Paper 008-31

# SAS/AF® Software Rises Again – Enhancements in SAS® 9.2

Lynn Curley, SAS Institute Inc., Cary, NC

## ABSTRACT

Is there a future for SAS/AF software? Absolutely! We have heard from many customers that SAS/AF is particularly useful for the development of interactive applications that can be executed with as little as an installation of Base SAS. In response to this feedback, SAS wants to demonstrate its support for SAS/AF, ensuring that SAS/AF will continue to be a useful product for many years to come. Accordingly, SAS/AF and the VIEWTABLE window are being enhanced to include support of long format names. Two classes with experimental status, the Dual Selector Control and the Tree View Control, are being enhanced and promoted to production status. This paper discusses the exciting enhancements to the classes and demonstrates how to use them.

## INTRODUCTION

Several years ago, SAS redirected development resources away from SAS/AF and toward the environment based in Java that is offered through SAS AppDev Studio. At the same time, SAS encouraged its customers to move in this direction. This migration led some SAS customers to question the future of SAS/AF. Although SAS is still committed to technologies that are based in Java, SAS recognizes that many of its customers want and need the rich object-oriented programming environment that SAS/AF offers. Through its drag-and-drop graphical user interface (GUI) builder, SAS/AF provides an easy-to-use application development environment that can produce applications that can be executed with as little as an installation of Base SAS. Furthermore, SAS Component Language (SCL), the scripting language behind SAS/AF, supports the SCL list data structure and can function as the scripting language for Web-based applications that are deployed through SAS/IntrNet software.

SAS wants to demonstrate its commitment to SAS/AF by:

- Including in the 2006 SASware ballot, after a five-year hiatus, features that are suggested by our customers for SAS/AF, SCL, and the VIEWTABLE window.
- Publishing a new book, *Getting Started with the SAS/AF Development Environment and FRAME Entries.* This publication, which is soon scheduled for release, is a task-oriented primer that is targeted for SAS programmers who are new to the SAS/AF development environment.
- Continually adding new samples for SAS/AF and SCL to the SAS Samples that are published on the SAS Customer Support Center Web site (support.sas.com). The samples that we add are based on questions that are asked by customers who contact SAS Technical Support.
- Enhancing SAS/AF with:
  - Support for long format and informat names. This enhancement has also been added for SAS/FSP software and the VIEWTABLE window.
  - Documentation of the V6GUIMODE system option for SAS on the mainframe.
  - Promotion of two experimental classes - Dual Selector Control and Tree View Control - to production status.

This paper discusses each software enhancement.

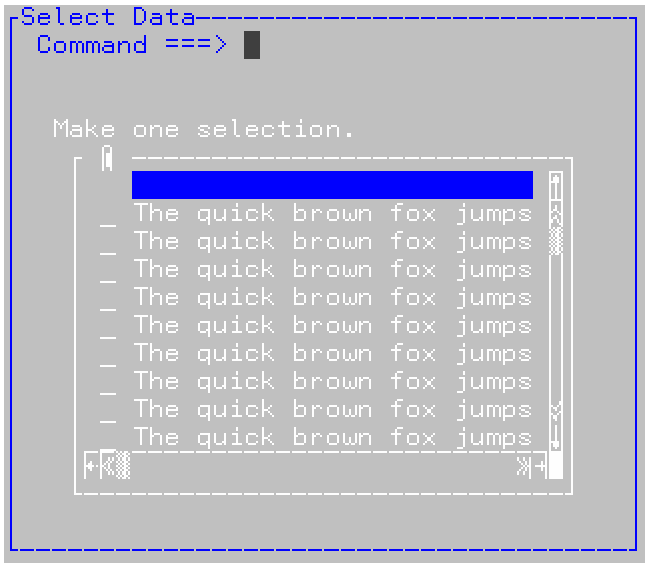
## LONG FORMAT AND INFORMAT NAMES

Although SAS®9 supports names for user-defined formats and informats that are longer than eight characters, SAS/AF, SAS/FSP, and the VIEWTABLE window have not made use of this feature. Up to now, because the long names were not supported, some data have been displayed unformatted, messages about the missing format or informat were not written to the SAS Log, and, in some cases, such as with the VARFMT function in SCL, format and informat names were truncated to eight characters.

Starting with SAS 9.2, SAS/AF, SAS/FSP, and the VIEWTABLE window will support long format and informat names. Accordingly, data displayed through the VIEWTABLE window will be formatted, and SAS/AF methods and SCL functions will no longer truncate format and informat names.

## V6GUIMODE SYSTEM OPTION IN THE MAINFRAME ENVIRONMENT

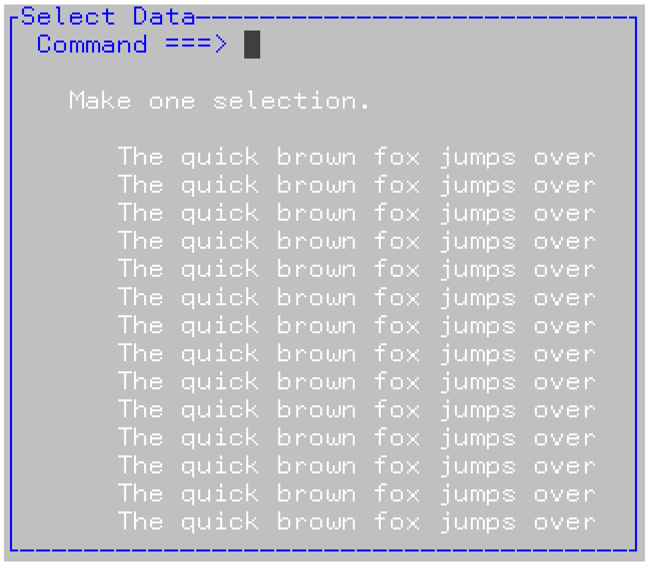
With SAS 8, customers using SAS/AF in a mainframe environment experienced difficulties with the BUILD Explorer window. Also, customers using SAS/AF or SAS/FSP saw changes with the selection list windows, such as the one displayed by the DATALISTC|N function in SCL, as shown in Figure 1.



```
Select Data
 Command ===>

  Make one selection.

   _ The quick brown fox jumps
   _ The quick brown fox jumps
   _ The quick brown fox jumps
   _ The quick brown fox jumps
   _ The quick brown fox jumps
   _ The quick brown fox jumps
   _ The quick brown fox jumps
   _ The quick brown fox jumps
     The quick brown fox jumps
```

**Figure 1. Selection List Window**

By specifying the V6GUIMODE option when starting SAS, mainframe customers can use the SAS 6 style of selection list windows. These selection list windows include those that are displayed via the SCL functions: DATALISTC|N, SHOWLIST, LISTC|N, VARLIST, DIRLIST, CATLIST, FILELIST, and LIBLIST. The V6GUIMODE option displays the SAS 6 style of the POPMENU function when the list of items is displayed in a scrollable list box, and it displays the SAS 6 Directory window for the BUILD procedure instead of an Explorer window. The V6GUIMODE option gives you a slightly larger window. Figure 2 displays the same selection list that is shown in Figure 1, but this list uses the V6GUIMODE option.



```
Select Data
 Command ===>

   Make one selection.

      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
      The quick brown fox jumps over
```

**Figure 2. Selection List Window with V6GUIMODE**

The following command starts SAS in the mainframe environment with the V6GUIMODE option specified. This example assumes that the *clist* that is used to start SAS is called *SAS9*.

```
      sas9 o('v6guimode')
```

Although support for the V6GUIMODE option began with SAS 8 (and many mainframe customers have taken advantage of it since that time), the option has not been documented or listed in output that is created by the OPTIONS procedure. Starting with SAS 9.2, the V6GUIMODE option will be documented.

## PROMOTION OF EXPERIMENTAL SAS/AF CLASSES TO PRODUCTION STATUS

Version 8 of SAS introduced a new class library that included a number of visual controls. Several of these controls, including the Dual Selector Control and the Tree View Control, were released with an experimental status. Although both classes were used in production SAS software, they were given the experimental status because their functionality was considered incomplete. In response to SAS customers' requests, these two classes are being enhanced and promoted to production status in SAS 9.2.

### THE DUAL SELECTOR CONTROL

The Dual Selector Control enables you to easily display and move items between and within two List Box Controls. Used in the VIEWTABLE window, a Dual Selector Control defines the column(s) to be hidden or displayed. The Dual Selector Control is a composite class comprised of two List Box Controls and several Push Button Controls.

#### Creating a Dual Selector the Hard Way

To demonstrate the convenience that the Dual Selector Control offers, consider the following FRAME entry that contains a dual selector made from two List Box Controls and four Push Button Controls (Figure 3). This dual selector displays items and supports moving them between two List Box Controls.

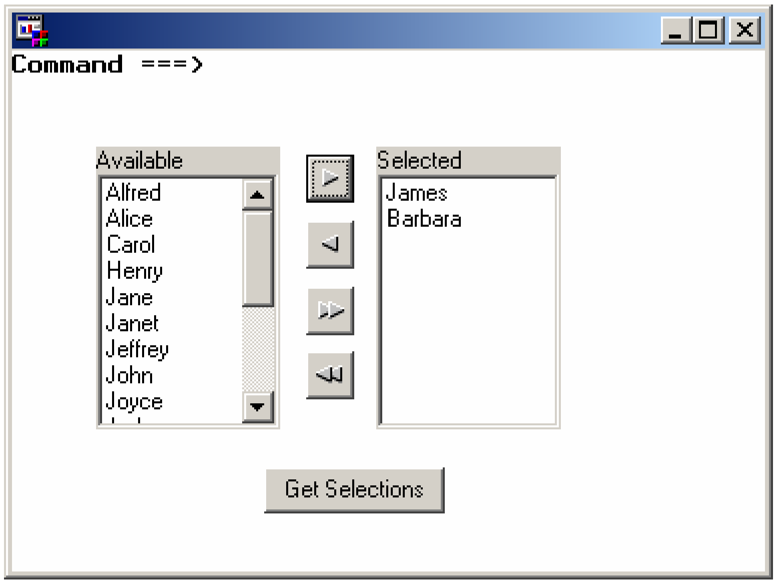


**Figure 3. Dual Selector Created Manually**

The SCL for the FRAME entry provides the functionality:

```
   /* Create SCL lists to populate the List Box Controls. */
   dcl char(8) selectedItem;
   dcl list itemsList = {},
       list emptyList = {},
       list tempList1 = {},
       list tempList2 = {};

   INIT:
      /* Populate the left-hand list box. */
      dsid = open('sashelp.class');
      nlevels = 0;
      rc = lvarlevel(dsid, 'name', nlevels, itemsList);
      if dsid then dsid = close(dsid);
      itemsList = revlist(itemsList);
      listBox1.items = copylist(itemsList);
    return;

   SINGLERIGHTARROW:
      /* Move one item from the left-hand list box to the right-hand list box. */
      tempList1 = copylist(listbox1.items);
      tempList2 = copylist(listbox2.items);
      numberSelections = listlen(listbox1.selectedItems);
      do i = 1 to numberSelections;
         selectedItem = getitemc(listBox1.selectedItems, i);
         rc = insertc(tempList2, selectedItem, -1);
         index = searchc(tempList1, selectedItem);
         rc = delitem(tempList1, index);
      end;
      listBox1.items = copylist(tempList1);
      listBox2.items = copylist(tempList2);
    return;

   SINGLELEFTARROW:
      /* Move one item from the right-hand list box to the left-hand list box. */
      tempList1 = copylist(listBox1.items);
      tempList2 = copylist(listBox2.items);
      numberSelections = listlen(listBox2.selectedItems);
      do i = 1 to numberSelections;
         selectedItem = getitemc(listBox2.selectedItems, i);
         rc = insertc(tempList1, selectedItem, -1);
         index = searchc(tempList2, selectedItem);
         rc = delitem(tempList2, index);
      end;
      listBox1.items = copylist(tempList1);
      listBox2.items = copylist(tempList2);
    return;

   DOUBLERIGHTARROW:
      /* Move all items from the left-hand list box to the right-hand list box. */
      tempList1 = copylist(listBox1.items);
      tempList2 = copylist(listBox2.items);
      do i = 1 to listlen(tempList1);
         rc = insertc(tempList2, getitemc(tempList1, i), -1);
      end;
      listBox1.items = copylist(emptyList);
      listBox2.items = copylist(tempList2);
    return;

   DOUBLELEFTARROW:
      /* Move all items from the right-hand list box to the left-hand list box. */
      tempList1 = copylist(listBox1.items);
      tempList2 = copylist(listBox2.items);
      do i = 1 to listlen(tempList2);
         rc = insertc(tempList1, getitemc(tempList2, i), -1);
      end;
      listBox2.items = copylist(emptyList);
      listBox1.items = copylist(tempList1);
    return;

   GETSELECTIONS:
      /* Retrieve selectedItems from listbox2. */
      call putlist(listBox2.items, 'Selected items', 0);
    return;

   TERM:
      /* Delete the SCL lists. */
      if itemsList then itemsList = dellist(itemsList);
      if emptyList then emptyList = dellist(emptyList);
      if tempList1 then tempList1 = dellist(tempList1);
      if tempList2 then tempList2 = dellist(tempList2);
      rc = rc;
    return;
```

This dual selector does not support enabling and disabling the Push Button Controls as the items are moved between the List Box Controls. It does not support changing the order of the items in the right-hand list box. Both of these features are built-in functionality with the Dual Selector Control.

**Creating a Dual Selector the Easy Way**
The following FRAME entry contains a single object - a Dual Selector Control (Figure 4).

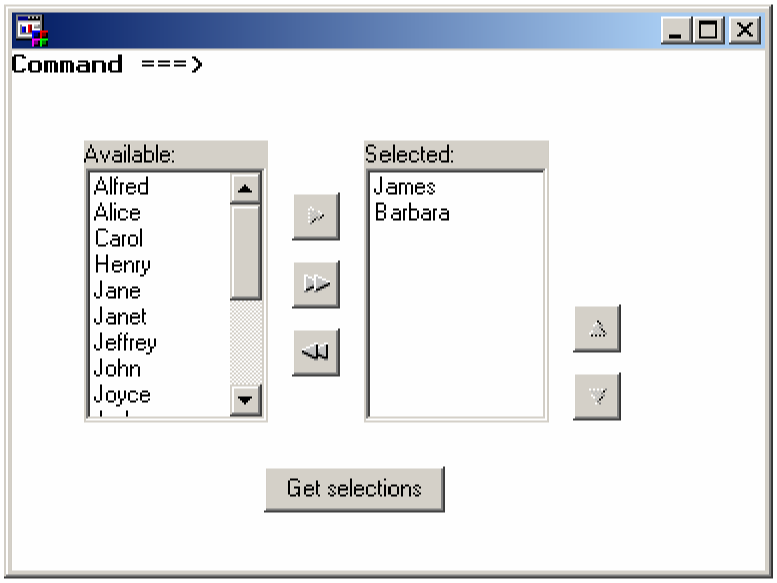


**Figure 4. Dual Selector Control**

The SCL for the FRAME entry contains much less code than the dual selector that was created manually, yet provides more functionality:

```
   dcl list itemsList = {};

   INIT:
      dualSelector1.list2ControlsVisible = ’yes’;
      /* Populate the left-hand list box. */
      dsid = open('sashelp.class');
      nlevels = 0;
      rc = lvarlevel(dsid, 'name', nlevels, itemsList);
      if dsid then dsid = close(dsid);
      itemsList = revlist(itemsList);
      dualSelector1.list1Items = copylist(itemsList);
   return;

   GETSELECTIONS:
      /* Retrieve items from right-hand list box. */
      call putlist(dualSelector1.list2Items, ’selectedItems=’, 0);
    return;

   TERM:
      if itemsList then itemsList = dellist(itemsList);
      rc = rc;
    return;
```

No additional logic is needed in the SCL to enable moving items between the two List Box Controls.

## TREE VIEW CONTROL

The Tree View Control enables you to create a hierarchical list of nodes similar to the Microsoft Windows Explorer folder list. The Tree View Control is used in SAS/AF, in both the Components window and the Properties window, to present a list of classes.

### Understanding the Tree View Control Structure

A Tree View Control is both a composite control comprised of a series of individual node objects (sashelp.classes.treenode_c.class), and also a host control that receives its visual style from the operating system. The node objects in a tree view are visible only at run time. A node can contain both an icon and a text label. In a tree view, the first node is the Root node. All other nodes are defined as children of the Root node. The attributes that define each node, including the Root node, are stored in an SCL list. The SCL list can contain the following items:

| **Item Name** | **Description and Valid Values** |
|---|---|
| text | is the text of the node. |
| iconOpen | is the icon number used when the node is expanded. The default is CATOPEN (#317). |
| iconClosed | is the icon number used when the node is collapsed. The default is SASCAT (#340). |
| showChildren | NO (the default) means that the tree is not expanded if there is a child list. |
| | YES means that the tree is expanded if there is a child list. |
| expandable | YES (the default) means that the node can be expanded. |
| | NO means that the node cannot be expanded. |
| children | is an SCL list that contains the child nodes. |
| userData | is an SCL list that contains information specific to each node. |

To understand the structure of the SCL list that is used to build a tree view, consider the following Tree View Control that has two nodes (Figure 5):

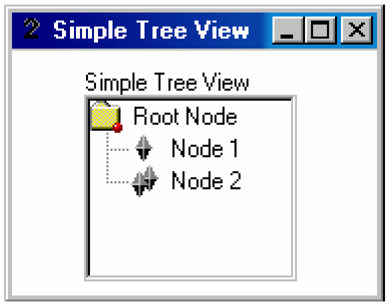


**Figure 5. A Simple Tree View**

The structure of the SCL list for the Tree View Control in Figure 5:

```
   Simple Tree View list: ( (TEXT='Root Node'
                               SHOWCHILDREN='Yes'
```

```
                                          EXPANDABLE='Yes'
                                          CHILDREN=( (TEXT='Node 1'
                                                      ICONCLOSED=610
                                                     )[19635]
                                                     (TEXT='Node 2'
                                                      ICONCLOSED=611
                                                     )[19637]
                                                   )[19633]
                                        )[19631]
                                      )[19623]
```

Nodes can be created and added to a Tree View Control using the _addNodes method. The syntax for _addNodes is:

```
        treeView1._addNodes(referenceNode, listOfNodes, 'insertMode');
```

| **Argument** | **Description and Valid Values** |
|---|---|
| referenceNode | is the object identifier of the starting node. Using a value of zero (0) defines that the nodes are to be added to the root node. |
| listOfNodes | is the list identifier that contains the nodes. |
| insertMode | identifies where to add the nodes in relation to the starting node. Valid values are BEFORE, AFTER (the default), FIRSTCHILD, or LASTCHILD. |

Because nodes added to a Tree View Control using the _addNodes method are not bound to the Tree View Control, you need to recursively delete all lists that you pass to the _addNodes method after it executes.

The SCL for the Simple Tree View in Figure 5:

```
   INIT:
      treeView1.title = 'Simple Tree View';
      dcl list treeList = {},
          list rootList = {};

      rc = insertc(rootList, 'Root Node', -1, 'text');
      rc = insertc(rootList, 'yes', -1, 'showChildren');
      rc = insertc(rootList, 'yes', -1, 'expandable');

      rootChildrenList = makelist();
      rc = insertl(rootList, rootChildrenList, -1, 'children');

      node1 = makelist();
      rc = insertc(node1, 'Node 1', -1, 'text');
      rc = insertn(node1, 608, -1, 'iconClosed');
      rc = insertl(rootChildrenList, node1, -1);

      node2 = makelist();
      rc = insertc(node2, 'Node 2', -1, 'text');
      rc = insertn(node2, 609, -1, 'iconClosed');
      rc = insertl(rootChildrenList, node2, -1);

      rc = insertl(treeList, rootList, -1);
      treeView1._addNodes(0, treeList, 'firstchild');
   return;

   TERM:
      if treeList then treeList = dellist(treeList, 'y');
      rc = rc;
   return;
```

## Finding a Node in a Tree View

For a tree view that contains many nodes, it might be quicker to perform a search to find a node than to scroll through the nodes to find a node. In response to customer feedback, the _findExact method has been added in SAS 9.2. As a result, the Tree View Control now offers two methods for searching, the _find method and the _findExact method. The _find method locates a node that contains a specified string. The _findExact method locates a node with exactly the specified string.

To demonstrate the difference between these two methods, consider the following example in Figure 6. Using the _find method to search for the string ‘aa’, starting from the node labeled “Root Node”, will locate the first node in which the text contains the string ‘aa’. The _find method locates the node labeled ‘aabb’.

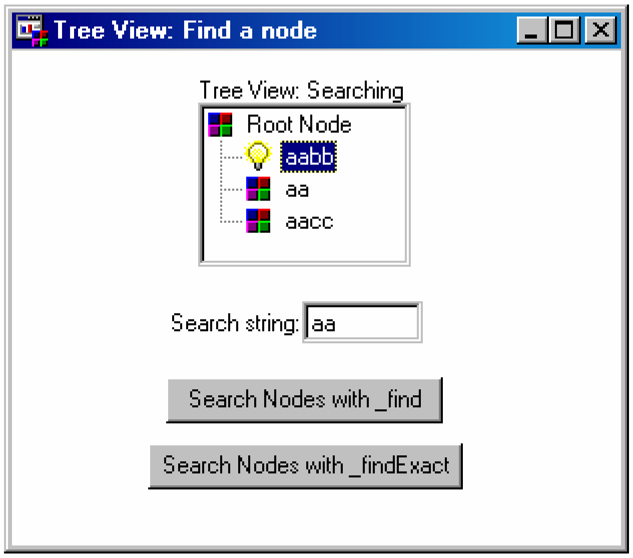


**Figure 6. The _find Method**

The _findExact method locates the node that is labeled ‘aa’ (Figure 7).

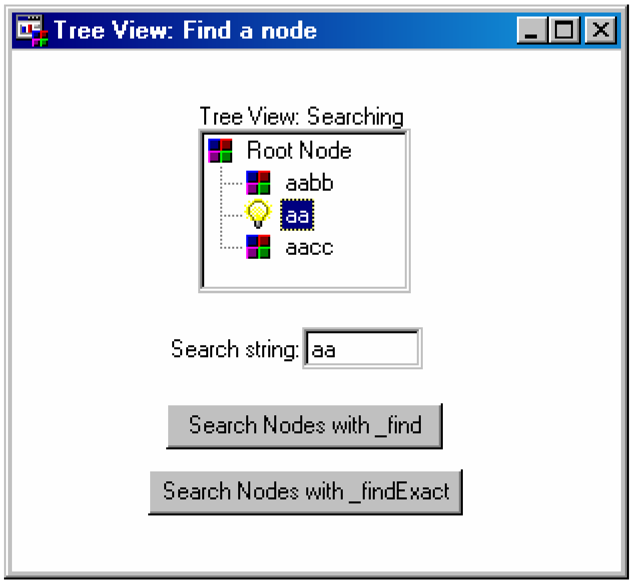


**Figure 7. The _findExact Method**

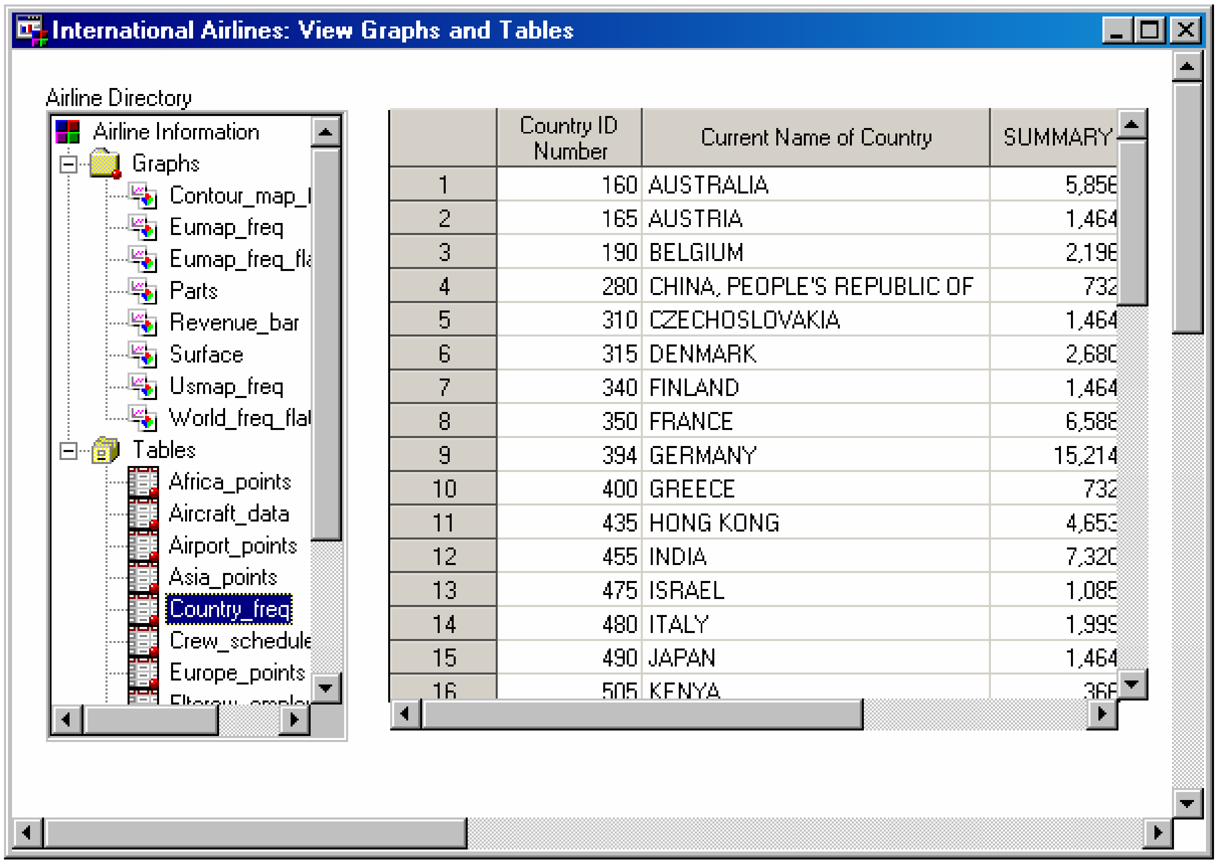
The SCL for the FRAME entry labeled “Find a Node”:

```
dcl sashelp.classes.treenode_c.class foundNode;

INIT:
   treeView1.title = 'Tree View: Searching';
   dcl list treeList = {},
       list rootList = {};

   rc = insertc(rootList, 'Root Node', -1, 'text');
   rc = insertc(rootList, 'yes', -1, 'showChildren');
   rc = insertc(rootList, 'yes', -1, 'expandable');
   rc = insertn(rootList, 590, -1, 'iconClosed');
   rc = insertn(rootList, 590, -1, 'iconOpen');

   rootChildrenList = makelist();
   rc = insertl(rootList, rootChildrenList, -1, 'children');

   node1 = makelist();
   rc = insertc(node1, 'aabb', -1, 'text');
   rc = insertn(node1, 590, -1, 'iconClosed');
   rc = insertn(node1, 123, -1, 'iconOpen');
   rc = insertl(rootChildrenList, node1, -1);

   node2 = makelist();
   rc = insertc(node2, 'aa', -1, 'text');
   rc = insertn(node2, 590, -1, 'iconClosed');
   rc = insertn(node2, 123, -1, 'iconOpen');
   rc = insertl(rootChildrenList, node2, -1);

   node3 = makelist();
   rc = insertc(node3, 'aacc', -1, 'text');
   rc = insertn(node3, 590, -1, 'iconClosed');
   rc = insertn(node3, 123, -1, 'iconOpen');
   rc = insertl(rootChildrenList, node3, -1);

   rc = insertl(treeList, rootList, -1);
   treeView1._addNodes(0, treeList, 'firstchild');
   treeView1._cursor();
   searchString.text = 'aa';
return;

SEARCH:
   /* Find the first node with a text label containing 'aa'. */
   treeView1._find(currentNode, searchString.text, 0, 0, foundNode);
   if foundNode then
   do;
      treeView1.selectedNode = foundNode;
      treeView1._cursor();
   end;
 return;

SEARCHEXACT:
  /* Find the first node with a text label containing the exact string 'aa'. */
  treeView1._findExact(currentNode, searchString.text, 1, 0, foundNode);
   if foundNode then
   do;
      treeView1.selectedNode = foundNode;
      treeView1._cursor();
   end;
 return;
```

```
   TERM:
      if treeList then treeList = dellist(treeList, 'y');
      rc = rc;
   return;
```

**Using the Tree View as a Menu**

To demonstrate the usefulness of a tree view, Figure 8 contains a tree view that serves as a menu. The tree view lists graphs and tables that relate to a fictitious airline company. When a node in the tree view is selected, the graph or table is displayed.



| | Country ID Number | Current Name of Country | SUMMARY |
|---|---|---|---|
| 1 | 160 | AUSTRALIA | 5,856 |
| 2 | 165 | AUSTRIA | 1,464 |
| 3 | 190 | BELGIUM | 2,196 |
| 4 | 280 | CHINA, PEOPLE'S REPUBLIC OF | 732 |
| 5 | 310 | CZECHOSLOVAKIA | 1,464 |
| 6 | 315 | DENMARK | 2,680 |
| 7 | 340 | FINLAND | 1,464 |
| 8 | 350 | FRANCE | 6,588 |
| 9 | 394 | GERMANY | 15,214 |
| 10 | 400 | GREECE | 732 |
| 11 | 435 | HONG KONG | 4,653 |
| 12 | 455 | INDIA | 7,320 |
| 13 | 475 | ISRAEL | 1,085 |
| 14 | 480 | ITALY | 1,999 |
| 15 | 490 | JAPAN | 1,464 |
| 16 | 505 | KENYA | 366 |

**Figure 8. Tree View That Displays Graphs and Tables**

To better understand the code behind the display that is shown in Figure 8, a brief overview of the design requirements, the components used, and the structure of the nodes follows. The code appears in its entirety later in this paper.

The design requirements for this FRAME entry are:

- Verify that the data library is available.
- If the library is not available, indicate this in the root node text.
- If the library is available, populate the tree view with the available graphs and tables.
- When populated, the tree view will have a separate node for Graphs and a separate node for Tables.
- When a selection is made, the corresponding graph or table will display. Graphs will be displayed in an Image Viewer Control. Tables will be displayed in a Table Viewer Control.

The component types and names are:

| Component Type | Component Name |
|---|---|
| Catalog Entry List Model | catalogEntryList1 |
| Data Set List Model | datasetList1 |
| Image Viewer Control | imageViewer1 |
| SAS Data Set Model | sasDataset1 |
| Table Viewer Control | tableViewer1 |
| Tree View Control | treeView1 |

The node structure uses both the Catalog Entry List Model and the Data Set List Model to dynamically populate the Graphs node and the Tables node respectively. The SCL list that populates this tree view has the following structure:

```
Root Node
   Graphs
       Image entries
   Tables
       Table entries
```

To determine whether the Image Viewer Control or the Table Viewer Control will be used to display the selected node, the node for each graph and table contains a userData item that identifies the node as referencing a graph entry or a table entry. Recall that the userData item contains information specific to each node. All the table and graph nodes have the same structure.

Here is the list structure for the node named “Revenue_bar” (a node that references a graph entry):

```
(TEXT='Revenue_bar'
ICONOPEN=16
ICONCLOSED=16
USERDATA=('image'
         )[21315]
)[21311]
```

The complete SCL for the FRAME entry is:

```
dcl sashelp.classes.treenode_c.class currentNode;
dcl char(25) imageName tableName currentNodeText,
    char(5) itemType;

INIT:
   imageViewer1.visible = 'no';
   tableViewer1.visible = 'no';
   currentLibrary = 'IAData';
   dcl list treeList = {},
       list rootList = {};

   libExists = libref('iadata');
   if libExists ne 0 then
   do;
      rc = insertc(rootList, 'Library Unavailable', -1, 'text');
      rc = insertc(rootList, 'no', -1, 'showChildren');
      rc = insertc(rootList, 'no', -1, 'expandable');
      rc = insertn(rootList, 590, -1, 'iconClosed');
   end;

   else
   do;
```

```
rc = insertc(rootList, 'Airline Information', -1, 'text');
rc = insertc(rootList, 'yes', -1, 'showChildren');
rc = insertc(rootList, 'yes', -1, 'expandable');
rc = insertn(rootList, 590, -1, 'iconClosed');
rc = insertn(rootList, 590, -1, 'iconOpen');

rootChildrenList = makelist();
rc = insertl(rootList, rootChildrenList, -1, 'children');

node1 = makelist();
rc = insertc(node1, 'Graphs', -1, 'text');
rc = insertc(node1, 'yes', -1, 'showChildren');
rc = insertc(node1, 'yes', -1, 'expandable');
rc = insertn(node1, 340, -1, 'iconClosed');
rc = insertn(node1, 317, -1, 'iconOpen');
rc = insertl(rootChildrenList, node1, -1);

catalogEntryList1.catalog = 'iadata.graphs';
catalogEntryList1.typeFilter = 'IMAGE';
catalogEntryList1.levelCount = 1;
graphEntryList = catalogEntryList1.items;
numberOfGraphs = listlen(graphEntryList);

if numberOfGraphs then
do;
   node1SubList = makelist();
   rc = insertl(node1, node1SubList, -1, 'children');
   do i = 1 to numberOfGraphs;
      imageName = getitemc(graphEntryList, i);
      graphData = makelist();
      rc = insertc(graphData, 'image', -1);
      imageNodeList = makelist();
      rc = insertl(node1SubList, imageNodeList, -1);
      rc = insertc(imageNodeList, imageName, -1, 'text');
      rc = insertn(imageNodeList, 16, -1, 'iconOpen');
      rc = insertn(imageNodeList, 16, -1, 'iconClosed');
      rc = insertl(imageNodeList, graphData, -1, 'userData');
   end;
end;

node2 = makelist();
rc = insertc(node2, 'Tables', -1, 'text');
rc = insertc(node2, 'yes', -1, 'showChildren');
rc = insertc(node2, 'yes', -1, 'expandable');
rc = insertn(node2, 100, -1, 'iconClosed');
rc = insertn(node2, 661, -1, 'iconOpen');
rc = insertl(rootChildrenList, node2, -1);

datasetList1.library = currentLibrary;
datasetList1.levelCount = 1;
tableList = datasetList1.items;
numberOfTables = listlen(tableList);

if numberOfTables then
do;
   node2SubList = makelist();
   rc = insertl(node2, node2SubList, -1, 'children');
   do i = 1 to numberOfTables;
      tableName = getitemc(tableList, i);
      tableData = makelist();
      rc = insertc(tableData, 'table', -1);
```

```
            tableNodeList = makelist();
            rc = insertl(node2SubList, tableNodeList, -1);
            rc = insertc(tableNodeList, tableName, -1, 'text');
            rc = insertn(tableNodeList, 988, -1, 'iconOpen');
            rc = insertn(tableNodeList, 988, -1, 'iconClosed');
            rc = insertl(tableNodeList, tableData, -1, 'userData');
         end;
      end;

   end;
   rc = insertl(treeList, rootList, -1);
   treeView1._addNodes(0, treeList, 'firstchild');
return;

TREEVIEW1:
   currentNode = treeView1.selectedNode;
   if currentNode then
   do;
      /* When a node is selected, the text and userData attributes are */
      /* queried for the entry name and entry type respectively. The   */
      /* corresponding graph or table is then displayed.               */
      currentNodeText = currentNode.text;
      nodeData = currentNode.userData;
      if listlen(nodeData) gt 0 then
      do;
         itemType = getitemc(nodeData, 1);
         if itemType = 'image' then
         do;
            tableViewer1.visible = 'no';
            sasDataset1.table = ' ';
            imageViewer1.image = 'iadata.graphs.' || trim(currentNodeText);
            imageViewer1.visible = 'yes';
         end;
         else if itemType = 'table' then
         do;
            imageViewer1.visible = 'no';
            imageViewer1.image = ' ';
            sasDataset1.table = 'iadata.' || trim(currentNodeText);
            tableViewer1.visible = 'yes';
         end;
      end;
   end;
return;

TERM:
   if treeList then treeList = dellist(treeList, 'y');
   if graphEntryList then graphEntryList = dellist(graphEntryList);
   if tableList then tableList = dellist(tableList);
   rc = rc;
return;
```

## CONCLUSION

SAS/AF software is a powerful application development tool that is easy to use and can execute in an environment with as little as an installation of Base SAS. The versatility of SCL (the scripting language that SAS/AF uses) enables it to also function as the scripting language for Web-based applications deployed through SAS/IntrNet software. The enhancements discussed in this paper, the publication of SAS Samples, the upcoming primer, and the return to the 2006 SASware ballot of SAS/AF, SCL, and VIEWTABLE demonstrate our long-standing commitment to this product. SAS/AF software is here to stay!

## ACKNOWLEDGEMENTS

The author thanks Chris Barrett and Ann Rockett who reviewed this paper and Kathleen Walch and her team for their assistance and patience during the editing process.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Lynn Curley
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Lynn.Curley@sas.com