Paper 174-26

Three for the Show—Three Objects for Presenting Categorized Data

Tom Miron, Miron InfoTec, Inc., Madison , WI

ABSTRACT

Data is often categorized into a small number general groups. For example, sales data by region, experimental data by trial number, or personnel by corporate division. You can visually distinguish such categories using SAS/AF FRAME controls. Three of these controls are discussed and demonstrated: the SAS table model with use of color, the org chart, and the tabber.

INTRODUCTION

This paper is not about graphic design. It is about three tools you can build and use to implement a specific design or use to create a functional application quickly. The tools in this case are SAS/AF controls (widgets). We will be using object-oriented development techniques, but an in-depth understanding of OO is not required.

The controls allow you to display categorized data stored in a SAS table. “Categorized” means that the data is broken down into a small number of groups, for example, sales data broken down by region. It’s common to want to visually distinguish categories when displaying the data and that’s what these controls do.

WHY OBJECTS?

Each of the techniques shown here could be implemented with straight frame SCL. We use objects even though the coding is a bit more complicated. Objects are reusable. Once developed you can use them on any number of frames yet support just one version of the underlying code. In addition, someone can use these controls without understanding or even caring about the processes behind them.

A NOTE ON TERMINOLOGY

In this paper “widget” and “control” will be used interchangeably to mean an item you can drop on a SAS/AF frame. It can be argued that there is a distinction between a widget and a control, but this distinction is not interesting for our purposes here. The SASDATASET_C class used below, is not properly a widget, but a model used with a display widget. But again, I’m using the terminology loosely here to avoid technical overload.

WHAT ARE WE GOING TO DO?

I want to present a lightweight tutorial on how to create and use customized subclasses of the ORGCHART, TABBER, and SASDATASET_C classes (SAS table model). By lightweight, I mean that the code is not covered in detail. Experienced SCL programmers should be able to follow (and probably improve on) the programs. Others, interested only in using the objects do not have to worry about the coding details.

The subclasses created will automatically display a SAS table broken into categories based on the value of a column on the table. Each widget presents a different view of the table and categories.

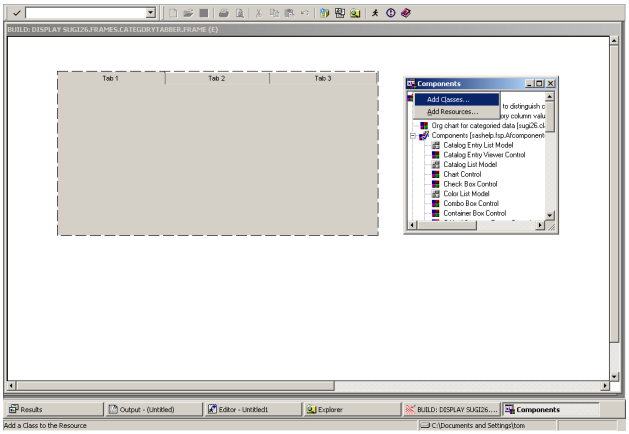
The programs used in this paper were developed and tested using SAS version 8.1 and require the SAS/AF product

COMMON USAGE STEPS

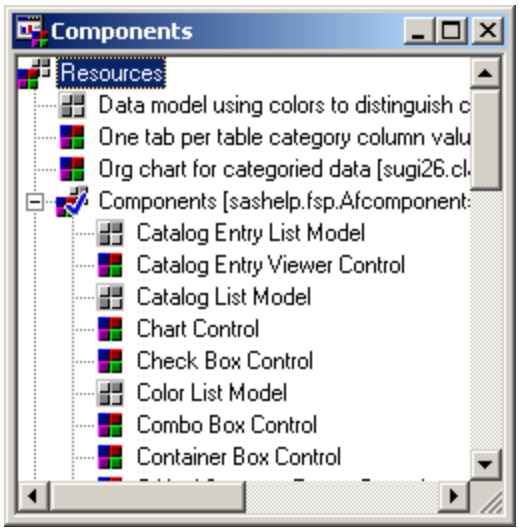
To use any of the three controls you must first compile the CLASS block SCL shown in the listings at the end of this paper. Use the SAVECLASS command to compile the methods and create the classes *categoryTabber*, *categoryModel*, and *categoryChart*. You will probably want to change the library, and possibly the catalog, names to something other than SUGI26.CLASSES. Edit the libref or catalog in the CLASS statement (see listings):

```
class
sugi26.classes.categoryTabber.class
extends sashelp.fsp.tabber.class / ( description=
 'One tab per table category column value' );
```

Once you have local names squared away, you need to make your new classes available to frames. When you create or open a frame entry, the Components window is also opened as shown below.



Right click on the Resources root node in the Components window and select Add Classes. Navigate to the location of your class entries and select them. Once added, they appear in your Components list as shown below.



OK, now you're ready to use the new category widgets.

CATEGORY TABBER

categoryTabber is a subclass of SASHELP.FSP.TABBER.CLASS. It shows one tab for each value of the category column as shown below, where the category column is "level2" and three categories are displayed: NEW YORK, LONDON, TOKYO.

level2: NEW YORK | level2: LONDON | level2: TOKYO

	LEVEL2	LEVEL1	LEVEL5	DEPTHEAD	LEVEL3
24	NEW YORK	International Ai	James Dargon	2	ADMIN
25	NEW YORK	International Ai	Natalie Besse	2	ADMIN
26	NEW YORK	International Ai	Emily P. Wallace	2	ADMIN
27	NEW YORK	International Ai	Deva K. Kumar	2	ADMIN
28	NEW YORK	International Ai	Veronica Delorge	1	ADMIN
29	NEW YORK	International Ai	Danielle Prost	2	ADMIN
30	NEW YORK	International Ai	Elizabeth Cousteau	2	ADMIN
31	NEW YORK	International Ai	Herbert J. Kirk	2	ADMIN
32	NEW YORK	International Ai	James H. Goodnight	1	ADMIN
33	NEW YORK	International Ai	Barrett R. Joyner	2	ADMIN
34	NEW YORK	International Ai	Sam Baker	1	SALES/MARK
35	NEW YORK	International Ai	Veronica Paulin	2	SALES/MARK
36	NEW YORK	International Ai	Patricia Smith	2	SALES/MARK
37	NEW YORK	International Ai	Camille Besseel	2	SALES/MARK
38	NEW YORK	International Ai	Elaine Dumas	2	SALES/MARK
39	NEW YORK	International Ai	Alan Picard	2	SALES/MARK
40	NEW YORK	International Ai	Jean Francois Dumas	2	SALES/MARK
41	NEW YORK	International Ai	Peter Caillon	2	SALES/MARK
42	NEW YORK	International Ai	Alan Bentz	2	SALES/MARK
43	NEW YORK	International Ai	Richard G. Roach	1	TECHN. SER
44	NEW YORK	International Ai	Danielle Biabaut	2	TECHN. SER
45	NEW YORK	International Ai	George H. Ruth	2	TECHN. SER

The categoryTabber class has two new attributes:

table - the name of the SAS table to display
categoryColumn - the name of the categorizing column on the table

and one overridden method:

_postInit - this is run after the frame and widget initialize

The CLASS block SCL to create this subclass is shown in Listing 1.

Drag the class from the resource list ("One tab per category…") and drop it on your frame. You'll probably want to resize it and maybe change some of the presentation attributes like border thickness. Don't worry about the number of tabs or their labels. In our example the widget is named TABBER.

Frame SCL to setup the TABBER widget is shown below.

```
dcl object objCatTab;
INIT:
   _frame_._getWidget( 'tabber', objCatTab );
   objCatTab.table = 'sashelp.company';
   objCatTab.categoryColumn = 'level2';
return;
```

We get the widget ID into variable objCatTab by calling the frame's _getWidget method. The next two lines name the table we want to display: SASHELP.COMPANY and the name of the category column on that table: LEVEL2. That's it. When run the COMPANY table is displayed with each value of LEVEL2 displayed on a separate tab as shown above.

categoryTabber takes advantage of the fact that you can create controls at run time. If you look at the CLASS block SCL you'll that a new pair of table viewer/model objects are created for each unique value of the category column. These are then assigned as the client objects for a tab.

CATEGORY MODEL

The *categoryModel* class is a subclass of SASHELP.CLASSES.SASDATASET_C. This is a SAS table model normally attached to the table viewer class SASHELP.CLASSES.TABLEVIEWER_C.CLASS. *categoryModel* is used with the table viewer to display a table. Each unique value of the category column is displayed with separate row color. As below:

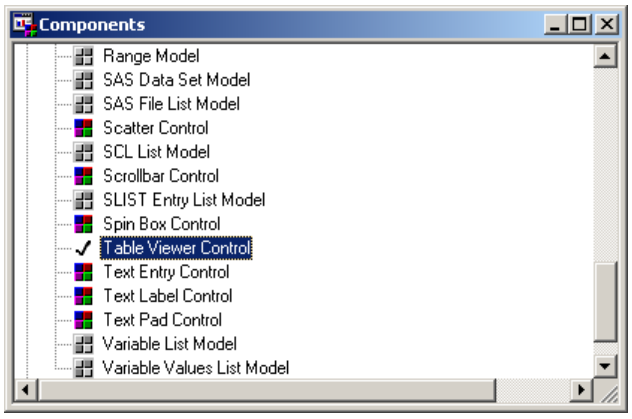
	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90
13	Louise	F	12	56.3	77
14	Mary	F	15	66.5	112
15	Philip	M	16	72	150
16	Robert	M	12	64.8	128
17	Ronald	M	15	67	133

The class uses the *table* and *categoryColumn* attributes as with the tabber class described above. Plus the following attributes:

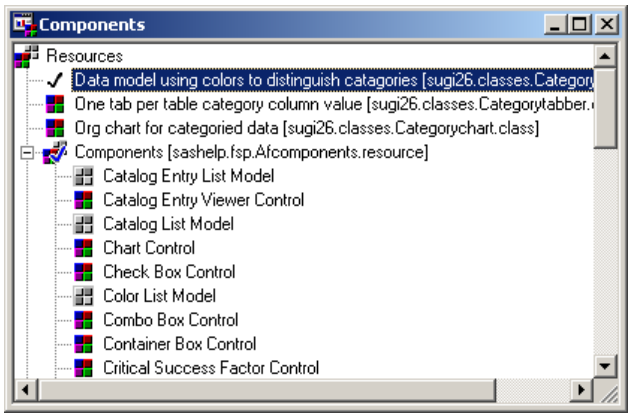
Colors – a blank delimited list of SAS colors to apply to category rows. This is optional, if not set the standard SAS color list is used.

MissingValueColor – the color to use when the value of a numeric category column is missing.

To use this control, open a frame and drag the Table viewer control onto your frame.



Then drag the *categoryModel* control onto the Table Viewer.



Frame SCL follows:

```
INIT
dcl object objTable;
_frame_._getWidget( 'table', objTable );
objTable.modelID.categoryColumn = 'sex';
objTable.modelID.colors={'orange','cyan','yellow'};
objTable.modelID.table = 'sashelp.class'
return;
```

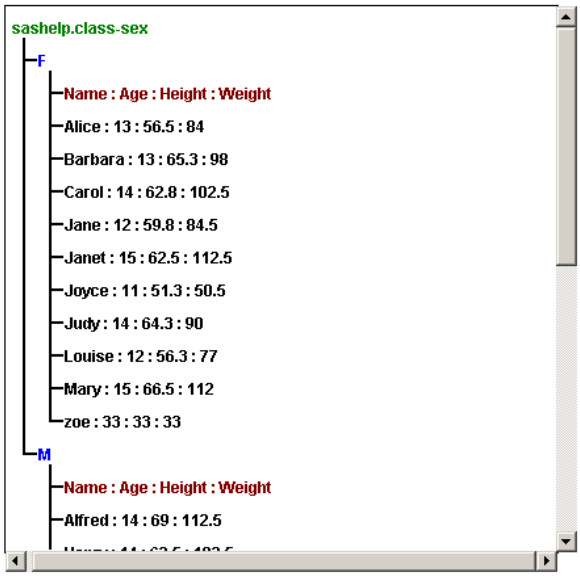
Here the SASHELP.CLASS table is displayed with SEX as the category column. Since three colors are named in the *colors* attribute, up to three unique SEX values could be displayed before recycling the colors, probably not a problem.

If you look at the class definition in Listing 2 you'll see how data vector object methods are called to control background color assignment. Also note that the row colors are assigned any time the *table* attribute is changed by overriding to *_setcamTable* method. This means you can assign a new table at runtime. This also means that you must set the *categoryColumn* and *colors* attributes before setting *table* so they affect the current display.

If you want contiguous groups of rows for each category, sort the table by the category column before displaying it.

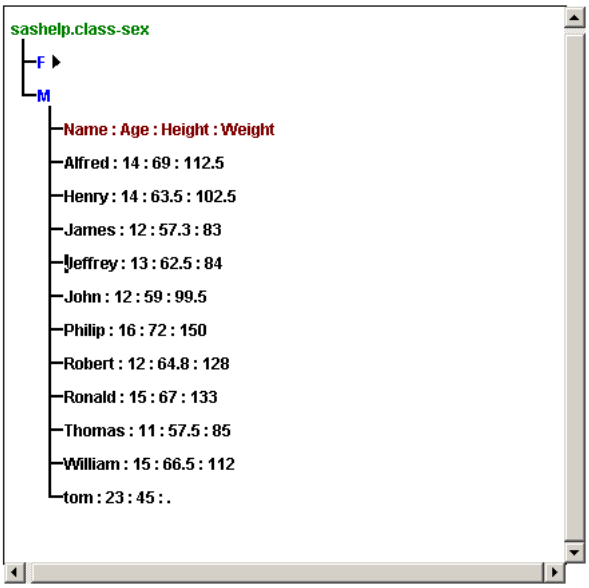
CATEGORY ORGCHART

The *categoryChart* class is a child of SASHELP.FSP.ORGCHART. It displays each category as a directory tree node and each row within that category as a child node:



The root node names the table and category column. Each child of the root is a category column value, here "F" and "M" from the SASHELP.CLASS table. Under the category nodes the first child node is a positional list of the remaining column names separated by a colon. The following nodes are the column values.

You can use any of the standard orgchart features such as node collapse, as shown below.



See the *orgchart* class documentation and object attributes window for other features. You'll probably want to change the chart style from hierarchical to directory in the attributes window.

categoryChart uses the following new attributes:

table – the name of the table to display
categoryColumn – name of the category column
rootColor – SAS color for the root node that displays table name and category column
categoryColor – SAS color for the category value node
headerColor – SAS color for the positional column name

header
dataColor – SAS color for the data nodes

All colors have defaults.

Frame SCL for sceen shown above is:

```
INIT:
dcl object objChart;
/* GET ID OF THE CHART CONTROL (WIDGET) */
_frame_.getWidget( 'chart', objChart );
/* SET CHART ATTRIBUTES */
objChart.table = 'sashelp.class';
objChart.categoryColumn = 'sex';
objChart.headerColor = 'red';
return;
```

LAST WORD

The need to represent categorized data is common in many data processing applications; business, research, or otherwise. When you have generalized presentation tools such as those shown here you can create data displays quickly and continue on to analysis. I hope these examples also demonstrate the power of objects for encapsulating the messy details of a process and facilitating reusability

LISTINGS

Listing 1 Create the categoryTabber Class

```
/* CLASS DEFINITION FOR categoryTabber */
class sugi26.classes.categoryTabber.class
   extends sashelp.fsp.tabber.class
   / ( description= 'One tab per table category column value' )
;
/* ATTRIBUTES */
   public char table
      / ( description= 'Name of the SAS table to display' )
   ;
   public char categoryColumn
      / ( description= 'Name of the table category column' )
   ;

/* METHODS */
   _postInit: public method
      / ( state = 'o' )
   ;
      /* LOCAL VARS */
      dcl object objTable;
      dcl object objModel;
      dcl list uniqueValues;
      dcl list attrs;
      dcl list region;
      dcl list tab;
      dcl list tabs;
      dcl num levels;
      dcl num dsid;
      dcl num i;
      dcl char( 200 ) valuec;
      dcl char( 200 ) msg;
      dcl char( 200 ) whereCondition;
      dcl char( 32 )  controlName;
      dcl char( 32 )  format;
      dcl char( 1 )   type;
      /* CALL SUPER */
      _super();
      /* VALIDATE TABLE, THEN OPEN */
      if not exist( table, 'data' ) then do;
         msg = sysmsg();
         put msg;
         return;
      end;
      dsid = open( table, 'i' );
      if not dsid then do;
         msg = sysmsg();
         put msg;
         return;
      end;
      /* VALIDATE CATEGORY COLUMN */
      if not varnum( dsid, categoryColumn ) then do;
         dsid = close( dsid );
         msg = sysmsg();
         put msg;
         return;
      end;
      /* GET UNIQUE VALUES OF THE CATEGORY COLUMN */
      uniqueValues = makelist();
      levels = 0;
      if lvarlevel( dsid, categoryColumn, levels, uniqueValues ) then do;
         msg = sysmsg();
         dsid = close( dsid );
         put msg;
         return;
      end;
      /* DETERMINE CATGORY COLUMN TYPE AND FORMAT */
      type = lowcase( vartype( dsid, varnum( dsid, categoryColumn ) ) );
      format = varfmt( dsid, varnum( dsid, categoryColumn ) );
      /* DONE WITH TABLE SO CLOSE IT */
      dsid = close( dsid );
      /* LISTS REQUIRED FOR RUNTIME CONTROL CREATION */
      attrs = makelist();
      region = makelist();
      /* EXPLICIT "_REGION_" IS REQUIRED TO AVOID THROWING USER */
      /* INTO BUILD MODE TO PLACE THE CONTROL.                  */
      attrs = setniteml( attrs, region, '_region_' );
      /* DUMMY REGION VALUES. */
      region = setnitemn( region, 1, 'ulx' );
      region = setnitemn( region, 1, 'uly' );
      /* THIS IS THE TABBER CONTROL'S "TABS" LIST */
      tabs = makelist();
      /* FOR EACH CATEGORY (LEVEL)... */
      do i = 1 to levels;
         /* CREATE A NEW TABLEVIEWER CONTROL AND ITS SAS TABLE MODEL */
         objTable = _neo_
            sashelp.classes.Tableviewer_c.class( attrs );
         objModel = _neo_ sashelp.classes.Sasdataset_c.class();
         /* ATTACH THE MODEL TO THE VIEWER */
         objTable.modelID = objModel;
         /* GET THE JUST-CREATED CONTROL NAME */
         controlName = objTable.name;
         /* ASSIGN THE TABLE TO DISPLAY FOR THE MODEL. THE WHERE= DSOPTION   */
         /* SEGREGATES BY CATEGORY. WE NEED TO DEAL WITH THE FORMATTED       */
         /* VALUES STORED IN THE THE UNIQUE VALUES LIST BY LVARLEVEL.        */
         valuec = getitemc( uniqueValues, i );
         select( type );
            when( 'n' ) do;
               if format = ' ' then whereCondition = categoryColumn || '=' ||
                  valuec;
               else whereCondition = 'put( ' || categoryColumn || ',' || format ||
                  ')=' || valuec;
            end;
            when( 'c' ) do;
               whereCondition = categoryColumn || '=' || quote( valuec );
            end;
         end;
         /* OK, GOT A WHERE CONDITION USE IT WITH THE 'TABLE' ASSIGNMENT. */
         objModel.table = table || '( where=(' || whereCondition || ') )';
         /* INFO LIST FOR THIS CATEGORY'S TAB */
         tab = makelist();
         /* TAB ID IS SEQUENCIAL NUMBER OF THIS CATEGORY */
         tab = setnitemn( tab, i, 'id' );
         /* BUILD UP LABELTEXT FROM COLUMN NAME AND VALUE */
         tab = setnitemc( tab, categoryColumn || ': ' || valuec, 'label' );
         /* TAB'S CLIENT IS THE SET BY NAME NOT ID. */
         tab = setnitemc( tab,  controlName, 'client' );
         /* OK, THIS TAB IS SET UP FOR THE CURRENT CATEGORY. INSERT IT */
         /* INTO THE LIST OF ALL TABS.                                 */
         tabs = insertl( tabs, tab, -1 );
      end;
      /* ASSIGN THE TABS DEFINITION LIST  */
      _setTabs( tabs );
      /* FORCE DISPLAY OF THE FIRST TAB */
      _setActiveTab( 1 );
      /* CLEAN UP */
      uniqueValues = dellist( uniqueValues );
      tabs = dellist( tabs, 'y' );
   endmethod;
endclass;
```

Listing 2 – categoryModel Class

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* CLASS DEFINITION FOR categoryModel                            */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class sugi26.classes.categoryModel.class
   extends sashelp.classes.Sasdataset_c.class
   / ( description= 'Data model using colors to distinguish categories' )
;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* ATTRIBUTES                                                    */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   public char( 41 ) table
      / ( state= 'o',  setCam= 'setcamTable' )
   ;
   public list colors
      / ( description= 'List of colors used to distinguish categories' )
   ;
   public char( 32 ) categoryColumn
      / ( description= 'Name of the table category column' )
   ;
```

```
   public char( 32 ) missingValueColor
      / ( description= 'Color for numeric missing values'
          ,initialValue= 'red' )
   ;
   protected char( 1 ) categoryColumnType
      / ( description= 'Type of the table category column' )
   ;
   protected char( 32 ) categoryColumnFormat
      / ( description= 'Format of the table category column' )
   ;
   protected list categoryColors
      / ( description= 'List item name is color, value is category column value' )
   ;

   /* IF WE DEFAULT TO THE SYSTEM COLORS LIST THESE COLORS WON'T */
   /* BE USED AS BACKGROUND HIGHLIGHT BECAUSE THEY WOULD OBSCURE */
   /* FOREGROUND TEXT.                                           */
   public list dontUseColors
      / ( InitialValue = { 'black', 'sysfore' }
          ,description= 'System colors to ignore' )
   ;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                             */
/* METHODS                                                     */
/*                                                             */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   /*                                                         */
   /* _setcamTable                                            */
   /*                                                         */
   /* THIS METHOD IS RUN WHEN THE 'TABLE' ATTRIBUTE IS SET    */
   /*                                                         */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   _setcamTable: protected method
      table : input : char( 41 )
      return= num
      / ( state= 'o' )
   ;
     /* LOCAL VARS */
     dcl sashelp.classes.colorlist_c.class objColors;
     dcl num dsid;
     dcl char( 200 ) msg;
     dcl num levels;
     dcl list uniqueValues;
     dcl num i;
     dcl num colorNumber;
     dcl num numberOfColors;
     dcl char( 16 ) colorName;
     dcl char( 200 ) valuec;

     /* MAKE SURE CATEGORY COLUMN IS SET */
     if categoryColumn = ' ' then do;
        put 'categoryColumn is blank.';
        return( 1 );
     end;

     /* VALIDATE, THEN OPEN TABLE */
     if not exist( table, 'data' ) then do;
        put table 'not found.';
        return( 1 );
     end;
     dsid = open( table, 'i' );
     if not dsid then do;
        msg = sysmsg();
        put msg;
        return( 1 );
     end;

     /* MAKE SURE CATEGORY COLUMN IS ON THE TABLE */
     if not varnum( dsid, categoryColumn ) then do;
        put categoryColumn 'is not on' table;
        dsid = close( dsid );
        return( 1 );
     end;

     /* GET TYPE AND FORMAT (FOR NUMERIC) OF CATEGORY COLUMN AND STORE AS
ATTRIBUTES*/
     categoryColumnType = lowcase( vartype( dsid, varnum( dsid, categoryColumn ) )
);
     if categoryColumnType = 'n' then do;
        categoryColumnFormat = varfmt( dsid, varnum( dsid, categoryColumn ) );
        if categoryColumnFormat = ' ' then categoryColumnFormat = 'best.';
     end;

     /* GET UNIQUE FORMATTED VALUES (ALWAYS CHARACTER) OF CATEGORY COLUMN */
     levels = 0;
     uniqueValues = makelist();
     if lvarlevel( dsid, categoryColumn, levels, uniqueValues ) then do;
        msg = sysmsg();
        dsid = close( dsid );
        uniqueValues = dellist( uniqueValues );
        put msg;
        return( 1 );
     end;
     dsid = close( dsid );

     /* IF THE COLORS LIST ATTRIBUTE IS EMPTY USE THE SYSTEM COLORS LIST */
     if listlen( colors ) < 1 then do;
        objColors = _new_ sashelp.classes.colorlist_c();

        objColors.basicColorsDisplayed = 'Yes';
        colors = copylist( objColors.items, ' ', colors );
        objColors._term();

        /* REMOVE PROBLEM COLORS NAMED IN THE dontUseColors LIST */
        /* AND THE MISSING VALUE COLOR.                          */
        do i = listlen( colors ) to 1 by -1;
           colorName = getitemc( colors, i );
           if searchc( dontUseColors, colorName, 1, 1, 'y' ) then
              colors = delitem( colors, i );
           if lowcase( colorName ) =  lowcase( missingValueColor ) then
              colors = delitem( colors, i );
        end;
     end;

     /* GET TOTAL NUMBER OF COLORS */
     numberOfColors = listlen( colors );

     /* WARNING MESSAGE FOR FEWER COLORS THAN LEVELS */
     if numberOfColors < listlen( uniqueValues ) then do;
        put 'WARNING: There are more unique values of' categoryColumn
          'than colors. Colors will be reused.';
      end;

     /* LOAD LIST OF COLORS (ITEM NAME) AND CORRESPONDING VALUE (ITEM VALUE) */
     do i = 1 to listlen( uniqueValues );

        /* ROTATE THROUGH COLOR LIST IN CASE THERE ARE MORE CATEGORY VALUES */
        /* THAN COLORS.                                                    */
        colorNumber = mod( i, numberOfColors );
        if colorNumber = 0 then colorNumber = numberOfColors;

        /* GET NEXT AVAILABLE COLOR FROM THE LIST */
        colorName = getitemc( colors, colorNumber );
        valuec = getitemc( uniqueValues, i );
        categoryColors = insertc( categoryColors, valuec, -1, colorName );
     end;

     /* CLEAN UP */
     uniqueValues = dellist( uniqueValues );

    /* CALL SUPER */
     return( _super( table ) );
   endmethod;

   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   /*                                                         */
   /* _getData                                                */
   /*                                                         */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   _getData: public method
      vecid :object   numcols :num
      / ( state = 'o', signature = 'n' )
   ;

      /* LOCAL VARS */
      dcl char( 200 ) valuec;
      dcl num         valuen;
      dcl char( 24 )  color;
      dcl num         i;

      /* CALL SUPER */
      _super( vecid, numcols );

      /* BASED ON CATEGORY COLUMN TYPE LOOK UP THE COLOR FOR THIS */
      /* CATEGORY COLUMN VALUE.                                   */
      select( categoryColumntype );
         when( 'c' ) do;
            _getColumnText( categoryColumn, valuec );
            i = searchc( categoryColors, valuec );
            color = nameitem( categoryColors, i );
         end;
         when( 'n' ) do;
            _getColumnValue( categoryColumn, valuen );

            /* FIRST CHECK FOR MISSING VALUE */
            if valuen = . then color = missingValueColor;
            else do;

               /* WE NEED TO CONVERT TO THE FORMMATTED VALUE STORED IN THE */
               /* categoryColors LIST.                                     */
               i = searchc( categoryColors, putn( valuen, categoryColumnFormat ) );
               color = nameitem( categoryColors, i );
            end;
         end;
      end;

      /* FOR EACH COLUMN SET THE BACKGROUND COLOR */
      do i = 1 to numcols;
         vecid._setIndex( i );
         vecid._setBackgroundColor( color );
      end;

   endmethod;
endclass;
```

Listing 3– categoryChart Class

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* CLASS DEFINITION FOR categoryChart                            */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class sugi26.classes.categoryChart.class
   extends sashelp.fsp.orgchart.class
   / ( description= 'Org chart for categoried data' )
;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* ATTRIBUTES                                                    */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   public char( 41 ) table
      / ( description= 'Name of the table to display' )
   ;
   public char( 32 ) categoryColumn
      / ( description= 'Name of the table category column' )
   ;
   public char( 24 ) rootColor
      / ( description= 'Foreground color of the chart root node'
          ,initialValue= 'green' )
   ;
   public char( 24 ) categoryColor
      / ( description= 'Foreground color of the category column values'
          ,initialValue= 'blue' )
   ;
   public char( 24 ) headerColor
      / ( description= 'Foreground color of the category column values'
          ,initialValue= 'cyan' )
   ;
   public char( 24 ) dataColor
      / ( description= 'Foreground color of the data nodes'
          ,initialValue= 'black' )
   ;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* METHODS                                                       */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   /*                                                           */
   /* _repopulate                                               */
   /*                                                           */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   _repopulate: public method
      optional=
         idType  : input : num
         dataID  : input : num
         mapList : input : num
         nodeID  : input : num
         append  : input : num
      / ( state= 'o',  signature= 'n' )
   ;
      /* LOCAL VARS */
      dcl char( 200 ) msg;
      dcl num levels;
      dcl list uniqueValues;
      dcl num i;
      dcl num j;
      dcl num dsid;
      dcl char( 200 ) whereCondition;
      dcl char( 1 ) type;
      dcl char( 24 ) format;
      dcl char( 32 ) name;
      dcl list chartList;
      dcl list rootChildren;
      dcl list categoryList;
      dcl list categoryChildren;
      dcl list categoryItems;
      dcl char( 40 ) displayText;
      dcl char( 200 ) line;
      dcl char( 200 ) header;
      dcl num numVars;

      if categoryColumn = ' ' then do;
         put 'categoryColumn is blank.';
         return;
      end;

      if not exist( table, 'data' ) then do;
         put table 'not found.';
         return;
      end;
      dsid = open( table, 'i' );
      if not dsid then do;
         msg = sysmsg();
         put msg;
         return;
      end;

      if not varnum( dsid, categoryColumn ) then do;
         put categoryColumn 'is not on' table;
         dsid = close( dsid );
         return;
      end;

      type = lowcase( vartype( dsid, varnum( dsid, categoryColumn ) ) );
      if type = 'n' then format = varfmt( dsid, varnum( dsid, categoryColumn ) );

      numVars = attrn( dsid, 'nvars' );

      levels = 0;
      uniqueValues = makelist();
      if lvarlevel( dsid, categoryColumn, levels, uniqueValues ) then do;
         msg = sysmsg();
         put msg;
         return;
      end;

      line = ' ';
      do i = 1 to numvars;
         name = varname( dsid, i );
         if lowcase( name ) = lowcase( categoryColumn ) then continue;
         if header = ' ' then header = name;
         else header = header || ' : ' || name;
      end;

      chartList = makelist();

      chartList = setnitemc( chartList, table || '-' || categoryColumn, 'text' );

      chartList = setnitemc( chartList, rootColor, 'foreground_color' );

      rootChildren = makelist();
      chartList = setniteml( chartList, rootChildren, 'children' );

      do i = 1 to listlen( uniqueValues );

         displayText = left( getitemc( uniqueValues, i ) );

         select( type );
            when( 'c' ) do;
               whereCondition = categoryColumn || '=' || quote( displayText );
            end;
            when( 'n' ) do;
               if format = ' ' then whereCondition = categoryColumn || '=' ||
                  displayText;
               else whereCondition = 'put( ' || categoryColumn || ',' || format ||
                  ')=' || displayText;
            end;
         end;

         categoryList = makelist();
         rootChildren = insertl( rootChildren, categoryList, -1 );

         categoryList = setnitemc( categoryList, displayText, 'text' );
         categoryList = setniteml( categoryList, makelist(), 'children' );
         categoryChildren = getniteml( categoryList, 'children' );

         categoryList = setnitemc( categoryList, categoryColor, 'foreground_color');

         categoryItems = makelist();
         categoryChildren = insertl( categorychildren, categoryItems, -1 );
         categoryItems = setnitemc( categoryItems, header, 'text' );

         categoryItems = setnitemc( categoryItems, headerColor, 'foreground_color');

         if where( dsid, whereCondition ) not in( 0, %sysrc( _swwrep ) ) then do;
            msg = sysmsg();
            dsid = close( dsid );
            put msg;
            return;
         end;

         do while( not fetch( dsid ) );

            categoryItems = makelist();
            categoryChildren = insertl( categorychildren, categoryItems, -1 );

            line = ' ';

            do j = 1 to numVars;

               if lowcase( varname( dsid, j ) ) = lowcase( categoryColumn ) then
continue;
               select( lowcase( vartype( dsid, j ) ) );
                  when( 'c' ) do;
                     displayText = getvarc( dsid, j );
                  end;
                  when( 'n' ) do;

                      displayText = getvarf( dsid, j );
                  end;
               end;
               if line = ' ' then line = left( displayText );
               else line = line || ' : ' || left( displayText );
            end;

            categoryItems = setnitemc( categoryItems, left( trim ( line ) ),'text');
```

```
          categoryItems = setnitemc( categoryItems, dataColor,
'foreground_color');

        end;
      end;

      dsid = close( dsid );
      uniqueValues = dellist( uniqueValues );

      _super( 1, chartList, mapList, nodeID, append );

      chartlist = dellist( chartlist, 'y' );
   endmethod;
endclass;
```

REFERENCES

Documentation for SASHELP.FSP.TABBER.CLASS is in the Version 6.12 and up online help.

For SASHELP.FSP.ORGCHART.CLASS see the Organizational Chart Class section in *SAS/AF Software: FRAME Class Dictionary*

SASHELP.CLASSES.SASDATASET_C.CLASS is covered in the Version 8 online help.

ACKNOWLEDGMENTS

Takes to all the SAS technical support folks who explained, researched, and commiserated on my SAS programming problems over so many years. You guys are the best!

CONTACT INFORMATION

You can contact me at:

Tom Miron
Miron InfoTec, Inc.
118 S. Hancock St.
Madison, WI 53703

608 255-3531

mironta@aol.com