opened. To accomplish this, place the following statement in the INIT section of the frame SCL:

```
model._setSource('MODEL.SCL','Y');
```

Component Methods

To be precise, a method is a labeled section of SCL that 1) is automatically invoked by an EventHandler in response to an Event, or 2) executes when the method is explicitly called by a program. For the sake of simplicity, think of a method as an action performed by a component.

The following tables list methods that are commonly used with these Model/View components. The SAS Data Set Model methods are the methods that are executed by the default run-time viewer popmenu.

Model Methods	SAS Data Set Model
_addRow	X
_commitNewRow	X
_copyRow	X
_deleteRow	X
_getColumnText (C) _getColumnValue (N)	X X
_getDatasetAttribute	X
_getRowNumber	X
_getRowStatus	X
_gotoAbsoluteRow	X
_rereadCurrentRow	X
_rereadDisplayedRows	X
_setColumnText (C) _setColumnValue (N)	X X
_setWhere	X

Viewer Methods	Table Viewer	Form Viewer
_getActiveCell _clearActiveCell	X X	
_getSelect _clearSelect	X X	
_gotoRow	X	X
_printPreview	X	
_selectRow	X	
_gotoCell	X	
_getComponentID		X
_getComponents		X
_gotoRowNumber		X

Adding Frame Controls to Communicate with a Viewer

In the example below you can see that 10 push button control components have been added to the top of the frame, giving the user control of the Form Viewer below. The user can easily page backwards and forwards, just by pushing buttons similar to a video player. He or she can also add new records to the SAS data set and perform other operations such as delete, save and cancel. Notice that the push button is on the frame and not within the form viewer. Let's look at what is involved in the SCL to tell a push button to ADD a new record.

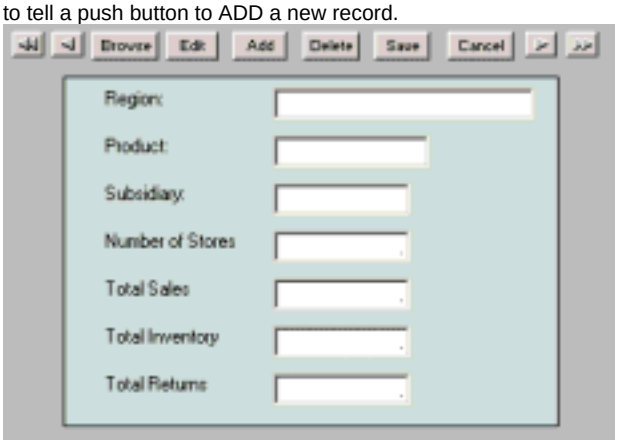


Figure 10 Adding Push Buttons to Drive the Viewer

Create a push button outside of the Form Viewer, then add the following code to the frame SCL. It executes when the push button is pressed.

```
PushButtonADD:
 Sasdataset1._addRow();
Return;
```

Note that the _addRow method does not provide a return code, nor a system return code. It is difficult to determine when the _addRow method fails. This method will fail when the current row is missing required values, violates an integrity constraint, violates a unique index, or other such event.

It is also useful to add an additional button to save the input once the user has finished (SAS® will save automatically if another row is added or if you exit out of add mode.) A save push button would have the following code in the frame.

```
Sasdataset1._commitNewRow();
```

Please note that the methods _addrow and _commitNewRow are applied to the SAS Data Set Model and not to the Form

Viewer. If you wanted to cancel the input and return to previous values, use the following method.

```
Sasdataset1._rereadCurrentRow();
```

Navigating within a Form or Table Viewer

To go directly to a specific row number (example: observation 50) issue the following method to the Form Viewer;:

```
formviewer1._gotoRowNumber(50);
```

Substitute: -1 for first row -2 for previous row
-3 for next row -4 for last row

The _vscroll method is also supported, see SAS Help.

Note that if you place your cursor within the formviewer while in run mode, you can use the PgUp and PgDn keys – even faster!

Starting an application on the most recent observation is also done this way, only we added the _refresh() method for it to work properly.

```
INIT:
   _frame_._refresh();
   formviewer1.gotoRowNumber(-4);
return;
```

Form Viewer Controls to Communicate with a Nested Table Viewer

If you want to create a pushbutton on a Form Viewer such that when it is pushed it would ADD a record to the Table Viewer matching the key value (eg. Region) currently in the Form Viewer. Insert SCL code similar to what is below into the SCL for your Form Viewer. The "DFINIT" section identifies the Table Viewer to the Form Viewer. Once the ADD pushbutton is pressed the value of the key column is updated and the addRow method is executed.

```
DFINIT: dcl object tableid modelid;
  _viewer_._getComponentID('tableviewer1',
    tableid);
 modelid=tableid.modelid;
return;

INIT:
  pushbutton1='ADD';
return;

PUSHBUTTON1:
  modelid._getColumnNumber
            ('Region',colnum);
  modelid.columns{colnum}.initialValue.
        numericValue=Region;
 /* note:above 2 lines is really 1 line
         - no spaces*/
  modelid._addRow();
return;
```

Setting pushbutton1 to "ADD" makes the word "ADD" visible on the button.

Adding a Title to the Top of the Frame

We suggest you try adding the following to the frame SCL.

```
MAIN:
 sasdataset1._getRowNumber(currow);
 if (currow=0) then _frame_.title='No Rows';
 return;
```

Add this to the model SCL:

```
dcl list infolst={}, list datalst={},
            object frameid;
DFINIT:
 frameid=_viewer_.frameid;
 return;
INIT:
 _self_._getRowStatus(infolst);
 new=getnitemc(infolst,'new');
 if new='Y' then
  frameid.title='Pending Row';
 else do;
   _self_._getRowNumber(currow);
   _self_._getMaxRow(maxrow);
   if maxrow > 0 then
  do; _self_._getDatasetAttributes(datalst);
numdel=getnitemn(datalst,'number_of_deleted_
rows');
  if numdel > 0 then maxrow=numdel+maxrow;
   frameid.title='Row '||currow||' of '
||maxrow||' rows';
  end;
   else do;
     _self_._getMaxRow(maxrow,'y');
     frameid.title='Subset mode '
       ||maxrow||' rows found';
    end;
  end;
return;
DFTERM:
 rc=dellist(infolst);  rc=dellist(datalst);
return;
```

Using this SCL will generate a different title when the data set is being subset. The value of MAXROW < 1 when the data is being subset.

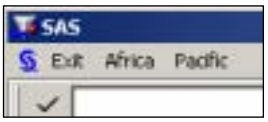
Issuing Commands from the SAS Command Box or Line

Another useful technique in communicating with a form viewer is shown in the Frame SCL below. In this example we have a form viewer called Formviewer1 on the frame. We wish to issue command such as "ADD" from the SAS Command Box and have the viewer add a new record automatically without having to have code for the _addrow() method. The code below makes this possible - the _execcmd() method send the command directly to the SAS data set model as requested.

```
MAIN:
 Control always allcmds;
 word=word(1,'u');
 select(word) ;
   when ('BACKWARD') DO;
     Formviewer1._VSCROLL('row',-1);
     call nextcmd();end;
   when ('FORWARD') DO;
     Formviewer1._VSCROLL('row',+1);
     call nextcmd();end;
   otherwise;
     If word ne _blank_ then
          Sasdataset1._execcmd(); end;
return;
```

Notice in the SCL code we have also added the FORWARD and BACKWARD commands and have told SAS to page ahead or back when these are submitted. (note, we could also have used the **_gotorowNumber()** method in place of the **_vscroll()** method. We have found this approach particularly useful in applications allowing us to send messages from a PMENU entry which could then be interpreted at the frame level. The code below generates a pmenu which can be used on the Frame by changing the Frame's pmenuentry attribute to "subset.pmenu".

```
proc pmenu c=work.sugi28;
  MENU SUBSET;
   item 'Exit' selection=ex mnemonic='E';
   item 'Africa' selection=af mnemonic='A';
   item 'Pacific' selection=pa mnemonic='P';
      selection ex 'END';
      selection af 'AFRICA';
      selection pa 'PACIFIC';
run;
```



The Figure to the right shows the pmenu generated at thew top of the frame. When the selection is made from the menu one of the words, END, AFRICA or PACIFIC will be sent to the command line. The Frame code must now be modified to be able to pick up these commands. The Frame code shown earlier must now be changed to pick up what will happen for the words AFRICA and PACIFIC - in this case the sasdataset model will be subset for the selection. Note that the word END will pass the "end" command to the Frame and will therefore be picked up by the _execcmd() method.

```
MAIN:
 Control always allcmds;
 word=word(1,'u');
 select(word) ;
   when ('AFRICA') DO; call nextcmd() ;
    sasdataset1.where="region = 'Africa'";
   when ('PACIFIC') DO; call nextcmd() ;
    sasdataset1.where="region = 'Pacific'";
   when ('BACKWARD') DO;
     Formviewer1._VSCROLL('row',-1);
     call nextcmd();end;
  otherwise;
     If word ne _blank_ then
          Sasdataset1._execcmd(); end;
return;
```

Developing Master/Detail Applications

Frequently we have applications which have Master/Detail Relationships, these can be easily done using SAS/AF. In the example below we have created a Form Viewer on a Frame containing three major components a) a Text Entry for Region, b) a Text Entry for a head office address and c) a Table Viewer, created from SASHELP.SHOES. The table viewer in this case is nested within the Form Viewer

The example below shows that when we page down on the master Form Viewer and encounter the "United States" region, only the records which match this region are displayed in the Detail Table Viewer**.** All this can be done in SAS/AF without having to write any SCL code.

Figure 11 shows a simple example of a Master/Detail relationship between a Form Viewer for region with a head office address and a Table Viewer for the SASHELP.SHOES data set displaying region linked using a one to many relationship.

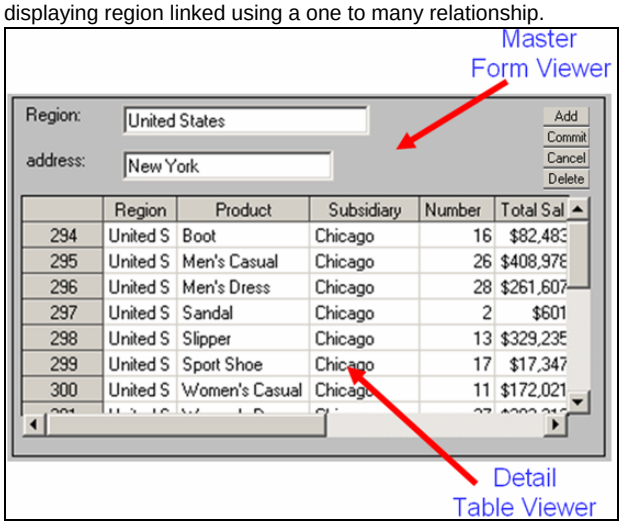


Figure 11 Nesting a Table Viewer Within a Form Viewer

To link a Master Form Viewer to a Detail Table Viewer:

1. click RMB **on the Form Viewer** then select "Form ->", then select "Display column window"
2. click on the "Region" in the "Display Column Window" and drag it over the "Region" column of the Table Viewer
3. set the **keyColumn** attribute to "Region" for the SAS date set model associated with the Table Viewer.

You can readily see how powerful this can make applications in SAS/AF. Also at the top right of the Form Viewer you can see 4 Push Button controls which allow you to add or delete records etc. from the linked Table Viewer. The following SCL code must be added to the Form Viewer SCL to make it work. (please note that the 3rd line with "modelid.columns…" wraps to the line below and should be typed as one continuous expression.

```
ADD:
  modelid._getColumnNumber('region',region);
  modelid.columns{region}.
      initialValue.characterValue=region;
  modelid._addRow();
return;

DELETE:
  modelid._deleteRow();
return;

COMMIT:
  modelid._commitNewRow();
return;

CANCEL:
  modelid._rereaddisplayedRows();
return;
```

Adjusting Column Width within a Nested Table Viewer.

If the Table Viewer is nested within a Form Viewer, you will find that you cannot directly adjust the column widths by dragging the label border as can be done when a Table Viewer is not nested. One trick is to cut the Table Viewer and paste it outside of the Form Viewer. Now adjust your column widths. Once completed, cut the adjusted Table Viewer and paste it back into the Form Viewer. (Note: it is wise to take a duplicate copy of your entire frame first, in case you do something incorrectly so as not to lose your original.)

Nesting a Detail Form Viewer within a Master Form Viewer

There is a slightly different procedure for nesting one Form Viewer within another.

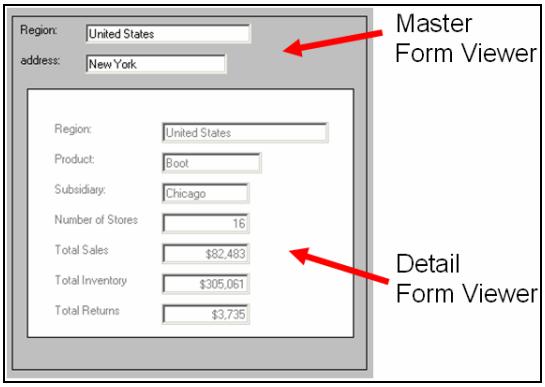


Figure 12 Nesting Form Viewers

1) You must first click in the main Form Viewer (formviewer1) to activate it.

2) To complete the linkage you must type the following command on the command box: **"LINK FORMVIEWER2 REGION"** and press enter– here formviewer2 is the name of the nested Viewer and REGION is the key column of both the models.

Linking Viewers Externally

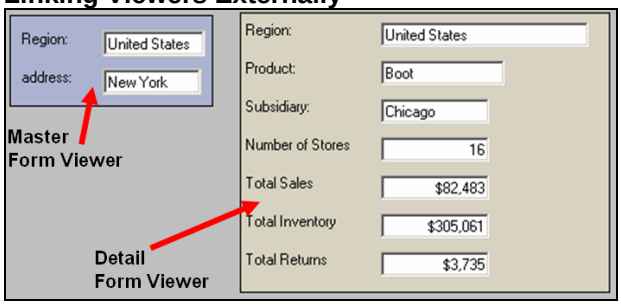


Figure 13 External Linkages of 2 Different Viewers

Creating a Form or Table Viewer externally on the same frame (as seen in Figure 13). To do this it is necessary to write some SCL code. Two approaches can be used.

Both approaches are shown since there are instances when it is more convenient to use one rather than the other.

Approach 1: The code could be written solely at the Form Viewer level. No special programming is required at the frame level. First the **"ModelSCL"** attribute must be changed on the Form Viewer to point to SCL for the viewer. The code for the Viewer is show below:

```
DFINIT:
  dcl object formviewer2 modelid;
  frameid=getnitemn(_viewer_,'_frame_');
  call send(frameid,'_getWidget',
       'formviewer2',formviewer2);
  modelid=formviewer2.modelid;
  vals=makelist();
  return;
INIT:
  link region;
  return;
REGION:
  rc=clearlist(vals);
  rc=insertc(vals,region,-1,'REGION');
  rc=modelid._setKey('region',
           'EQ','SCROLL',vals);
  return;
DFTERM:
  vals=dellist(vals);
  return;
```

This approach identifies the second fornviewer to the first in the DFINIT section and then uses the "_setKey" method to link the second Form Viewer to the first each time the "mialno" changes.

Approach 2:

This approach requires the follow code at the frame level.

```
INIT:
  control always;
  formviewer1._setEventHandler
         (formviewer1,'FV_ROW_CHANGED',
          '_set_key_',formviewer2);
  sasdataset1._rereadCurrentRow();
  return;
```

Then at the Form Viewer level the following ModelSCL is required:

```
DFINIT:
  vals=makelist();
  return;
INIT:
  link region;
  return;
REGION:
  rc=clearlist(vals);
  rc=insertc(vals,region,-1,'REGION');
  _viewer_._sendEvent('FV_ROW_CHANGED',
     'REGION','EQ','SCROLL',vals);
  return;
DFTERM:
  vals=dellist(vals);
  return;
```

Moving Controls within a Nested Form Viewer.

The developers at SAS looked into this and found that it was only a problem on PC platforms. It had to do with a conflict with cursor tracking and drag and drop status and they couldn't find a solution that wouldn't cause them to turn off some functionality that was needed so they were not able to change the behavior.

A circumvention that might help you position your columns a little better than when you cut and paste is to select the column, then select the Layout pmenu item and then the Move pmenu item. This gives you the outline of the column that you can move around and position as you would like. This is much easier to position than when you do a paste action and aren't sure exactly where your pasted column is going to wind up.

Controlling Component Attributes Within a Viewer While the Viewer is Executing

Occasionally one would like to have change the attributes of a component within a Form Viewer or Table Viewer from within the SCL of the viewer. In the example shown in Figure 14, you will notice the EMPLOYEE data set displayed on the left of the figure and a Form Viewer displayed on the right side of the Figure. Initially the fields for name and Salary are displayed with a white background color. When the Form Viewer encounters the record corresponding to "SMITH" the component for salary become PROTECTED and the background color get set to gray.

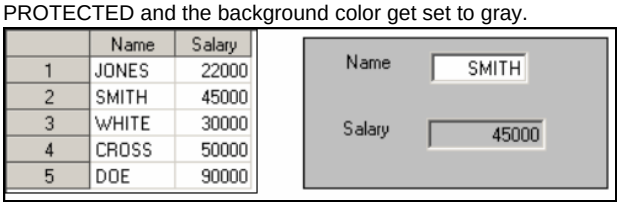


Figure 14 Setting an attribute from within the Viewer

To make this behavior work it requires that we add SCL code to the Form Viewer. Notice below that this is done using the _setViewerAttribute().

```
INIT:
  link NAME;
return;

NAME:
  if name = 'SMITH' then do;
    _self_._setViewerAttribute
            ('salary','protect','N');
    _self_._setViewerAttribute
            ('salary','bcolor','gray');
  end;
  else  do;
    _self_._setViewerAttribute
           ('salary','protect','Y');
    _self_._setViewerAttribute
           ('salary','bcolor','white');
    end;
return;
```

Also notice that the Form Viewer can be called using _self_ while you are within the Viewer SCL. There is another method of accomplishing the same thing. By using the _getComponentID() method it is possible to the object salary to the Form Viewer such that dot notation can be used as shown below.

```
DCL object salary;
DFINIT:
  _viewer_._getComponentID
       ('salary',salary);
return;
```

```
INIT:
   _self_._setViewerAttribute
         ('salary','protect','Y');
return;

NAME:
  if name = 'SMITH' then do;
   _self_._setViewerAttribute
        ('salary','protect','N');
   salary.backgroundColor='yellow';
   end;
return;
```

Please notice that the attribute names which are used with the _setViewerAttribute are different from those used fir dot notation. A listing of those which can be used is shown below.

Attribute Name	Description	Form Viewer	Table Viewer
BCOLOR	Background Color	X	X
FCOLOR	Foreground Color	X	X
BPATTERN	Background Pattern	X	
BDRCOLOR	Border Color	X	
FONT	Font	X	X
HJUST	Horizontal Justification	X	X
VJUST	Vertical Justification	X	
LTSOURCE	Light Source	X	
MARGIN	Margin	X	
REVERSE	Reverse Video	X	
PROTECT	Protection	X	X

Implementing an Application

You could use the –initcmd config option to specify a command that would invoke a SAS/AF application when a SAS session is started:

```
-initcmd "afa c=prod.sys1.opening.frame"
```

This method is suggested rather than embedding the AF command within the AUTOEXEC.SAS file because it gives the developer control over opening the Editor window etc.

Conclusion

Form and Table Viewers can be complicated components to employ in a SAS/AF application. However, the alternative options are to 1) use SAS/FSP procedures, which appear quite crude by today's standards, or 2) develop your own SCL to duplicate the functionality provided by these components, which is a daunting task.

Simple functionality can be easily accomplished with these components, but moving beyond the basics may be more challenging. To utilize these components effectively you are advised to gain a thorough knowledge of their attributes and methods.

Once you become familiar with the component attributes, many of which are common, you will quickly discover that you can develop such applications very easily. It is advisable during development that you do not add too many components to a viewer. Viewers which contain a large number of controls tend to execute slower. Keeping it simple makes development and maintenance easier, as well as improves response time later in production.

This paper has given a substantial overview of the SAS/AF Model/View solutions for interacting with a SAS data set and will allow you to develop much functionality using these components.

Further Reading

Other papers published by the authors on this subject.

Imken, Bernd	
Developing SAS/AF Applications Made Easy	***28th Sas Users Group International Proceedings***
Developing Master/Detail Applications Using Form & Table Viewers	***27th Sas Users Group International Proceedings***
A SAS Based Correspondence Management System	***26th Sas Users Group International Proceedings***
The Use of Data Forms, Tables and Other new Objects in SAS/AF	***24th Sas Users Group International Proceedings***
Improving SAS Data Access - An Evolutionary Experience	***24th Sas Users Group International Proceedings***

Wilson, Steve	
Getting Started with SAS/AF Software	***27th Sas Users Group International Proceedings***
Using Table Viewer and Form Viewer Controls in SAS/AF	*Western Users of SAS Software 2001 Conference*

Acknowledgements

The authors extend sincere thanks to Ann Rockett and Lynn Curley at SAS Technical Support for their assistance as well as the countless solutions they have provided. It has not always been easy doing SAS/AF development, but their patience and responsiveness has been greatly appreciated.

About the Authors

Bernd Imken is of Chief, Information Systems for the Patented Medicine Prices Review Board, an agency of the Government of Canada. He has also worked in senior IS/IT positions for the Export Development Corporation, taught at Queen's University, was a Board Member of Elrond College and was President of the Queen's University Faculty Club.

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.

Both Bernd and Steve have been developing applications in SAS for nearly 20 years and have given numerous presentations at SUGI as well as regional and local conferences. Steve is a Version 8 SAS Certified Professional and was selected as SAS Programmer of the Year 2001 by the Bay Area SAS Users Group. In 2002, Bernd was presented the "Award of Excellence" by the Ottawa office of SAS Institute.

Bernd E. Imken
Patented Medicine Prices Review Board
333 Laurier Ave West
Ottawa, Ontario, Canada K1P 1C1
Email: bimken@pmprb-cepmb.gc.ca

Steve Wilson
MAJARO InfoSystems, Inc.
1731 Technology Drive
Suite 560
San Jose, CA 95110-1331
Email: SWilson@majaro.com