Paper 091-29

Advanced Programming Techniques for Working with Color in SAS® Software

Perry Watts, Independent Consultant, Elkins Park, PA

Abstract

Programming issues associated with color are defined by "going under the hood" of the SUGI 28 paper entitled *Working with RGB and HLS Color Coding Systems in SAS® Software*. Since colors are expressed as hexadecimals in SAS, the numbering system is reviewed at the outset. Next HEX formats are described along with a core set of V8 conversion macros. The core macros use the HEX formats to recast color codes in different numbering and color systems. For completeness, the SAS-supplied Version 9 color utility macros that are similar to the V8 core macros are also described.

An effort is made to simplify source code that is described in the paper. For example, lists of macro variables representing individual colors in a chart can be replaced by a single format in Version 8.2 that is invoked from a DEFINE statement in PROC REPORT. The procedure is exercised in ODS with output to an HTML file. Similarly, the Annotate %BAR macro that works with color constants inside a macro is transformed to the %_BAR macro that manipulates color variables directly within a data step.

Besides showing how source code can be simplified, another goal of the paper is to extend programming concepts for color to issues that confront the BASE SAS programmer. For example, the discussion of the hexadecimal numbering system is generalized to show how hexadecimals show up in all types of applications. Similarly the color conversion macros bring up more general issues related to scope in the SAS macro language, and the discussion about colors as constant literals is extended to include data set assignments in the OUTPUT statement. Finally the versatility of the macro function is demonstrated by nesting V8 core macro calls and by using them as labels in a format.

The Hexadecimal Numbering System

From Decimal to Hexadecimal and Back Again

If you download the SUGI 29 logo as a JPG file to a PC, you can easily obtain RGB code values for the shade of red in Microsoft® Paint (*right-mouse click: Select color with eye drop: Edit Colors: Define Custom Colors*). Unfortunately, however, the values are in decimal, and SAS only understands hexadecimal. A tedious method for handling the conversion would be to manually translate each color component into its HEX value. The translation is illustrated in Figure 1 for decimal 213 assigned to the red component in SUGI red.

Figure 1. Manual conversions between the hexadecimal and decimal number systems are illustrated for decimal 213.

RGB: **213**, 29, 41

Decimal	Hexadecimal
$10^2\ 10^1\ 10^0$ **2 1 3**	$16^1\ 16^0$ **D 5**
$16\overline{)213}^{13(D)}$ $R=5$, $1\overline{)5}^{5(5)}$ $R=0$ = **D5**	$13(D)\times16^1 = 208 + 5\times16^0 = 5$ **= 213**
`HexNum=put(213, hex2.);` = **'D5'**	`decNum=input('D5',hex2.);` = **213**

As seen from the table in Figure 1, numbers are defined as powers of 10(decimal) or 16(hexadecimal). Only Base 16, however, is used for both conversions in the second row. To go from decimal to hexadecimal, a number is divided by powers of 16. For decimal 213, 16^1 is the starting point because the next higher power, 16^2 (256), is too large. The quotient 13 becomes the first hexadecimal digit 'D'. The remainder of the first division, 5, is then divided by 16^0 or 1 leaving a final remainder of zero. The quotient 5 in the second division is also represented in hexadecimal as '5'. To reverse the process hexadecimals are multiplied powers of 16, so 'D'(13) X 16^1 equals 208 and 208 + 5 X16^0(1) equals 5 for a total of 213, the original value.

Bypass Manual Translations with HEX Informats and Formats

To work with numeric conversions programmatically, you need to know that hexadecimals are always defined as character in SAS. RGB color constants are written as **'**CXrrggbb**'** where *rr*, *gg*, and *bb* are two-digit hexadecimal numbers for color components that range in value from '00' to 'FF' (255 decimal). SUGI red, for example, is expressed as 'CXD51D29'. HLS codes are written as **'**Hhhhlss**'** where hue (*hhh*), a three-digit hexadecimal, ranges from '000' to '168' (360° decimal), and both lightness (*ll*) and saturation (*ss*) go from '00' to 'FF'. The HLS code for SUGI red, based on the Tektronix standard, is 'H07479C2'. Other hexadecimal constants you may be familiar with are ASCII values '00'X for the null character and '09'X for the tab character. Note the difference: for color '00' in RGB means black, or the absence of color, whereas '00'X references the NULL *character*.

Because hexadecimals are character, conversions can be accommodated by formats rather than pre-defined functions. In fact two formatting combinations used in conjunction with the INPUT and PUT functions provide all the functionality required for working with hexadecimals in SAS. The HEX*n* format used with the PUT function shown in the third row of the table in Figure 1 converts a decimal to a hexadecimal, whereas the same format used with the INPUT function does just the opposite: converts a hexadecimal to its decimal equivalent. The value *n* appended to the HEX format or informat always references the width of the hexadecimal - NOT the decimal. Therefore the source hexadecimal value 'D5' is properly handled by an HEX2 informat even though the result is a three-digit decimal, 213. It should also be noted that the HEX format accommodates decimal fractions by truncating them prior to conversion. In the V8 core conversion macros truncation is overridden by rounding decimals before applying the HEX format. The rationale for the override is explained later in the paper. Hexadecimal fractions, on the other hand, are disregarded in both the RGB and HLS coding systems.

The **$**HEX informat|format pair is reserved for ASCII character conversions in SAS. Again *informat* and *format* are inverses of each other. Here is how to get the hexadecimal value for the character 'A':

```
numberHex=put('A',$hex2.);
```

Since 'A' and '41' assigned to NUMBERHEX are both characters, the format width accommodates the return value assigned to NUMBERHEX. Conversely, the input function returns a letter 'A' for an ASCII hexadecimal, but this time the width references the INPUT *source* not the result:

```
letterHex=input('41',$hex2.);
```

The RANK and BYTE functions for decimals play the same role as the $HEX informat|format pair:

```
DecNum=rank('A');       *returns a numeric value of 65;
LetterA=byte(DecNum);   *returns an 'A';
```

Hexadecimal Applications in SAS Software

While hexadecimal numbers play a relatively minor role in SAS software, they come in very handy for completing selected programming assignments. In this section, we look at how they are used in the DELIMITER option of the INFILE statement, the FORMCHAR option of multiple SAS procedures, and the elimination of redundant labels from multiple-plot displays generated from BY-GROUP processing in the GPLOT procedure. Additional applications not covered include initializing and terminating SAS/GRAPH print jobs and using '09'X (TAB) or '0A'X (LINEFEED) to communicate with EXCEL via DDE (Dynamic Data Exchange). In the applications just mentioned, hexadecimals represent ASCII numbers. This means that there should always be an even count of digits, because two consecutive digits resolve to one character. Again, hexadecimals for color are not related to ASCII codes, and SAS software is not alone in using hexadecimals for color codes. The HTML code "#D51D29" for SUGI red contains the same digits as the SAS RGB code.

<u>The DELIMITER Option</u>

The DELIMITER option is used for demarcating variables in SAS list input. Note the comma delimited file below can easily accommodate missing values in the CARDS statement:

```
data testResults;
  infile cards delimiter=',';
  input score1 score2 score3;
  cards;
91,87,92
48,88, ,
28, ,92
, ,70,99
run;
```

If you have a tab-delimited external file, the delimiter must be defined as `'09'X` rather than as an invisible tab character between two quotes (' ').

The FORMCHAR Option in Multiple SAS Procedures

The FORMCHAR option can be invoked directly in the CALENDAR, CHART, FREQ, PLOT, and TABULATE procedures. It is also available as a system option for PROC REPORT, and you can permanently reset its value by editing CONFIG.SAS (V6) or SASV8.CFG. In the following application of PROC TABULATE a string of hexadecimal numbers is assigned to FORMCHAR:

```
proc tabulate data=combine
 formchar='7C2D2D2D2D7C2B7C2D2D2D'X;
  title1 'Exhibit A';
  class W X Y Z;
  table W=' ' X='xxxx' Y='yyyy',
        Z=' '*(n='#'*F=comma8. PctN<W X Z>='%'*F=8.1);
run;
```

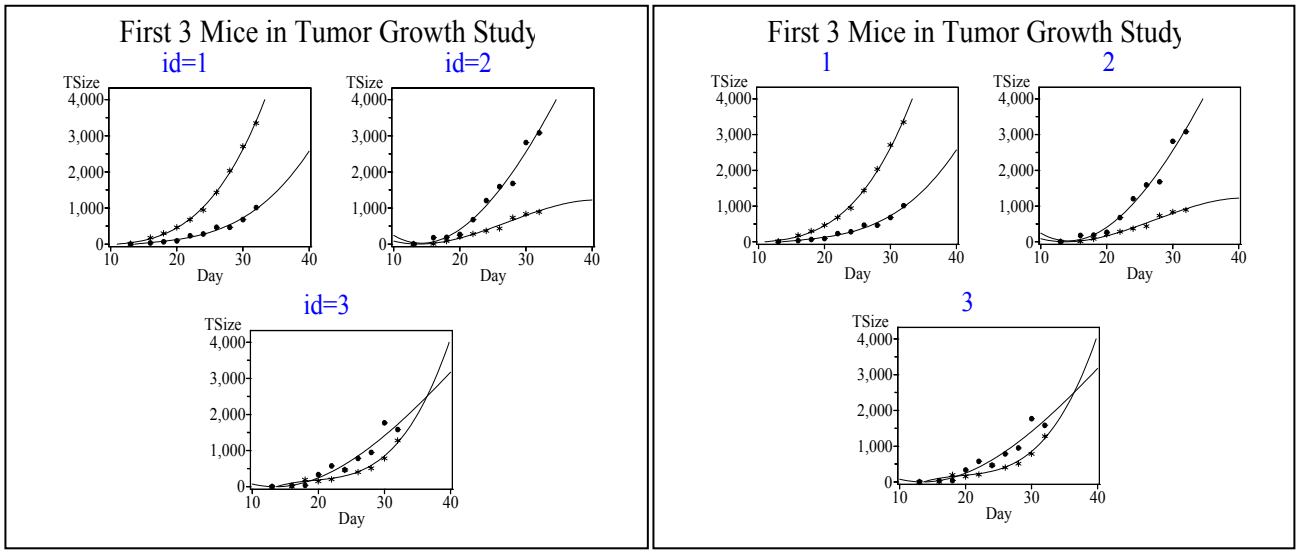
If digits in the FORMCHAR argument reference visible characters, then characters should be used instead. The $HEX format applied below returns a commonly used character string:

```
Data;
  length formStr $11;
  formStr = input('7C2D2D2D2D7C2B7C2D2D2D',$hex22.); *EVEN NUMBER;
  put 'formStr = ' formStr;                          *=  |----|+|--- ;
run;
```

Eliminating Redundant Labels from Multiple-Plot Displays that use By-Group Processing

To eliminate all by-group headings from a multiple plot display, specify HBY=0 in your GOPTIONS statement. However, to get rid of the variable label segment in the BY group headings, label the by-group variable with a NULL ('00'X) in the graphics procedure. The consequence of such labeling is observed in the second graph in Figure 2. The graphs in Figure 2 are reproduced from *Multiple-Plot Displays: Simplified with Macros*.

Figure 2. Using '00'X in a label statement for the by-group variable in a multiple-plot display removes the label portion of the heading.

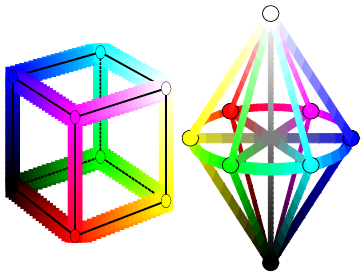


The Core Set of V8 Color Conversion Macros

Conversion macros are needed for supporting RGB and HLS color spaces, charts, and scales. Pictorial definitions for the three target structures appear in Figure 3 below. For additional displays, please see the SUGI 28 paper referenced in the abstract. It should be noted that the macros are essential for building any application that makes heavy use of color in SAS, not just the three structures in Figure 3. The V8 macros pre-date the SAS Version 9 color utility macros that also perform basic *numeric* and *color system* conversions. The Version 9 macros are described later on.

Figure 3. Pictorial definitions for the color space, chart and scale.

Color spaces are 3-D structures that show how a gamut of codes is mapped in a color system. The RGB space is a cube whereas the HLS space is double-ended cone.

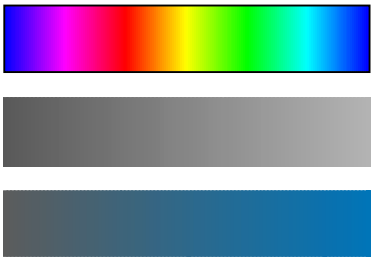


Color charts show colors as discrete entities in tabular format. Here are the unique SAS predefined colors and a digitized listing of a shade of blue.

CX00FFFF	CX3883A8	CX4F4F4F	CX700000
CXC0FF81	CXE06090	CXE0A860	CXE8D898
CXFF0055	CXFF00FF	CXFF6060	CXFFAA00

CX001928	CX002032	CX00263C	CX002C46	CX003350	CX00395A
CX003F64	CX00466E	CX004C78	CX005282	CX00598C	CX005F96
CX0065A0	CX006CAA	CX0072B4	CX0078BE	CX007FC8	CX0085D2
CX008BDC	CX0092E6	CX0098F0	CX009EFA	CX05A3FF	CX0FA7FF
CX19ABFF	CX23AEFF	CX2DB2FF	CX37B6FF	CX41B9FF	CX4BBDFF
CX55C1FF	CX5FC4FF	CX69C8FF	CX73CCFF	CX7DCFFF	CX87D3FF
CX91D7FF	CX9BDAFF	CXA5DEFF	CXAFE2FF	CXB9E5FF	CXC3E9FF
CXCDEDFF	CXD7F0FF	CXE1F4FF	CXEBF8FF	CXF5FBFF	CXFFFFFF

Color scales represent colors along a continuum. Here are hue, lightness and saturation scales built with HLS codes.



Motivation for Numeric Conversions

Because we think in decimal, SAS programs are much more readable if colors are referenced in decimal. Furthermore, since colors do not exist in isolation, ratios are needed for understanding relationships among them. That half of 'FF', for example, is '80' is not intuitive, whereas 256 divided by 2 is a simple calculation. The listing in Table 1 emphasizes the need to reference colors as decimals. It is derived from a series of un-annotated PATTERN statements in Erick Tilanus' otherwise excellent SUGI 28 paper *%Mondriaan: Presenting 3D information in 2D*. The original RGB Hex codes are listed in column 2. Commentary in column 3 requires familiarity with color and the hexadecimal numbering system, but familiarity is not sufficient for calculating the ratios in Column 5. The necessary information is supplied by the decimals in Column 4.

Be sure to look at the cube in Figure 3 and the cube faces in Figure 4 while inspecting Table 1. Key principles are:

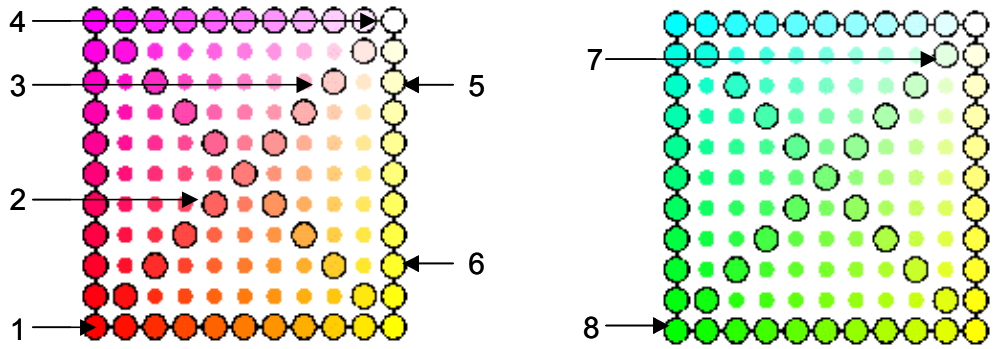
- Black is positioned at the origin in the RGB cube with Red, Green and Blue serving as the X, Y, and Z-axes respectively. Again code values range from '00' to 'FF' (hex) 0 to 255 (decimal) and 0 to 1 (coordinate).
- The component or components with the highest value determines a color's basic shade. Yellow has maximum values for both red and green components.
- If any component in a color code is equal to 0 or 255 the color is located on the RGB cube's surface.

Table 1: Listing of Related Colors in Eight Pattern Statements

#	PATTERN RGB Hex Code	Observations from HEX Code Inspection	RGB Decimal Code	Ratios from Decimal Code Inspection	Color
1	CXFF0000	True Red.	255,0,0	100% Red	
2	CXFF6666	Lighter red	255,102,102	102=2/5ths of 255	
3	CXFFCCCC	Even lighter red	255,204,204	204=4/5ths of 255	
4	CXFFFFFF	Lightest red possible -i.e. white	255,255,255	100% White	
5	CXFFFFCC	Lighter yellow	255,255,204	4/5ths up the yellow edge	
6	CXFFFF44	Darker yellow	255,255,68	68 is 1/3rd of 204 (#5)	
7	CXDDFFDD	Lighter green	221,255,221	221=~9/10ths of 255	
8	CX00FF00	True (darker) green	0,255,0	100% green	

Given the ratios in Table 1 it is now possible to identify colors with a high degree of precision in the cube faces shown in Figure 4.

Figure 4. Two of the six faces associated with the RGB cube in Figure 3. The numbers reference PATTERN statements from Table 1. The ratios listed in Table 1 make color identification possible.



Decimal equivalents for color codes are also needed for additional reasons. Since axes-coordinates and color values have a one-to-one relationship with each other in RGB space, SAS macros must be in place for managing translations between numbering systems. Decimal here is needed for plotting, hexadecimal for coloring. Similarly to plot a color scale, a looping structure is used to cycle through the hues. In this case, decimal is required for looping and hexadecimal is again used for coloring.

Motivation for Color System Conversions

Occasionally a code needs to be expressed in a different color system. For example, HLS codes cannot be written out directly to an HTML file, because the Web only understands RGB. Therefore a translation must occur beforehand. A second example involves the digitized color chart in Figure 3. HLS codes are used for assigning color, and RGB equivalents are used in the labels.

Description of the Macros

The V8 core conversion macros are actually macro functions. Art Carpenter lists the following characteristics of macro functions in his SUGI 27 paper entitled *Macro Functions: How to Make Them - How to Use Them*:

- All statements in the macro must be macro statements.
- The macro should create NO macro variables other than those that are local to that macro.
- The macro should resolve to the value that is to be returned.

While a listing of all the core conversion macros can be found in the Appendix, let's look at RGBHEX to make sure it meets the criteria defined by Art Carpenter for macro functions:

```
%macro RGBHex(rr,gg,bb);
  %let rr=%sysfunc(round(&rr));
  %let gg=%sysfunc(round(&gg));
  %let bb=%sysfunc(round(&bb));
  %sysfunc(compress(CX%sysfunc(putn(&rr,hex2.))
                    %sysfunc(putn(&gg,hex2.))
                    %sysfunc(putn(&bb,hex2.))))
%mend RGBHex;
```

RGBHEX qualifies as a macro function. All statements are macro statements. The only macro variables in the function are parameters, and macro parameters are always local to the macro that defines them. And finally the macro resolves to a hexadecimal color code.

The input parameters to RGBHEX are three decimal digits ranging in value from 0 to 255. The output is a hexadecimal color string configured as "CXrrggbb" where each color component is two bytes in length. Heavy use is made of SYSFUNC in this macro. In the first three assignment statements SYSFUNC is used for rounding fractional values. Rounding is preferable to truncation, because color changes are not linear. This means that slight adjustments won't make any difference most of the time, but occasionally color will change drastically when the value of a component is adjusted slightly. This observation can be confirmed by looking again at the hue scale in Figure 3. Color shifts occur suddenly and at unequal intervals. Denis White and Jean Sifneos describe a weighting function that counteracts the non-linearity in their paper *Regression Tree Cartography*.

After rounding occurs, SYSFUNC is used again to access the COMPRESS function. COMPRESS rather than %CMPRES is needed for squeezing out internal blanks. Otherwise CX•rr•gg•bb would be returned instead of

CXrrggbb. Finally the third group of SYSFUNC invocations result in the actual conversion from decimal to hexadecimal.

Table 2 below contains a description of the core conversion macros listed in the Appendix. The macros are used most efficiently if they are placed in a macro library that is accessed with a SASAUTOS option:

```
options sasautos('c:\MySASAutos', sasautos);
```

For a complete description about how AUTOCALL libraries work, see Art Carpenter's SUGI 27 paper *Building and Using Macro Libraries*. You may also find it convenient to insert customized options statements into an Enhanced Editor abbreviation for easy recall.

Table 2: The Core Set of V8 Color Conversion Macros

#	Name	Input | Invocation	Output
1	RGBHEX	Three decimal RGB code components	One hex RGB code
		%RGBHEX(0,117,187)	CX0075BB
2	RGBDEC	One hex RGB code	One decimal RGB code that can be parsed by the SUBSTR function
		%RGBDEC(CX0075BB)	D000117187
3	HLSHEX	Three decimal HLS code components	One hex HLS code
		%HLSHEX(322,93,255)	H1425DFF
4	HLSDEC	One hex HLS code	One decimal HLS code that can be parsed by the SUBSTR function
		%HLSDEC(H1425DFF)	D322093255
5	RGBTOHUE	Three decimal RGB code components	One decimal HLS hue component
		%RGBTOHUE(0,117,187)	322
6	RGBTOLUM	Three decimal RGB code components	One decimal HLS LUM (lightness) component
		%RGBTOHUE(0,117,187)	93
7	RGBTOSAT	Three decimal RGB code components	One decimal HLS saturation component
		%RGBTOHUE(0,117,187)	255
8	RGBTOHLS	Three decimal RGB code components	Three decimal HLS code components
		%RGBTOHLS(0,117,187)	322,93,255
9	HLSTORGB	Three decimal HLS code components	Three decimal RGB code components
		%HLSTORGB(322,93,255)	0,117,187

Macro functions are very versatile. If you need hexadecimal output from HLSTORGB in Table 2, just *nest* the function calls as shown in the following program fragment for the digitized color chart in Figure 3:

```
%macro getData;
 %let xh=322;
 %let xs=255;
  data digitize;
    length rgbhex $8;
    %do i=20 %to 255 %by 5;
      cnum+1;
      i=input("&i",3.);
      rgbhex="%RGBHex(%HLStoRGB(&xh,&i,&xs))"; ❶
      output;
    %end;
    stop;
  run;
%mend getData;
```

❶ HLSTORGB in the nested macro function call returns a decimal triplet that is then converted to a hexadecimal RGB code by RGBHEX. The color chart is labeled with the hexadecimal RGB code values.

Macro function calls can also appear in format labels. For example, the following format increases program readability for the PATTERN colors in Table 1:

```
proc format;
 value ColFmt 1="%rgbhex(255,0,0)"     2="%rgbhex(255,102,102)"
              3="%rgbhex(255,204,204)" 4="%rgbhex(255,255,255)"
              5="%rgbhex(255,255,204)" 6="%rgbhex(255,255,68)"
              7="%rgbhex(221,255,221)" 8="%rgbhex(0,255,0)";
run;
```

And finally, macro functions can be built incrementally. The macro, RGBTOHLS, just invokes RGBTOHUE, RGBTOLUM, and RGBTOSAT in the proper order:

```
%macro RGBtoHLS(r,g,b);
  %RGBtoHue(&r,&g,&b),%RGBtoLum(&r,&g,&b),%RGBtoSAT(&r,&g,&b)
%mend RGBtoHLS;
```

Note that the commas between the internal macro calls are also returned when RGBtoHLS is invoked. They are an essential part of the three decimal components in the returned HLS code.

Scope Issues

Working with related programs that use global macro variables in a single interactive SAS session can produce unwanted side effects. Color programs are particularly vulnerable because of the layers of macro source code and because of the limited number of names that can be used to reference red, green and blue components in a color code.

Max Johnson in a posting dated October 27, 2002 on SAS-L uses PROC SQL plus the SAS supplied SYMDEL macro available in Version 8.2 to delete global macro variables in an interactive session:

```
proc sql noprint;
          select distinct NAME
          into :gmacs
          separated by ' '
          from dictionary.macros
          where upcase(SCOPE) eq 'GLOBAL' and NAME NE 'GMACS';
quit;

%symdel &gmacs gmacs;
```

The code has only been modified by the addition of a clause that excludes GMACS from the list of macro variables stored in the global macro variable of the same name. This way the recursion warning disappears when SYMDEL is invoked. GMACS is deleted after the macro variables it stores are deleted.

In addition, the SAS program that contains this code is saved to a directory named *SASInclude*. This way it can be *%included* whenever necessary into a SAS session:

```
filename delGlob "c:\SASInclude\DeleteGlobals.sas";
%include delGlob;
```

V9 Color Utility Macros Replace Predefined Color Lists

The list of 280 predefined SAS colors has been replaced by a set of color utility macros described in Version 9 *OnLineDoc* available at http://support.sas.com/publishing/cdrom/index.html. Overlap between the V8 user-defined core macros and the V9 color utility macros is confined to the RGB and HLS coloring systems. In addition CMY(cyan, magenta, yellow), CMYK(cyan, magenta, yellow, black), CNS(color naming scheme base on the HLS coding convention), and HSV(hue, saturation, value[brightness]) decimal-to-hex conversion macros are available in Version 9.

A review of Table 3 below shows that there are two major differences between the V8 and V9 macros. First, the V8 macros are defined with a higher degree of refinement (decimal 0 to 255) whereas the V9 macros work with percentages. Secondly, the V8 macros always use decimal input whereas input can be decimal or hexadecimal in the V9 macros. Presumably though, V9 macro calls can still be nested like their V8 counterparts so that decimal values can be used for input:

```
V9RGBHexCode=%HLS2RGB(%HLS(322,36,100));
```

SAS V9 *OnlineDoc* also warns that round-trip conversions with HLS2RGB and RGB2HLS may produce codes that are slightly different in value and in appearance from the originals. The color CXA**B**C**D**E**F** is used as an example. With RGB2HLS it becomes H14ACDAD, but when the inverse HLS2RGB macro is applied, the output code changes to CXA**A**C**C**E**E**. The V8 macros do not have this problem, possibly because decimals are rounded

prior to transformation by HEX formats. (V9 software is not available to the author for verification). Figure 4 shows graphical output from the *OnlineDoc* round-trip with the V8 core conversion macros.

Table 3. Comparable V8 Core Conversion Macros and V9 Color Utility Macros

V8 Macros			V9 Macros		
Name	**Input**	**Output**	**Name**	**Input**	**Output**
RGBHEX	Decimal Code Value	Hexadecimal	***RGB***	%Code Value	Hexadecimal
	%RGBHEX(0,117,187)	CX0075BB		%RGB(0,46,73)	~CX0075BB
HLSHEX	Decimal	Hexadecimal	***HLS***	(Decimal,%,%)	Hexadecimal
	%HLSHEX(**322**,93,255)	H1425DFF		%HLS(**322**,36,100)	~H1425DFF
RGBTOHLS	Decimal	Decimal	***RGB2HLS***	Hexadecimal	Hexadecimal
	%RGBtoHUE(0,117,187)	322,93,255		%RGB2HLS(CX0075BB)	H1425DFF
HLSTORGB	Decimal	Decimal	***HLS2RGB***	Hexadecimal	Hexadecimal
	%HLSTORGB(322,93,255)	0,117,187		%HLS2RGB(H1425DFF)	CX0075BB

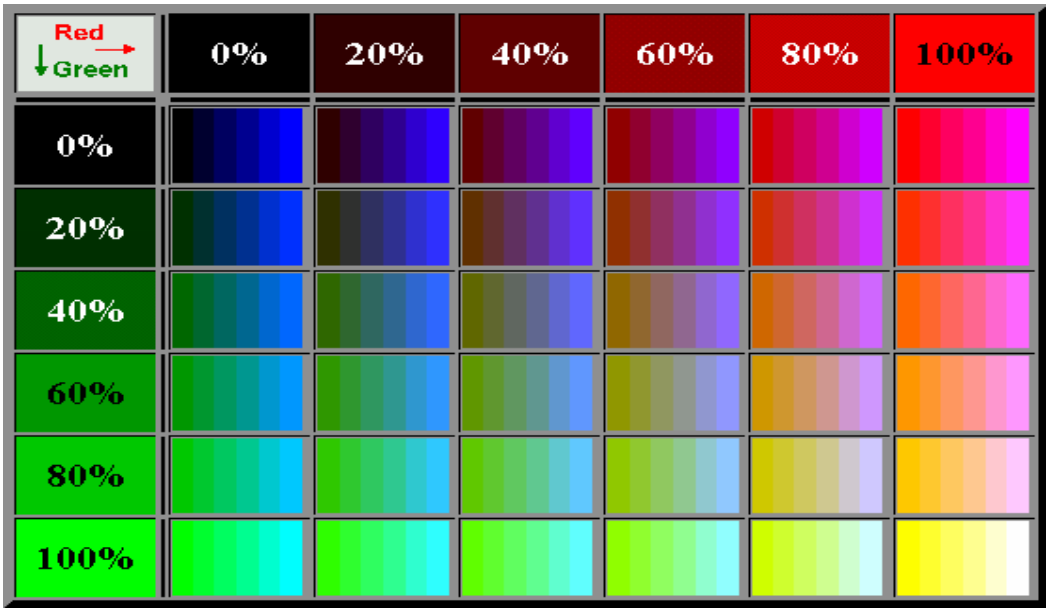
Figure 4. Round-Trip Conversions with the V8 core macros return the original code and color.

```
RGB In: CXABCDEF
RGBDec: 171,205,239
```

```
HLSHex: H14ACDAD
HLSDec: 330,205,173
```

```
HLStoRGB(330,205,173)
= CXABCDEF
```

Figure 5. The Browser-Safe "Picker" palette as a color chart reflects the percentage-based V9 Utility Macro structure.



Even though the predefined list of SAS colors has been abandoned in favor of the color utility macros, a chart is still needed as an aid for selecting colors in V9 SAS. The chart displayed in Figure 5 and described further in the

SUGI-29 poster *New Palettes for SAS® Version 9 Color Utility Macros* shows bands of colors uniformly spaced apart in RGB space. *Red* and *Green* components are fixed in the margins and *Blue* varies in the cell GIF files. With 20% as the fixed interval, the Browser-safe palette of 216 or 6^3 uniformly distributed colors in RGB space is replicated in this example.

Optimizing Code for Generating Color Charts

Palettes known as color charts are subsets of a given color space. They are derived from PROC REPORT with ODS output to HTML. They can assume multiple configurations as shown in Figure 6. The chart to the left is a fully annotated version of the unique colors from the predefined SAS list. The one immediately to the right displays the same colors in a more compact matrix format. RGB codes from the matrix can be copied and pasted directly into a SAS program. When all labels are removed, the amount of space saved is considerable. The small matrix in the lower right-hand corner paints an RGB cube face with 18^2 or 324 distinct colors. In this instance, ODS outperforms SAS/GRAPH software which is limited to 255 colors per display.

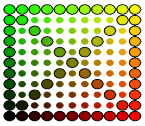
Figure 6. Color charts as lists and matrices. Source code for the *color list* and cube *face matrix* is reviewed below.

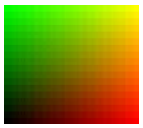
Color List

C	SAS Color Name	RGB	HLS
	CYAN	CX00FFFF	H12C80FF
	STEEL	CX3883A8	H1407080
	CHARCOAL	CX4F4F4F	H0784F00
	MAROON	CX700000	H07838FF
	LIME	CXC0FF81	H0D2C0FF
	LILAC	CXE06090	H062A0AC
	TAN	CXE0A860	H09AA0AC
	CREAM	CXE8D898	H0A8C0A2
	SALMON	CXFF0055	H06480FF
	MAGENTA	CXFF00FF	H03C80FF
	ROSE	CXFF6060	H078B0FF
	GOLD	CXFFAA00	H0A080FF

CX00FFFF	CX3883A8	CX4F4F4F	CX700000
CXC0FF81	CXE06090	CXE0A860	CXE8D898
CXFF0055	CXFF00FF	CXFF6060	CXFFAA00

SAS/GRAPH ODS





Face Matrix

Original SAS Code for the Color List

The color list in Figure 6 is a subset of the predefined SAS colors in V8 SAS *OnlineDoc*. Missing from the display in *OnlineDoc*, however, is a sample of the colors themselves, and the list itself has been completely removed from the V9 documentation. To obtain the V8 listing, therefore, follow the steps listed below:

1) Access the table of contents in SAS *OnlineDoc* and drill down to:
   SAS/GRAPH
   SAS/GRAPH COLORS
   Color-naming schemes
2) Select *All* and then *Copy* and *Paste* to insert the material into a Microsoft Word file.
3) Delete extraneous text and annotations keeping only the table.
4) Select the table, and press *Convert Table to Text* specifying comma delimited output.
5) Save the output to a TXT file that is used as input to a SAS program.
6) Create a SAS data set of the UNIQUE colors with variables CNUM (1-12), CNAME(cyan…gold), RGB, and HLS.

The SAS program that processes the UNIQUE SAS data set uses a macro together with traffic lighting to add color blocks to output from PROC REPORT. The macro that is invoked from a compute block in PROC REPORT assigns the correct background color to Column #1 with a SELECT statement:

```
%macro selectRGB;
  select (_C1_);
  %do i = 1 %to &NColors;
     when (&i.) CALL DEFINE(_COL_,"STYLE","STYLE=[ BACKGROUND=&&RGB&i. ❶]");
  %end;
  otherwise;
  end;
%mend selectRGB;
```

❶ Macro variables `&&RGB&i` for the RGB codes are created in a Data _NULL_ step. ***&i*** corresponds to CNUM in the data. Note the macro variable references a constant not a variable.

Macro variables must be used for assigning the background colors, because the syntax for the STYLE option in the ODS DEFINE statement calls for unquoted constant text, not a SAS variable. For example, ***BACKGROUND = black*** is correct; ***BACKGROUND ='black'*** is not. Here is how the macro is invoked in PROC REPORT:

```
ods listing close;
ods html
  body='c:\SUGI27\SASColorChrt.html';
  proc report data=Unique nowindows;
    columns Cnum ❶ cName RGB HLS;
    define  cNum   / display width=2 format=missf. ❷ 'C';
    define cName   / display width=10  'SAS*Color Name';
    define RGB     / display width=8 'RGB';
    define HLS     / display width=8 'HLS';
    compute CNum;  %selectRGB; ❸  endcomp;
  run;
ods html close;
ods listing;
```

❶ The variable CNUM is column #1, the argument of the SELECT statement in the SELECTRGB macro.
❷ CNUM is set to missing with the MISSF format so that only the background color is displayed. The MISSF format is defined with a single range:

```
value missf low-high=' ';
```

❸ The SELECTRGB macro is invoked from the COMPUTE block.

While the SELECTRGB macro works satisfactorily with a small number of colors, it is still too complicated, and processing slows noticeably with many colors because of the sequential access method implicit in the embedded SELECT statement.

Optimized Source Code for the RGB Cube Face Matrix

In Version 8.2 a format attached to the ODS STYLE statement obviates the need for both the data _NULL_ macro variables and the SELECTRGB macro. Nevertheless the simplicity gained in the macro workaround is offset by data processing required for generating a color matrix. Presented below are program fragments showing how RGB codes for the RGB face in Figure 6 are generated with an application of RGBHEX, and how arrays are processed in both row|column and column|row order for color assignments to the matrix.

```
%macro getFaceB0;
  data face(keep=cnum rgb);
    retain Blue 0; /*Blue = 0 face*/
    length rgb $8;
    %do green= 255 %to 0 %by -15;
      %do red = 0 %to 255 %by 15;
         rgb="%rgbhex(&red,&green.,0)";❶
         cnum+1;
         output;
      %end;
    %end;
  run;
%mend getFaceB0;
%getFaceB0
```

❶ RGBHEX converts three loop-generated decimals to an RGB code. 324 colors are produced in the data step.

While data generation is pretty straightforward, data display requires experience with arrays. Colors need to be displayed in row|column order so that they can be read left to right, top to bottom. But column values must also be identified so that variables Col1 … Col*n* can be processed in PROC REPORT. Figure 7 contains the source code with sample output that accomplishes both tasks:

Figure 7. Source code and sample output for the matrix used as input to PROC REPORT for a Color Chart.

```
data matrix(keep=cnum1-cnum&maxcol);
    array cnum[&maxcol] cnum1-cnum&maxcol;
    array x[&maxrow,&maxcol];
    do i=1 to &numcolors;❶
      col=mod(i,&maxcol);
      if col eq 0 then col=&maxcol;
      row=floor(i/&maxcol)+1;
      if col eq &maxcol then row=row-1;
      x[row,col]=i;
    end;
    do row=1 to &maxrow;❷
      do col=1 to &maxcol;
        cnum[col]=x[row,col];
      end;
      output;
    end;
  run;
```

Sample Output

MAXCOL=4, MAXROW=3, NUMCOLS=12

Obs	cnum1	cnum2	cnum3	cnum4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12

The MATRIX data set actually contains 18 variables (Cnum1…Cnum18) and 18 observations. With a full-sized matrix, Obs #2 for CNUM1 would be set to 19.

❶ Data are being positioned here in Row|Column order. Note the use of the MOD and FLOOR functions for calculating the values for ROW and COL.

❷ Column *variables* saved in the data step are created in this loop. Note the placement of the OUTPUT statement within the loop.

Finally the matrix data set is written out to an HTML file from PROC REPORT. FACEFMT for coloring comes from a control-in data set derived from the input data. Because cells containing space characters are only 1 pixel wide in HTML tables, a second format, BLANKF, increases table size by assigning a visible dot (.) to all the CNUM variables in the MATRIX data set. PROC REPORT is also contained within a macro so that the program can generate variable-sized charts.

```
ods listing close;
ods html body='c:\N03Col\AdvTut\htm\FaceNoMac.html'
style=noborder;
  /*Get a Matrix Report*/
%macro Mreport;
  proc report data=matrix nowindows headline headskip ls=80 ps=75;
   columns CNUM1-CNUM&maxcol;
   %do i=1 %to &maxcol;
     define cnum&i / display width=1 format=BlankF.
                     style=[FOREGROUND=FaceFmt. background=FaceFmt.]❶  " ";
   %end;
  run;
%mend Mreport;
%Mreport;
 ods html close;
 ods listing;
```

❶ The dots (.) are hidden by assigning the same color to FOREGROUND and BACKGROUND.

An alternative to PROC OUTPUT for handling Constant Literals

Two years ago a suggestion was made on SAS-L to create a new procedure that could handle output data set assignments at run-time. Again there was a problem with how to process in the absence of variable assignment capabilities. The argument to an OUTPUT statement is constant text, just as it is for the COLOR parameter in a STYLE statement. In both situations, binding occurs early at compile-time. Interestingly, the initial solution proposed for the OUTPUT data set problem on SAS-L was identical to the one for colors: insert a SELECT statement into a macro that expands to accommodate all possible outcomes. Unfortunately SELECT does not work when millions of records from a large transaction file are being routed to hundreds of output files. Again it became necessary to "think outside of the box" as Lauren Haworth did when she suggested using a format for colors in PROC REPORT.

David Ward supplied the work-around for the run-time output destination. He abandoned the slow, sequential access SELECT statement in favor of INDEXING with binary search capabilities:

```
proc datasets lib=work nolist;
  modify VeryLargeDS;
  index create group;
quit;

%macro SplitMac2;
  %do i=1 %to 400;
    %let ds=DS%sysfunc(putn(&i,z3.));
      data &ds;
        set VeryLargeDS (where=(group="%sysfunc(putn(&i,z3.))"));
      run;
  %end;
%mend;
%SplitMac2;
```

Ward also timed the SELECT and INDEX solutions on a one-million observation data set where GROUP assignments were made with the RANUNI( ) function. It took 60.61 seconds for the SELECT statement processing to complete whereas INDEXING required 10.99 seconds plus an additional 27.67 seconds for splitting *VeryLargeDS* into 400 subset data sets. Ward's SAS-L posting is listed in the reference section. See also Watts' NESUG '01 paper *On the Relationship between Format Structure and Efficiency in SAS* that compares resolution times for binary, sequential, and embedded formats against highly skewed distributions of start values. The binary (default) format outperformed all others regardless of the configuration of the skewed distribution, and a recommendation was made in the paper to disable the NOTSORTED option in the format procedure if the results hold up over time.

Simplifying Macro Processing for Color Scales

Figure 8. SUGI Logo with a red saturation and lightness scales. In the saturation scale, the amount of gray is varied whereas white varies in the lightness scale.

In any color scale, two color components are kept fixed while the third is systematically varied in a looping structure with output being written to an ANNOTATE data set. In the original version of the program for the saturation scale in Figure 8, a macro was required:

```
%macro scale (Hue=, lum=, SLow=, SHigh=, R=N);
  data anno;
  %system(3,3,3);
  /*GET STARTING VALUE FOR SATURATION*/
  %if &R eq %upcase(N) %then %do; /*UNSATURATED TO SATURATED*/
     %let S=&SLow;
  %end;
  %else %do;                      /*REVERSE TO GO FROM SATURATED TO UNSATURATED*/
     %let S=&SHigh;
  %end;
  %let Sincr= %sysevalf((&SHigh.-&SLow.)/100);
  %do x=0 %to 99;
    xleft=&x; xright=xleft+1; ylow=0; yhigh=100;
    %bar(xleft,ylow,xright,yhigh,%hlshex(&Hue,&Lum,&S),0,solid);
    output;
    %if &R eq %upcase(Y) %then %let S = %sysevalf(&S - &Sincr);
    %else %let S = %sysevalf(&S + &Sincr);
```

```
  %end;
  run;
%mend scale;
```

The SAS-supplied ANNOTATE BAR macro contained within macro loop above is designed to save the SAS/GRAPH programmer key strokes. Two function calls, MOVE and BAR, are automatically invoked from inside the macro. However, since the color parameter in BAR is a constant literal rather than a variable, invocation must occur inside another macro. Again, programming becomes more complicated with the layers of macros. Note the amount of blue macro source code in the program fragment above.

Simplifying the source code takes advantage of two facts: 1) the color parameter in the underlying BAR *function* can be a variable or quoted text, and 2) SAS-supplied macros can be re-written. For clarity, rename the user-defined version of the macro as _BAR, and for accuracy do an MPRINT on the BAR macro to make sure that the revision captures the full functionality of the original. Below is the code for user-defined _BAR macro stored in an AUTOCALL library:

```
%macro _bar(x1,y1,x2,y2,color);
   function='move'; x=&x1; y=&y1; output;
   function='bar'; x=&x2; y=&y2; line=0; style='s'; ❶
 %if %index(&color,*) eq 0 %then %str(color=&color;)❷;
   output;
%mend _bar;
```

❶ Constants are defined for LINE and STYLE. Bar offset is removed with line=0 and style=s for 'solid' is best for colors.

❷ An asterisk(*) in the original BAR function means that the most recently defined color is to be used again. That convention is carried over here. Note by the structure of the assignment statement that COLOR can be a variable.

The following color assignments now work as intended with the revised _BAR macro:

```
%_bar(xLeft,yLow,xRight,yHigh,myColorVbl);
%_bar(xLeft,yLow,xRight,yHigh,*);
%_bar(xLeft,yLow,xRight,yHigh,"%HLSHEX(345,84,140)");
```

Immediately below is the revised data step for the saturation scale shown in Figure 8. Original macro parameters are recast as global macro variables, and there is no role for the core conversion macros. They have been replaced by repeated invocations of the HEX format. The result is compact source code that doesn't rely heavily upon macros.

```
data anno;
 %system(3,3,3);
 length R $1 color $8;
 retain Hue &Hue  Lum &Lum  SLow &SLow  SHigh &SHigh  R "&R";
 diff=SHigh-Slow+1;
 xincr=1/diff*100;
 x=0;
 if upcase(R) eq 'N' then do;
   strt=sLow; eend=sHigh; byval=1; /*UNSATURATED TO SATURATED*/
 end;
 else do; /*SATURATED TO UNSATURATED*/
   strt=sHigh; eend=sLow; byval=-1;
 end;
 do s = strt to eend by byval;
   xleft=x; xright=xleft + xincr; ylow=0; yhigh=100;
   substr(color,1,1)='H';
   substr(color,2,3)=put(hue,hex3.);
   substr(color,5,2)=put(lum,hex2.);
   substr(color,7,2)=put(s,hex2.);
   %_bar(xleft,ylow,xright,yhigh,color);
   x= xright;
  end;
run;
```

Summary and Conclusions

Programming issues associated with color are identified in this paper. The hexadecimal numbering system is reviewed along with HEX formats that play a key role in the V8 core conversion macros. The importance of the conversion macros for managing color in SAS software is also emphasized, and detailed instructions are provided for their construction and usage. Throughout the paper, recommendations are made for simplifying source code and making it more efficient. In one instance simplicity and efficiency are simultaneously promoted by developing alternatives to a macro language solution, and in another, a SAS-supplied macro is altered so that colors can be directly assigned to variables in a data step.

References

Carey, Patrick. *Creating Web Pages with HTML: 2nd Edition*. Cambridge, MA: Course Technology, 2000.

Carpenter, Arthur. *Building and Using Macro Libraries*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #17.

Carpenter, Arthur. *Macro Functions: How to Make Them - How to Use Them*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #100.

Johnson, Max. *Deleting All Global Macro Variables*. SAS-L@LISTSERV.UGA.EDU. October 27, 2002.

SAS Institute Inc. SAS/GRAPH® Software: Reference, Version 8, Volume 1. Cary, NC: SAS Institute Inc., 1999. [Topic: BAR ANNOTATE function and macro].

SAS Institute Inc. SAS® Language Reference: Dictionary, Version 8, Volume 2. Cary, NC: SAS Institute Inc., 1999. [Topics: HEX and $HEX informats and formats plus the DELIMITER= option].

SAS Institute Inc. *SAS OnLineDoc V9*. Cary, NC: SAS Institute Inc., 2002. [Path: SAS/GRAPH 9 Reference > SAS/GRAPH Colors and Images > Specifying Colors in SAS/GRAPH Programs].

SAS Institute Inc. *SAS® Macro Language: Reference, Version 8*. Cary, NC: SAS Institute Inc., 1999. [Topic: Scope of macro variables].

Tilanus, Erik W. *%Mondriaan: Presenting 3D information in 2D*. Proceedings of the Twenty-Eighth SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2003, paper #232.

Ward, David. *RE: Efficient splitting of a data set WAS: Proc Output: A Proposal for a new procedure*. SAS-L@LISTSERV.UGA.EDU. February 12, 2002.

Watts, Perry. *Multiple-Plot Displays: Simplified with Macros*. Cary, NC: SAS Institute Inc., 2002.

Watts, Perry. *New Palettes for SAS 9 Color Utility Macros*. Proceedings of the Twenty-Ninth SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2004, paper #162.

Watts, Perry. *The Relationship Between Format Structure and Efficiency in SAS*. Proceedings of the 14th Annual Northeast SAS Users Group Conference. Baltimore, MD, pp. 697-705, 2001.

Watts, Perry. *Using ODS and the Macro Facility to Construct Color Charts and Scales for SAS® Software Applications*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #125.

Watts, Perry. *Working with RGB and HLS Color Coding Systems in SAS® Software*. Proceedings of the Twenty-Eighth SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2003, paper #136.

White, Denis and Jean C. Sifneos. *Regression Tree Cartography*. Journal of Computational and Graphical Statistics. 11(3):600-614, September 2002.

Appendix: Core Set of the V8 Color Conversion Macros

The programs below are named so that they can be stored in an AUTOCALL library. They replace the programs listed in the NESUG 15 paper: *Working with RGB and HLS Color Coding Systems in SAS® Software*.

```
/*  ------------------------------------------------------------------
    Program  :  HLSDec.sas
    Purpose  :  Convert a SAS HEX HLS color to its decimal equivalent.
                (For more readable codes).
    Input    :  A SAS HLS color Code Hhhhllss where hhhllss represent
                three hexadecimal numbers hhh, ll, and ss.
    Output   :  A String representing three decimal numbers
                Dhhhlllsss.
    ------------------------------------------------------------------ */
%macro HLSDec(Hhhhllss);
  %local hue lite sat;
  %let hue=%substr(&Hhhhllss,2,3);
  %let lite=%substr(&Hhhhllss,5,2);
  %let sat=%substr(&Hhhhllss,7);
  %sysfunc(compress(D%sysfunc(putn(%sysfunc(inputn(&hue,hex3.)),z3.))
                    %sysfunc(putn(%sysfunc(inputn(&lite,hex2.)),z3.))
                    %sysfunc(putn(%sysfunc(inputn(&sat,hex2.)),z3.))))
%mend HLSDec;

/*  ------------------------------------------------------------------
    Program  :  HLSHex.sas
    Purpose  :  Convert SAS Decimal HLS color to HEX which SAS understands.
                Round first, because HEX fcns truncate.
    Input    :  Three decimal numbers for HLS color components.
    Output   :  a SAS HLS color Code Hhhhllss where hhh,ll,ss are
                three hexadecimal numbers.
    ------------------------------------------------------------------ */
%macro HLSHex(hhh,ll,ss);
  %let hhh=%sysfunc(round(&hhh));
  %let ll=%sysfunc(round(&ll));
  %let ss=%sysfunc(round(&ss));

  %sysfunc(compress(H%sysfunc(putn(&hhh,hex3.))
                    %sysfunc(putn(&ll,hex2.))
                    %sysfunc(putn(&ss,hex2.))))
 %mend HLSHex;

/*  ------------------------------------------------------------------
    Program  :  HLStoRGB.sas
    Purpose  :  Convert an HLS decimal code to its RGB counterpart.
    Algorithm:  Foley, J.D. and A. Van Dam. Fundamentals of Interactive
                Computer Graphics". Reading, MA: Addison-Wesley Publishing
                Company, 1983, p. 619.
    Input    :  Three decimals for hue, light, and saturation.
    Output   :  A character string of three RGB decimals separated by
                commas: rrr,ggg,bbb.
    ------------------------------------------------------------------ */
%macro HLStoRGB(h,l,s);
  %local hue light sat m1 m2 rhue ghue bhue r g b rr gg bb;
  %let hue=%sysevalf(&h - 120);
  %let light =%sysevalf(&l/255);
  %let sat=%sysevalf(&s/255);
  %if &light le 0.5 %then %let m2=%sysevalf(&light*(1+&sat.));
  %else %let m2=%sysevalf(&light+&sat.-&light*&sat);
  %let m1 = %sysevalf(2 * &light. - &m2.);
  %if &sat eq 0 %then %do;
    %let r=&l; %let g=&l; %let b=&l;
  %end;
  %else %do;
    %let rhue=%eval(&hue+120);
    %if &rhue gt 360 %then %let rhue= %eval(&rhue.-360);
```

```
    %if &rhue lt 0 %then %let rhue= %eval(&rhue.+360);
    %if &rhue lt 60 %then %let r = %sysevalf((&m1+(&m2-&m1)*&rhue./60)*255);
    %else %if &rhue lt 180 %then %let r=%sysevalf(255*&m2.);
    %else %if &rhue lt 240 %then %let r=%sysevalf((&m1+(&m2-&m1)*(240-&rhue)/60)*255);
    %else %let r=%sysevalf(255*&m1);

    %let ghue=&hue;
    %if &ghue gt 360 %then %let ghue= %eval(&ghue.-360);
    %if &ghue lt 0 %then %let ghue= %eval(&ghue.+360);
    %if &ghue lt 60 %then %let g = %sysevalf((&m1+(&m2-&m1)*&ghue./60)*255);
    %else %if &ghue lt 180 %then %let g=%sysevalf(255*&m2.);
    %else %if &ghue lt 240 %then %let g=%sysevalf((&m1+(&m2-&m1)*(240-&ghue)/60)*255);
    %else %let g=%sysevalf(255*&m1);

    %let bhue=%eval(&hue-120);
    %if &bhue gt 360 %then %let bhue= %eval(&bhue.-360);
    %if &bhue lt 0 %then %let bhue= %eval(&bhue.+360);
    %if &bhue lt 60 %then %let b = %sysevalf((&m1+(&m2-&m1)*&bhue./60)*255);
    %else %if &bhue lt 180 %then %let b=%sysevalf(255*&m2.);
    %else %if &bhue lt 240 %then %let b=%sysevalf((&m1+(&m2-&m1)*(240-&bhue)/60)*255);
    %else %let b=%sysevalf(255*&m1);
  %end;
  %let rr=%sysfunc(round(&r));
  %let gg=%sysfunc(round(&g));
  %let bb=%sysfunc(round(&b));
  &rr.,&gg.,&bb.
%mend HLStoRGB;

/*  ------------------------------------------------------------------------
    Program  :  RGBDec.sas
    Purpose  :  Convert an RGB Hex color to three decimal numbers
                for readability and for looping (inverse of RGBHEX.sas).
    Input    :  An RGB color Code CXrrggbb where rrggbb are three
                hexadecimal numbers.
    Output   :  A String representing three decimal numbers
                Drrrgggbbb.
    ------------------------------------------------------------------------ */
%macro RGBDec(CXrrggbb);
  %local rr gg bb;
  %let rr=%substr(&CXrrggbb,3,2);
  %let gg=%substr(&CXrrggbb,5,2);
  %let bb=%substr(&CXrrggbb,7);
  %sysfunc(compress(D%sysfunc(putn(%sysfunc(inputn(&rr,hex2.)),z3.))
                    %sysfunc(putn(%sysfunc(inputn(&gg,hex2.)),z3.))
                    %sysfunc(putn(%sysfunc(inputn(&bb,hex2.)),z3.))))
%mend RGBDec;

/*  ------------------------------------------------------------------------
    Program  :  RGBHex.sas
    Purpose  :  Convert Decimal RGB color to HEX which SAS understands.
                Round first, because HEX truncates.
    Input    :  Three decimal numbers representing a color in terms
                of its red, greeen, and blue components.
    Output   :  a SAS RGB color Code CXrrggbb where rrggbb are
                three hexadecimal numbers.
    ------------------------------------------------------------------------ */
%macro RGBHex(rr,gg,bb);
  %let rr=%sysfunc(round(&rr));
  %let gg=%sysfunc(round(&gg));
  %let bb=%sysfunc(round(&bb));
  %sysfunc(compress(CX%sysfunc(putn(&rr,hex2.))
                    %sysfunc(putn(&gg,hex2.))
                    %sysfunc(putn(&bb,hex2.))))
%mend RGBHex;
```

```
/*  ------------------------------------------------------------------
    Program  :  RGBtoHUE.sas
    Purpose  :  Calculate hu in HLS from an RGB code.
    Algorithm:  Foley, J.D. and A. Van Dam. Fundamentals of Interactive
                Computer Graphics". Reading, MA: Addison-Wesley Publishing
                Company, 1983, p. 618.
    Input    :  Three decimals for red, green and blue.
    Output   :  One decimal for hue.
    ------------------------------------------------------------------ */
%macro RGBtoHue(r,g,b);
 %local red green blue mmax mmin hue;
 %let red = %sysevalf(&r/255);
 %let green =%sysevalf(&g/255);
 %let blue=%sysevalf(&b/255);
 %let mmax=%sysfunc(max(&red,&green,&blue));
 %let mmin=%sysfunc(min(&red,&green,&blue));
 %if &mmax eq &mmin %then %let hue=0;
 %else %do;
   %let rc=%sysevalf((&mmax-&red)/(&mmax-&mmin));
   %let gc=%sysevalf((&mmax-&green)/(&mmax-&mmin));
   %let bc=%sysevalf((&mmax-&blue)/(&mmax-&mmin));
   %if &red eq &mmax %then %let hue=%sysevalf(&bc-&gc);
   %else %if &green eq &mmax %then %let hue=%sysevalf(2+&rc-&bc);
   %else %if &blue eq &mmax %then %let hue=%sysevalf(4+&gc-&rc);
   %let hue=%sysevalf(&hue*60);
   /*if hue lt 0*/
   %if %index(&hue,'-') gt 0 %then %let hue=%sysevalf(&hue+360);
   %let hue=%sysfunc(round(&hue+120));
   %if &hue ge 360 %then %let hue=%eval(&hue-360);
  %end;
 %sysfunc(putn(&hue,3.))
%mend RGBtoHue;
/*  ------------------------------------------------------------------------------------------
    Program  :  RGBtoLum.sas (see header comments for RGBtoHue.sas - gets a light val)
    ------------------------------------------------------------------------------------------ */
macro RGBtoLUM(r,g,b);
   %local red green blue mmax mmin ll lite;
   %let red = %sysevalf(&r/255);
   %let green =%sysevalf(&g/255);
   %let blue=%sysevalf(&b/255);
   %let mmax=%sysfunc(max(&red,&green,&blue));
   %let mmin=%sysfunc(min(&red,&green,&blue));
   %let ll=%sysevalf((&mmax+&mmin)/2);
   %let lite=%sysfunc(round(%sysevalf(&ll*255)));
   %sysfunc(putn(&lite,3.))
%mend RGBtoLUM;
/*  ------------------------------------------------------------------------------------------
    Program  :  RGBtoSat.sas (see header comments for RGBtoHue.sas - gets a saturation val)
    ------------------------------------------------------------------------------------------ */
   %macro RGBtoSat(r,g,b);
       %local red green blue mmax mmin ll sat;
       %let red = %sysevalf(&r/255);
       %let green =%sysevalf(&g/255);
       %let blue=%sysevalf(&b/255);
       %let mmax=%sysfunc(max(&red,&green,&blue));
       %let mmin=%sysfunc(min(&red,&green,&blue));
       %let ll=%sysevalf((&mmax+&mmin)/2);
       %if &mmax eq &mmin %then %let sat=0;
       %else %do;
         %if %sysfunc(putn(&ll,3.1)) le 0.5 %then
           %let sat=%sysfunc(round((%sysevalf((&mmax-&mmin))/%sysevalf((&mmax+&mmin)))*255));
         %else
           %let sat=
                  %sysfunc(round((%sysevalf((&mmax-&mmin))/%sysevalf((2-&mmax-&mmin)))*255));
        %end;
       %sysfunc(putn(&sat,3.))
    %mend RGBtoSat;
```

```
/*  ------------------------------------------------------------------------------------
    Program  :  RGBtoHLS.sas (see header comments for RGBtoHue.sas - returns an HLS code)
    ------------------------------------------------------------------------------------ */
   %macro RGBtoHLS(r,g,b);
     %RGBtoHue(&r,&g,&b),%RGBtoLum(&r,&g,&b),%RGBtoSAT(&r,&g,&b)
   %mend RGBtoHLS;
```


Acknowledgements

Thanks to Samuel Litwin, David Cassell, Michael Friendly, and Mike Zdeb for their insight and encouragement.

Trademark Citation

Contact Information

The author welcomes feedback via email at: **perryWatts@comcast.net**