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.
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.

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.

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:

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:

```sas
INIT
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
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 char      / ( 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 attr;
   dcl list region;
   dcl list tab;
   dcl list tabs;
   dcl sun levels;
   dcl sun dsid;
   dcl sun i;
   dcl char( 200 ) values;
   dcl char( 200 ) msg;
   dcl char( 100 ) whereCondition;
   dcl char( 32 ) controlName;
   dcl char( 32 ) format;
   dcl char( i ) type;
   /* CALL SUPER */
   super();
   /* VALIDATE TABLE, THEN OPEN */
   if not exist( table, 'data' ) then do;
      msg = sysmsg();
      dais = close( dais );
      put msg;
      return;
   end;
   dais = open( table, 'l' );
   if not dais then do;
      msg = sysmsg();
      put msg;
      return;
   end;
   /* VALIDATE CATEGORY COLUMN */
   if not varnum( dais, categoryColumn ) then do;
      dais = close( dais );
      msg = sysmsg();
      put msg;
      return;
   end;
   /* GET UNIQUE VALUES OF THE CATEGORY COLUMN */
   uniqueValues = makelist();
   levels = 0;
   if varlevel( dais, categoryColumn, levels, uniqueValues ) then do;
      uniqueValues = dellist( uniqueValues, 'y' );
   end;
   /* FETCH THE COLUMN VALUES */
   whereCondition = makelist( uniqueValues, 'y' );
   /* BUILD LAYOUT */
   tab = makelist( whereCondition, 'y' );
   /* FORCE DISPLAY OF THE FIRST TAB */
   setActiveTab( 1 );
   /* CLEAN UP */
   uniqueValues = dellist( uniqueValues );
   tabs = dellist( tabs, 'y' );
   end;
endmethod;
endclass;```
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. */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* METHODS */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* * * * * * * * * * * * * * * * */

/* IF THE COLUMNS LIST ATTRIBUTE IS EMPTY USE THE SYSTEM COLORS LIST */
if listlen( colors ) < 1 then do;
   objColors = _new_ sashelp.classes.colorlist_c();
   objColors.basicColorsDisplayed = 'Yes';
   objColors.colors = copylist( objColors.items, '', colors );
   objColors._term();
end;

/* REMOVE PROBLEM COLORS NAMED IN the dontUseColors LIST */
/* AND THE MISSING VALUE COLOR. */
do i = 1 to listlen( colors ) by 1;
   if searchc( dontUseColors, categoryName, i, ',' | 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 );
   colorNumber = mod( i, numberOfColors );
   if colorNumber = 0 then colorNumber = numberOfColors;
   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 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* LOCAL VARS */
dcl object vecid;      dcl num numcols;      dcl char( 200 ) valuec;      dcl num         valuen;

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

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

/* FIRST CHECK FOR MISSING VALUE */
if valuen = . then color = missingValueColor;
else do;
   /* WE NEED TO CONVERT TO THE FORMATTED VALUE STORED IN THE */
   /* categoryColors LIST. */
   i = searchc( categoryColors, put( valuen, categoryColumnFormat ) );
   color = nameitem( categoryColors, i );
end;
end;

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

Information Visualization
Listing 3– categoryChart Class

```plaintext
/* CLASS DEFINITION FOR categoryChart */
   /* */
   /* */
   /* CLASS categoryChart */
   extends sashelp.crp.orgchart.class
   / (description = 'Org chart for categorized data' );
   /* */
   /* */
   public char(4) 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 */
   /* */
   /* public */
   main()
   _repopulate();
   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;
   line = ' ';
   do i = 1 to numVars;
   name = varname( dsid, i );
   if lowcase(name) = lowcase(.categoryColumn ) then continue;
   header = ' ';
   else header = header || ' : ' || name;
   end;
   chartlist = makelist();
   chartlist = setitem( chartlist, table || '-' || categoryColumn, 'text' );
   chartlist = setitem( chartlist, rootColor, 'foreground_color' );
   rootChildren = makelist();
   chartlist = setitem( 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 = setitem( categoryList, rootColor, 'foreground_color' );
   categoryList = setitem( categoryList, categoryColor, 'foreground_color' );
   categoryItems = makelist();
   categoryChildren = insertl( categoryItems, children );
   categoryChildren = setitem( categoryChildren, 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( categoryItems, children );
   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;
   if line = ' ' then line = left( displayText )
   else line = line || ' : ' || left( displayText )
   end;
   categoryItems = setitem( categoryItems, left( trim( line ) ), 'text' );
   end;
   end;
   close( dsid );
   return;
   end;
```

Information Visualization
categoryItems = setitem( 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