Paper 019-2008

Branching Out with the Tree View Control in SAS/AF® Software

Lynn Curley, SAS Institute Inc., Cary, NC
Scott Jackson, SAS Institute Inc., Cary, NC

ABSTRACT

Harness the power of the Tree View Control in SAS/AF® software to present hierarchical data like the Microsoft Windows Explorer folder list. The Tree View Control was previously experimental, but has been promoted to production status in SAS® 9.2 thanks to customer feedback. Through examples, this paper introduces you to both the Tree View and Tree Node classes and details the attributes and methods that help make the Tree View a customizable, multi-purpose object.

INTRODUCTION

Several years ago, SAS redirected development resources from SAS/AF software toward SAS® AppDev Studio, which is based on a Java development environment. At that time, SAS encouraged its customers to move in this direction. The migration led some SAS customers to question the future of SAS/AF software. Although SAS is still committed to developing technologies that are based on Java, SAS recognizes that many of its customers want and need the rich object-oriented programming environment that SAS/AF software offers. Therefore, in response to customer feedback, there are several enhancements for SAS/AF software in SAS 9.2, including promotion of the experimental Tree View and Tree Node Controls to production status.

This paper discusses the capabilities of the Tree View Control in SAS/AF software and meets the following objectives:

- details the hierarchical structure of a Tree View Control, including the node objects that populate the tree view
- demonstrates two methods to search for nodes in a Tree View Control
- includes a customizable example of using a Tree View Control as a menu via drag-and-drop communication.

For information on other SAS/AF software enhancements in SAS 9.2, see the Appendix.

UNDERSTANDING THE TREE VIEW CONTROL STRUCTURE

The tree view enables you to create a hierarchical list of nodes similar to a Microsoft Windows Explorer folder list. A Tree View Control is both a composite control consisting 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 Control 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 a SAS Component Language (SCL) list. The SCL list can contain the following items:

Item Name	Description and Valid Values
`text`	the text of the node
`iconOpen`	the icon number used when the node is expanded. The default is CATOPEN (#317).
`iconClosed`	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`	an SCL list that contains the child nodes
`userData`	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 Control, consider the simple tree view that is shown in Figure 1, which has two nodes.

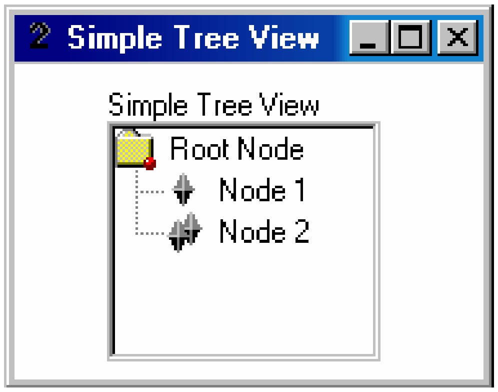


Figure 1. A Simple Tree View

Here is the structure of the SCL list for the simple tree view that is shown in Figure 1:

```
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. Here is the syntax for this method:

```
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 that are added to a Tree View Control using the _addNodes method are not bound to the Tree View Control, they will not be deleted when you terminate the Tree View Control. Therefore, you need to recursively delete all lists that you pass to the _addNodes method after the method executes.

Here is the complete SCL for the simple tree view that is shown in Figure 1:

```
INIT:
   treeView1.title = 'Simple Tree View';
   dcl list treeList = {},
       list rootList = {text = 'Root Node',
                        showChildren = 'yes',
                        expandable = 'yes' };

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

   dcl list node1 = {text = 'Node 1',
                     iconClosed = 610 };
   rc = insertl(rootChildrenList, node1, -1);

   dcl list node2 = {text = 'Node 2',
                     iconClosed = 611 };
   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 CONTROL

For a tree view that contains many nodes, it is faster to find a node with a search than by scrolling through the nodes. In response to customer requests to broaden the search capability of the Tree View Control, the class 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. Suppose you want to search for the string **`aa`** from the **`Root`** node. Using the _find method will locate the first node in which the text contains the string **`aa`**. In this example, the _find method locates the node labeled **`aabb`** (see Figure 2). Using the _findExact method locates the node that is labeled **`aa`** (see Figure 3).

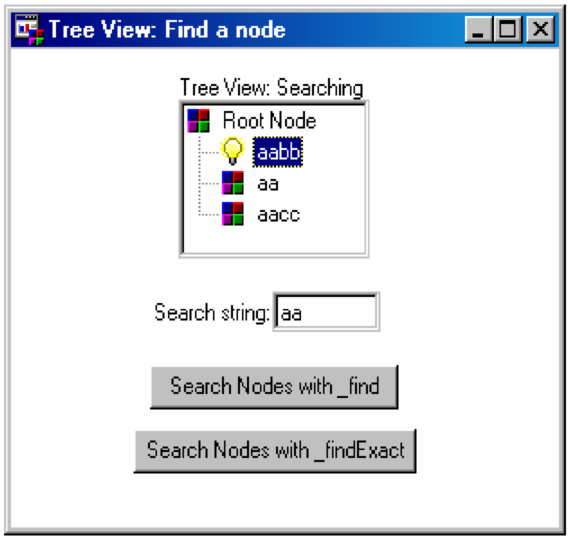


Figure 2. Searching for Nodes using the _find Method

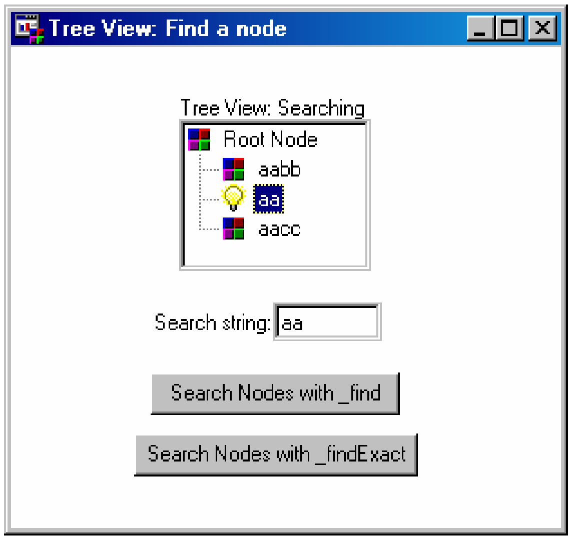


Figure 3. Searching for Nodes using the _findExact Method

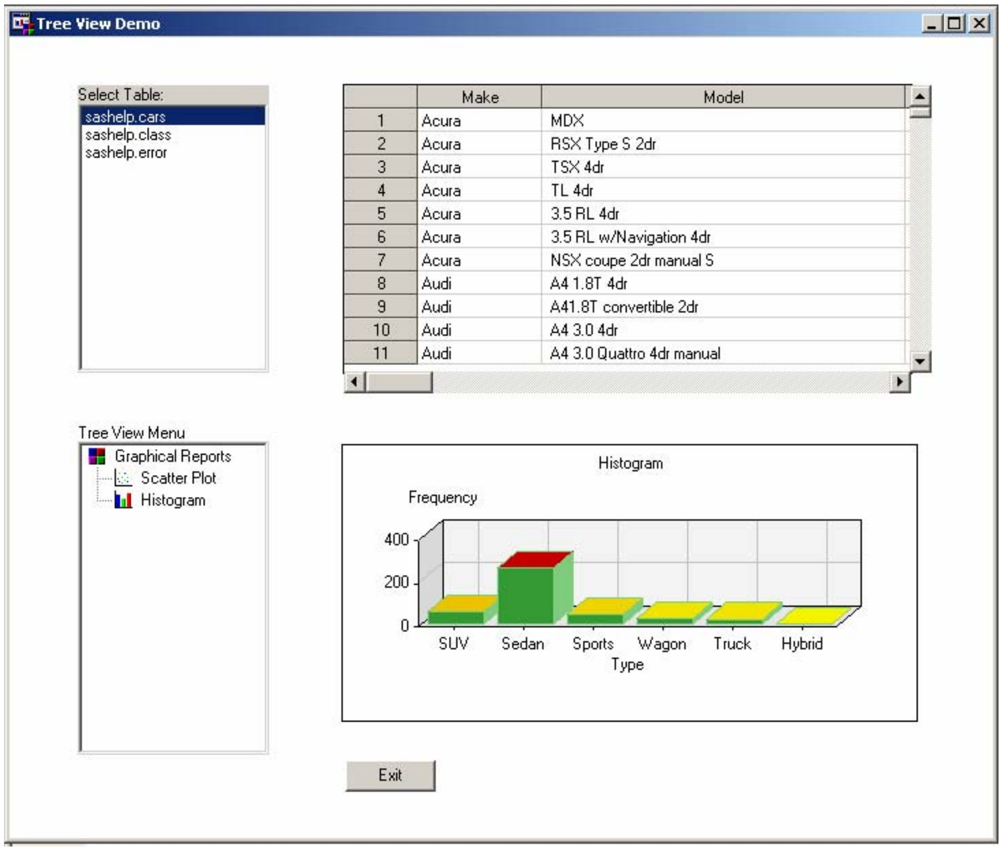
Here is the SCL for the “Find a Node” FRAME entry:

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

INIT:
   treeView1.title = 'Tree View: Searching';
   dcl list treeList = {},
       list rootList = {text = 'Root Node',
                        showChildren = 'yes',
                        expandable = 'yes',
                        iconClosed = 590,
                        iconOpen = 590 };

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

   dcl list node1 = {text = 'aabb',
                     iconClosed = 590,
                     iconOpen = 123 };
   rc = insertl(rootChildrenList, node1, -1);

   dcl list node2 = {text = 'aa',
                     iconClosed = 590,
                     iconOpen = 123 };
   rc = insertl(rootChildrenList, node2, -1);

   dcl list node3 = {text = 'aacc',
                     iconClosed = 590,
                     iconOpen = 123 };
   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

A tree view has the ability to display a large number of items, or nodes, in a relatively small space. This capability is equivalent to using menus to display a compact list of choices. With the Tree View Control, you can easily incorporate the functionality of a menu into a FRAME entry through a single object. The following example demonstrates this capability.

Figure 4 contains a Tree View Control that serves as a menu. In the example that follows, the Tree View Control, labeled **`Tree View Menu`**, contains nodes for two types of graphical reports: a histogram and a scatter plot. To create the report, you drag a table name from the list box and drop it on the appropriate node. A dialog box is displayed from which you then select the variables for the X and Y axes. Thereafter, the data populates a Table Viewer Control and the graph displays.



	Make	Model
1	Acura	MDX
2	Acura	RSX Type S 2dr
3	Acura	TSX 4dr
4	Acura	TL 4dr
5	Acura	3.5 RL 4dr
6	Acura	3.5 RL w/Navigation 4dr
7	Acura	NSX coupe 2dr manual S
8	Audi	A4 1.8T 4dr
9	Audi	A41.8T convertible 2dr
10	Audi	A4 3.0 4dr
11	Audi	A4 3.0 Quattro 4dr manual

Figure 4. The Tree View, Labeled `Tree View Menu`, Serves as a Menu to Create Graphical Reports

EXAMPLE: BUILDING THE TREE VIEW MENU

This example is easy to create. Each step is explained in detail in the sections that follow. The first two steps create two new classes that process the information; one class drags and the other class drops the information. For a detailed discussion of drag-and-drop communication in SAS/AF software, refer to the chapter "Drag and Drop Communication" in *The SAS Guide to Application Development* (SAS Institute Inc. 2004).

These new classes are then combined with standard classes on a frame to enable the drag-and-drop feature between the list box and the tree view. The steps presented in the following sections assume that everything is being built in the SASUSER library and in a catalog named TREEVIEW. If you use another library or catalog, you must modify the example accordingly.

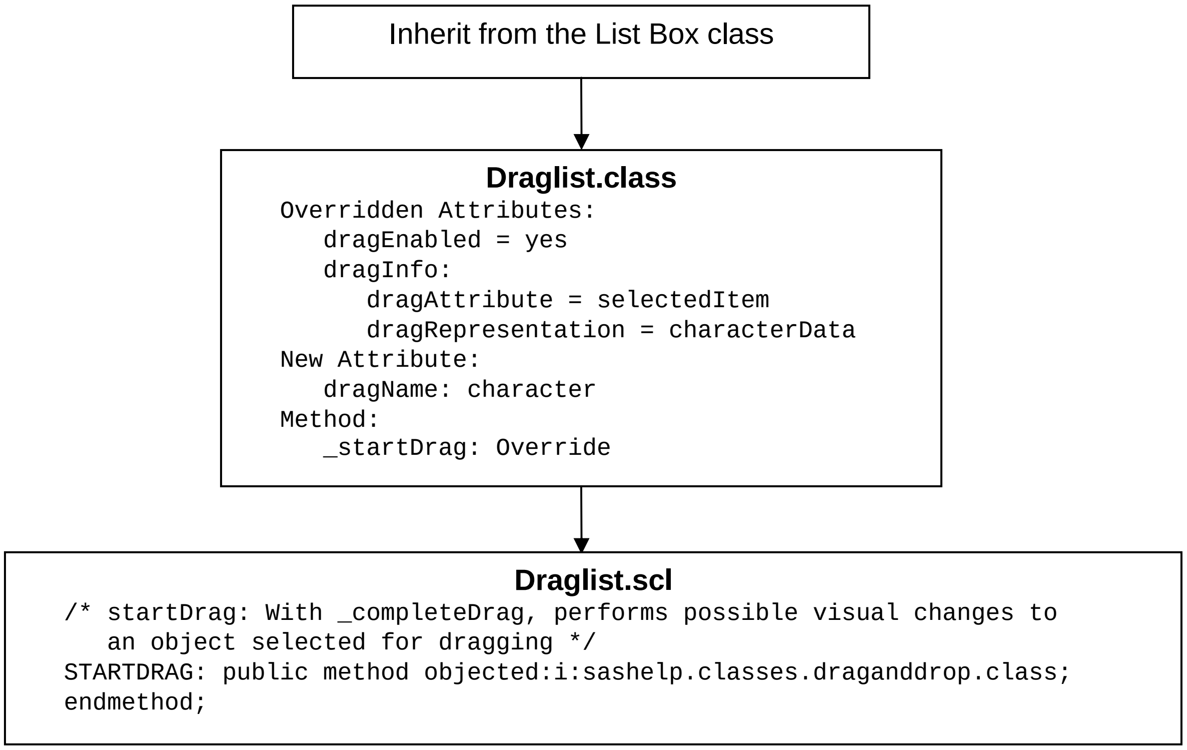
Creating the Draglist Class

The first class that you need to create is a subclass of the List Box class. In this example the subclass is named Draglist class. This new class includes the following customizations:

- a new attribute, **`dragName`**
- an override of the **`dragEnabled`** attribute

- an override of the **`dragInfo`** attribute
- an override of the _startDrag method

Figure 5 shows an overview of how the resulting class is created and the SCL code for the _startDrag method.



```
Overridden Attributes:
   dragEnabled = yes
   dragInfo:
      dragAttribute = selectedItem
      dragRepresentation = characterData
New Attribute:
   dragName: character
Method:
   _startDrag: Override
```

```
/* startDrag: With _completeDrag, performs possible visual changes to
   an object selected for dragging */
STARTDRAG: public method objected:i:sashelp.classes.draganddrop.class;
endmethod;
```

Figure 5. The Draglist Class and SCL Code for the _startDrag Method

Follow these steps to create the Draglist class:

1. Issue the command **`build sasuser.treeview.draglist.class`**. The Class Editor appears.
2. In the Class Editor, specify the parent as *sashelp.classes.listbox_c.class*.
3. Select **`Attributes`** from the tree view on the left.
4. Right-click **`Attributes`** and select **`New Attribute`**.
5. In the Attributes table, specify the following:
   a. Name as *dragName*
   b. Type as *Character*
6. Right-click the **`dragEnabled`** attribute and select **`Override`**. Specify **`Yes`** for Initial Value.
7. Right-click the **`dragInfo`** attribute and select **`Override`**. Update the list for Initial Value as follows:
   a. Drag Attribute = *selectedItem*
   b. Drag Representation = *characterData*
8. Select **`Methods`** from the tree view on the left.
9. Right-click the _startDrag method with the signature `(o:sashelp.classes.draganddrop.class;)v` and select **`Override`**. Specify the source entry as *sasuser.treeview.draglist.scl*.
10. Right-click the displayed code at the bottom of the window and select **`Copy code`**.
11. Right-click the _startDrag method a second time and select **`Source`**. This action will create and open the source entry sasuser.treeview.draglist.scl.
12. Paste the code you copied in Step 9 into the SCL entry.
13. Compile and save the SCL code. Close the SCL entry.
14. Save and close Draglist.class.

Creating the Dropnode Class

The second class that you need to create is a subclass of the Tree Node class. In this example the subclass is named Dropnode class. This class includes the following customizations:

- a new attribute, **`dropTableNameAttr`**
- a new method, setcamDropTableNameAttr

The new attribute is created to contain the table name that is dropped on the node. The new method is a custom access method (CAM), which is automatically invoked when the attribute's value is set. In this example, the method supports selecting the variables for the X and Y axes and creating the graph.

The actions performed by this method require an understanding of FRAME entries and object-oriented programming. In this example, the new method belongs to the Dropnode class and needs to access information from other objects that are on the frame (the Table Viewer, Scatter, and Histogram Controls). To get the information, the node object must know the object identifier of each of those objects. But, because objects only know what they contain or are contained by (and not about other objects on the frame), you must retrieve the object identifiers of the other object from the top level container (the FRAME entry). Put another way, you must traverse the hierarchy up from the node to the tree view to get the ID of the FRAME entry. Because the FRAME entry knows what it contains, you can get the Table Viewer, Scatter, and Histogram Control identifiers. With these identifiers, you can then assign a table to display in the Table Viewer Control and set the variables for the X and Y axes in the scatter plot or histogram.

To do all of this, you must first retrieve the ID of the FRAME entry that contains all the objects. Because the node knows that it is contained by the tree view, and the tree view knows the identifier of the FRAME entry that contains it, you need only walk up the hierarchy to access the appropriate attributes.

The **`viewer`** attribute on the Tree Node class contains the identifier of the parent Tree View object. The **`frameId`** attribute on the Tree View object contains the identifier of the frame. Therefore, to retrieve the ID of the FRAME entry from the node, you must execute the following statement:

```
      frameID = _self_.viewer.frameid;
```

Once you have the value of the **`frameID`** attribute, you can then execute the _getWidget method to retrieve the object identifiers for the Table Viewer Control, Scatter Control, and Histogram Controls; for example,

```
      frameID._getWidget('tableViewer', tableID);
```

You can easily determine whether the graph chosen via drag-and-drop communication is a scatter plot or a histogram. Each node in the Tree View Control contains a **`userData`** list which contains an item named **`type`** that identifies the graph type as a scatter plot or a histogram. Recall that the **`userData`** item contains information that is specific to each node. In this case, the **`userData`** list contains an item named **`type`** with a value of *scatter* or *histogram*. The following statements retrieve the value of type for the current node:

```
      userList = _self_.userData;
      nodeType = getnitemc(userList, 'type', 1, 1);
```

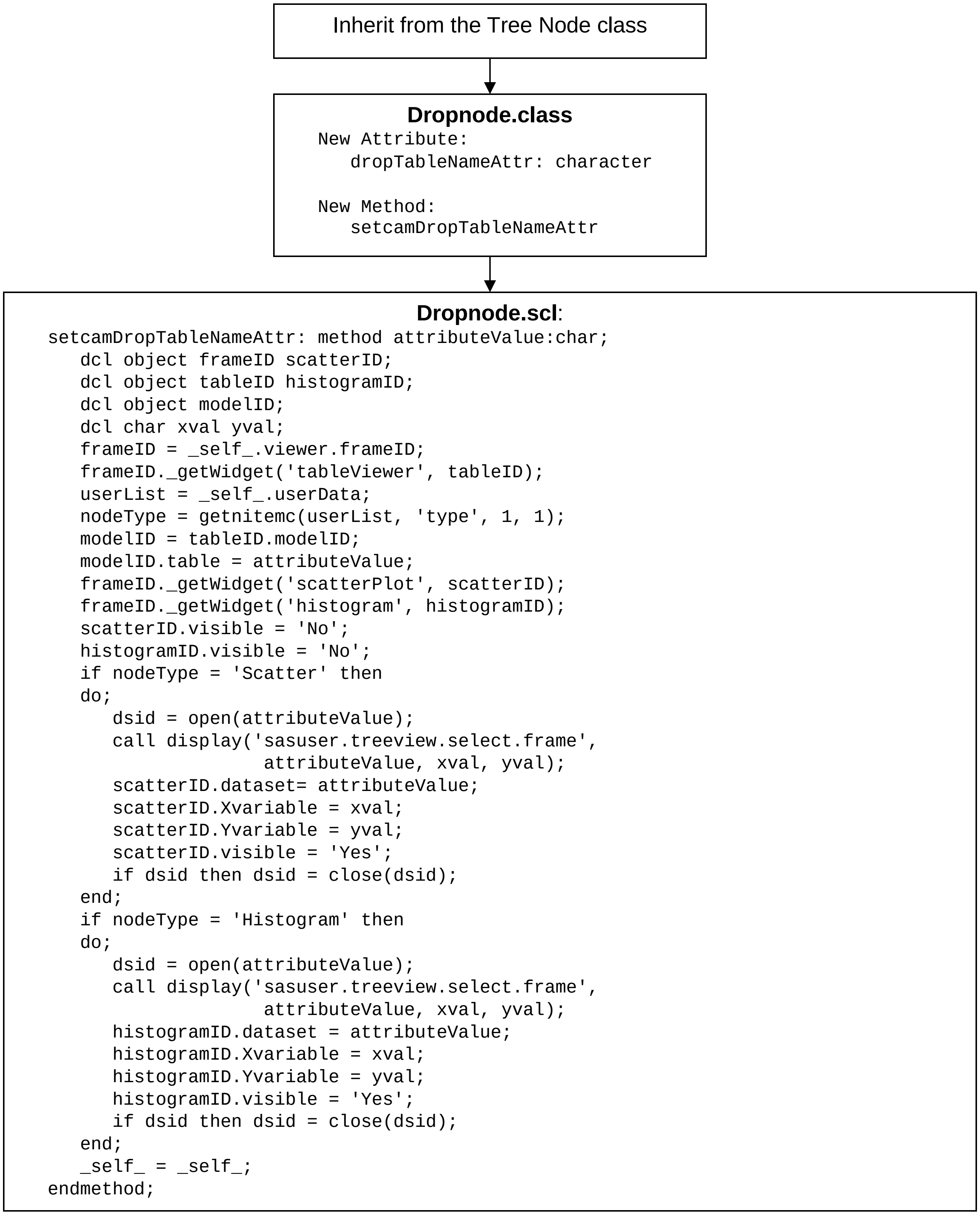
The table selected in the List Box Control is stored in the attribute **`dropTableNameAttr`**. Within the setcamDropTableNameAttr method, we have stored this value in the attributeValue argument of the method. Therefore, if the table named is dropped on the node for the scatter plot, you set the **`dataset`** attribute as follows:

```
      scatterID.dataset = attributeValue;
```

In this example, both the scatter plot and the histogram require the definition of analysis variables for the X and Y axes. This is accomplished by calling SELECT.FRAME:

```
   call display('sasuser.treeview.select.frame', attributeValue, xval, yval);
```

The parameters *attributeValue*, *xval*, and *yval* contain the values for the **`dataset`**, **`Xvariable`**, and **`Yvariable`** attributes, respectively. Figure 6 shows an overview of the DropNodeclass and the SCL code for the setcamDropTableNameAttr method.



```
New Attribute:
   dropTableNameAttr: character

New Method:
   setcamDropTableNameAttr
```

```
setcamDropTableNameAttr: method attributeValue:char;
   dcl object frameID scatterID;
   dcl object tableID histogramID;
   dcl object modelID;
   dcl char xval yval;
   frameID = _self_.viewer.frameID;
   frameID._getWidget('tableViewer', tableID);
   userList = _self_.userData;
   nodeType = getnitemc(userList, 'type', 1, 1);
   modelID = tableID.modelID;
   modelID.table = attributeValue;
   frameID._getWidget('scatterPlot', scatterID);
   frameID._getWidget('histogram', histogramID);
   scatterID.visible = 'No';
   histogramID.visible = 'No';
   if nodeType = 'Scatter' then
   do;
      dsid = open(attributeValue);
      call display('sasuser.treeview.select.frame',
                   attributeValue, xval, yval);
      scatterID.dataset= attributeValue;
      scatterID.Xvariable = xval;
      scatterID.Yvariable = yval;
      scatterID.visible = 'Yes';
      if dsid then dsid = close(dsid);
   end;
   if nodeType = 'Histogram' then
   do;
      dsid = open(attributeValue);
      call display('sasuser.treeview.select.frame',
                   attributeValue, xval, yval);
      histogramID.dataset = attributeValue;
      histogramID.Xvariable = xval;
      histogramID.Yvariable = yval;
      histogramID.visible = 'Yes';
      if dsid then dsid = close(dsid);
   end;
   _self_ = _self_;
endmethod;
```

Figure 6. The Dropnode Class and SCL Code

Follow these steps to create the Dropnode class:

1. Issue the command **`build sasuser.treeview.dropnode.class`**. The Class Editor appears.
2. In the Class Editor, specify the parent as *sashelp.classes.treenode_c.class*.
3. Select **`Attributes`** from the tree view on the left.
4. Right-click **`Attributes`** and select **`New Attribute`**.
5. In the Attributes table, specify
   c. Name as *dropTableNameAttr*
   d. Type as *Character*
   e. setCAM as *setcamDropTableNameAttr*
6. When you are prompted to create the method setcamDropTableNameAttr, click **`Yes`**.
7. In the New Method window, click **`Source`**. The SCL entry sasuser.treeview.dropNode.scl is created and opened.
8. Paste the Dropnode.scl code that is shown in Figure 6 into the SCL entry.
9. Compile and save the SCL code.
10. Close the SCL entry and click **`OK`** to close the New Method window.
11. Save and close the Dropnode.class.

Creating SELECT.FRAME

The Dropnode class relies on a frame to define the variables for the X and Y axes in the Scatter Plot and Histogram Controls. This frame contains two Dual Selector controls: one to select the variable for the X axis and one to select the variable for the Y axis. Each Dual Selector is populated by a Variable List Model. The frame also contains a Push Button Control that is used to close the window. Figure 7 details this frame and the SCL code that is behind it.

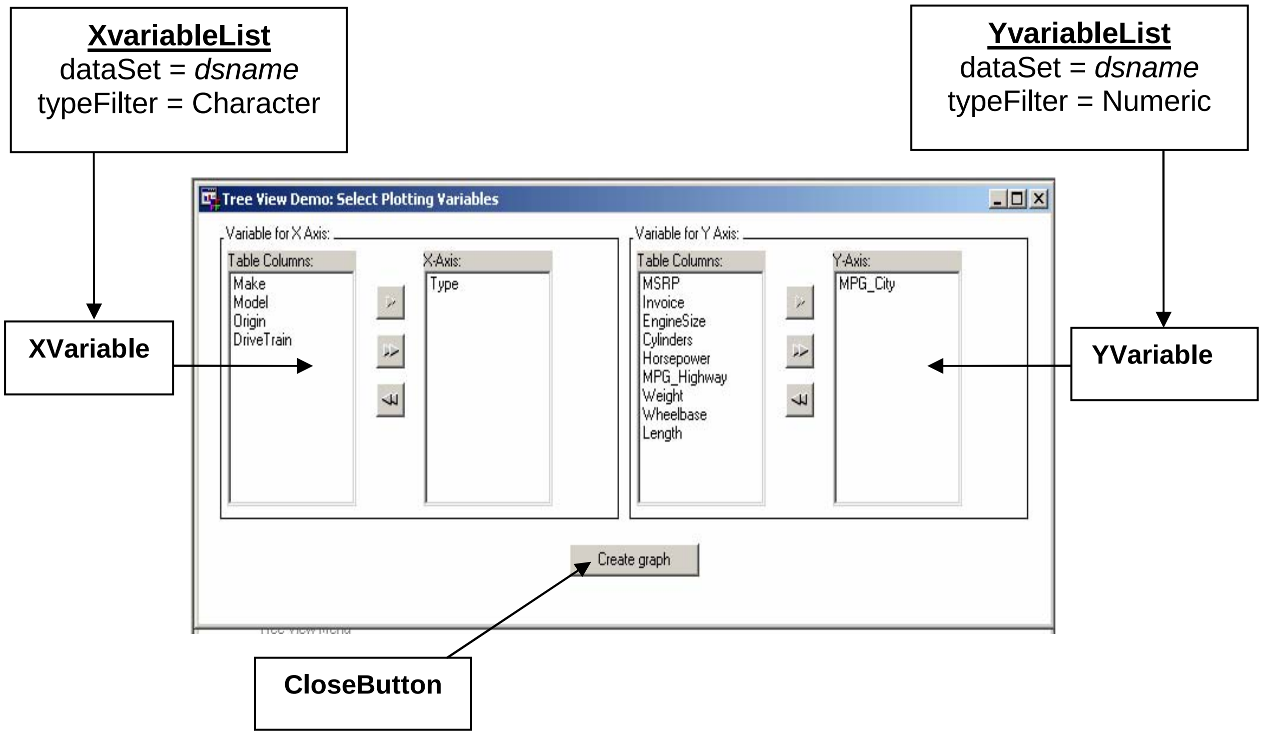


Figure 7. SELECT.FRAME

Here is the SCL for SELECT.FRAME:

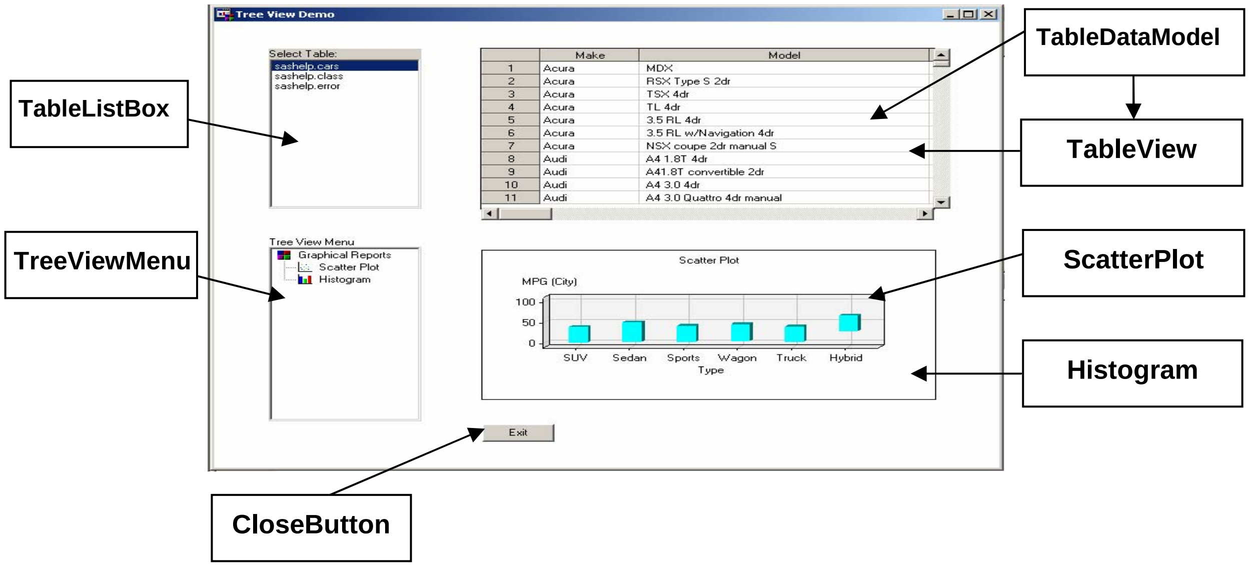
```
entry dsname:char xvar:char yvar:char;
INIT:
   XvariableList.dataSet = dsname;
   YvariableList.dataSet = dsname;
   closeButton.commandOnClick = 'End';
return;

TERM:
   if listlen(Xvalue.list2Items) gt 0 then
      xvar = getitemc(Xvalue.list2Items, 1);
   else
   do;
      _status_='R';
      dcl list commandlist = {'Please select a variable for the X axis.'};
      command = messagebox(commandlist, '!', 'O', '', 'O', '');
      commandlist = dellist(commandlist);
   end;
   if listlen(Yvalue.list2Items) gt 0 then
   yvar = getitemc(Yvalue.list2Items, 1);
   else
   do;
      _status_='R';
      dcl list commandlist = {'Please select a variable for the Y axis.'};
      command = messagebox(commandlist, '!', 'O', '', 'O', '');
      commandlist = dellist(commandlist);
   end;
return;
```

Follow these steps to create SELECT.FRAME:

1. Issue the command **`build sasuser.treeview.select.frame`**. The new frame opens.
2. In the new frame, drag the following controls from the Components window onto the frame and position them as you see in Figure 7:
   a. two Dual Selector Controls
   b. a Push Button Control
   c. two Variable List Models
3. In the Properties window, set the **`name`** attribute of each control as follows:
   a. Dualselector1 to *XVariable*
   b. Dualselector2 to *YVariable*
   c. Pushbutton1 to *CloseButton*
   d. Variablelist1 to *XVariableList*
   e. Variablelist2 to *YVariableList*
4. In the Properties window, set these additional attributes:
   a. _FRAME_.bannerType = *None*
   b. _FRAME_.title = “*Tree View Demo*”
   c. CloseButton.label = *Create Graph*
   d. Xvariable.borderStyle = *Simple*
   e. Xvariable.borderTitle = “*Variable for X Axis:*”
   f. Xvariable.model = *XVariableList*
   g. XvariableList.typeFilter = *Character*
   h. Yvariable.borderStyle = *Simple*
   i. Yvariable.borderTitle = “*Variable for Y Axis:*”
   j. Yvariable.model = *YVariableLis*t
   k. YvariableList.typeFilter = *Numeric*
5. Paste the code for SELECT.FRAME into the SCL entry.
6. Compile and save the FRAME entry.
7. Close the FRAME entry.

Creating TREEDEMO.FRAME

The final step in this example is to create TREEDEMO.FRAME. This FRAME entry contains an instance of Draglist.class that you created earlier. This list box contains table names that, when selected, will serve as the data source for the graph that you choose to create. The FRAME entry also contains a Tree View Control, a Table Viewer Control, a Scatter Control, and a Histogram Control. Dropnode.class will be associated to the Tree View Control and the SAS Data Set Model class will be attached to the Table Viewer Control. The details of this frame are shown in Figures 8.



	Make	Model
1	Acura	MDX
2	Acura	RSX Type S 2dr
3	Acura	TSX 4dr
4	Acura	TL 4dr
5	Acura	3.5 RL 4dr
6	Acura	3.5 RL w/Navigation 4dr
7	Acura	NSX coupe 2dr manual S
8	Audi	A4 1.8T 4dr
9	Audi	A41.8T convertible 2dr
10	Audi	A4 3.0 4dr
11	Audi	A4 3.0 Quattro 4dr manual

Figure 8. TREEDEMO.FRAME

Here is the code for the method override of _validateDropData. This is stored in Treedemomethods.scl.

```
/* _validateDropData: Allows a drop site to validate a drag operation's data */
validateDropData: public method objectID:i:sashelp.classes.draganddrop.class;
   dsName = objectID.attributeValue.characterValue;
   objectID.completeDrag = 'Yes';
   if dsName = 'sashelp.error' then
   do;
      objectID.completeDrag = 'No';
      dcl list commandlist = {'The table does not exist.'};
      dcl char(10) command;
      command = messagebox(commandlist, '!', 'O', 'Warning');
      commandlist = dellist(commandlist);
   end;
endmethod;
```

Here is the SCL for TREEDEMO.FRAME:

```
INIT:
   scatterPlot.visible = 'No';
   histogram.visible = 'No';
   closeButton.commandOnClick = 'end;';
   dcl list treeList = {};
   dcl list rootList = {text = 'Graphical Reports',
                        showChildren = 'yes',
                        expandable = 'yes',
                        iconClosed = 590,
                        iconOpen = 590 };

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

   dcl list node1 = {text = 'Scatter Plot',
                     iconClosed = 393,
                     iconOpen = 393 };
   dcl list userData1 = {type = 'Scatter' };
   rc = insertl(node1, userData1, -1, 'userData');
   rc = insertl(rootChildren, node1, -1);

   dcl list node2 = {text = 'Histogram',
                     iconClosed = 128,
                     iconOpen = 128 };
   dcl list userData2 = {type = 'Histogram' };
   rc = insertl(node2, userData2, -1, 'userData');
   rc = insertl(rootChildren, node2, -1);

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

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

Follow these steps to create TREEDEMO.FRAME:

1. Issue the command **`build sasuser.treeview.treedemo.frame`**. The new frame opens.
2. In the new frame, drag the following controls from the Components window onto the frame and position them as you see in Figure 8:
    a. Tree View Control
    b. Table Viewer Control
    c. SAS Data Set Model
    d. Scatter Control
    e. Histogram Control
3. Add the Draglist class by dragging this class from a SAS Explorer window onto the FRAME entry. Position this object as you see in Figure 8.
4. In the Properties window, set the **`name`** attribute of each control as follows:
    a. Draglist1 to *TableListBox*
    b. Histogram1 to *Histogram*
    c. Pushbutton1 to *CloseButton*
    d. Sasdataset1 to *TableDataModel*
    e. Scatter1 to *ScatterPlot*
    f. Tableviewer1 to *TableView*
    g. Treeview1 to *TreeViewMenu*

5. In the Properties window, update the following attributes:
   a. _FRAME_.bannerType = *None*
   b. _FRAME_.title = "*Tree View Demo*"
   c. CloseButton.label = *Exit*
   d. Histogram.borderStyle = *Simple*
   e. Histogram.title1.text = *Histogram*
   f. Histogram.visible = *No*
   g. ScatterPlot.borderStyle = *Simple*
   h. ScatterPlot.title1.text = *Scatter Plot*
   i. ScatterPlot.visible = *No*
   j. TableListBox.items = *sashelp.class*, *sashelp.cars*, and *sashelp.error* (a table that does not exist)
   k. TableListBox.title = *Select Table*
   l. Tableview.growColumnsToFillTable = *Yes*
   m. Tableview.growRowsToFillTable = *Yes*
   n. Tableview.model = *TableDataModel*
   o. Tableview.showPartialRow = *No*
   p. TreeViewMenu.dropEnabled = *Yes*
   q. TreeViewMenu.dropInfo:
      i. Drop Attribute = *dropTableNameAttr*
      ii. Drop Representation = *characterData*
   r. TreeViewMenu.nodeClass = *sasuser.treeview.dropnode.class*
   s. TreeViewMenu.title = "*Tree View Menu*"
6. In the Properties window, override the following method:
   a. TreeViewMenu: _validateDropData `(o:sashelp.classes.draganddrop.class;)v`
   b. Right-click the _*validateDropData* method and select **`Source`**. This action will create and open sasuser.treeview.Treedemomethods.scl.
   c. Paste the code for Treedemomethods.scl into the SCL entry.
   d. Compile and save the SCL code.
   e. Close Treedemomethods.scl.
7. Close the Properties window.
8. Right-click in the FRAME entry and select **`Frame SCL`**.
9. Paste the code for TREEDEMO.FRAME into the SCL entry.
10. Compile and save the FRAME entry.

CONCLUSION

The Tree View Control is a hierarchical list of nodes similar to a Microsoft Windows Explorer folder list. It is very customizable and can easily serve as a menu tool within a FRAME entry. You can make selections by clicking on a node or through drag-and-drop communication. The Tree View Control is already used in SAS/AF software, in both the Components window and the Properties window, to present a list of classes. Now, with SAS 9.2, you can have the opportunity to use these controls in your own SAS/AF applications!

RECOMMENDED READING

Curley, Lynn. 2006. "SAS/AF® Rises Again: Enhancements in SAS® 9.2". *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at www2.sas.com/proceedings/sugi31/008-31.pdf.

Massengill, Darrell and Jackson, Scott. 2006. "SAS/AF®–Building a Tree View Hierarchy with Drag and Drop". *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at www2.sas.com/proceedings/sugi31/037-31.pdf.

SAS Institute Inc. 2004. "Drag and Drop Communication". *SAS® Guide to Applications Development, Second Edition*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/af_appguide_7821.pdf#page=103.

SAS Institute Inc. 2007. SAS Note 24440: "Available documentation and instructional resources for SAS/AF® software". Available at support.sas.com/kb/24/440.html.

ADDENDIX

The following enhancements for SAS/AF software are being delivered in SAS 9.2:

Support for long format and informat names (This enhancement has also been added for SAS/FSP® software and the VIEWTABLE window.)

SAS customers, especially those who use the VIEWTABLE window, have requested support of long format and informat names for a long time. 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. Because the long names were not supported, some data would display unformatted. Support for long format and informat names is included in SAS 9.2.

Promotion of two experimental classes to production status: Dual Selector Control and Tree View Control

SAS 8 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 customer requests, these two classes are being enhanced and promoted to production status in SAS 9.2.

Documentation of the V6GUIMODE system option for SAS in a mainframe operating environment

Beginning with SAS 8, customers using SAS/AF software in a mainframe operating environment experienced difficulties with the BUILD Explorer window. Also, customers using SAS/AF or SAS/FSP saw changes with the selection list windows. By specifying the V6GUIMODE option when starting SAS, mainframe customers can use the SAS 6 style of selection list windows that is slightly larger than the SAS 8 style selection list windows. 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. The V6GUIMODE option is documented in SAS 9.2.

ACKNOWLEDGEMENTS

The authors thank Chris Barrett, Adam Budlong, Ivy Parker, and Ann Rockett who reviewed this paper and Kathleen Walch and her team for their assistance during the editing process.

CONTACT INFORMATION

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

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

Scott Jackson
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Scott.Jackson@sas.com