SAS4330-2020

Expressions in **SAS®** Graph Template Language and Other Tips

Prashant Hebbar, SAS Institute Inc.

ABSTRACT

This paper is a reprise of the SAS® Global Forum 2013 paper entitled "Free Expressions and Other GTL Tips". SAS® Graph Template Language (GTL) provides many powerful features for creating versatile graphs. The statistical graphics capability in GTL lets you use expressions directly in the templates in order to simplify data preparation. This presentation covers some ways of using DATA step functions to create grouped plots based on conditions and to select a subset of the observations. It also illustrates how to use the SAS Function Compiler (FCMP) procedure functions in GTL expressions. Tips on using non-breaking spaces for creating "chunked" graphs and indenting text are discussed. Workarounds for rendering Unicode characters using data column variables are also discussed.

INTRODUCTION

Output Delivery System (ODS) Graphics allows you to create effective analytical graphs. The Statistical Graphics (SG) procedures (also known as the ODS Graphics procedures), and the Graph Template Language (GTL) are significant parts of ODS Graphics. This system has been a part of Base SAS since SAS® 9.3.

All graphs in this system require a STATGRAPH template composed of GTL statements generated by the TEMPLATE procedure. A procedure then requests a rendering of the template with your data. The SGRENDER procedure allows you to directly associate your data with a template to render a graph. Other procedures offer varying levels of abstraction of this two-step process. Many analytical procedures use this mechanism to generate their graphs.

GTL offers you the following building blocks that you can combine in myriad ways to meet your graphing needs:

- Layouts: Overlay, OverlayEquated, Gridded, Region, Lattice, DataLattice, and DataPanel.
- Plots: Scatter, Series, Histogram, Bubble, High-Low, BoxPlot, BarChart, Fit Plots, Axis Table, Text, Polygon, and more.
- Text and Legends: EntryTitle, EntryFootnote, Entry, DiscreteLegend, ContinuousLegend, and more.
- Annotations: text, line, arrow, rectangle, oval, polygon, and image.
- Other Features: Expressions, conditional blocks, dynamic variables, and macro variables.

This presentation shows you how expressions in GTL can be used to prepare data for graphs. It also shows creative uses of non-breaking space in graphs. Last, it describes workarounds for rendering character data containing Unicode characters. Other than expressions, the rest of these ideas can also be applied in the SG Procedures.

You can download a copy of the code and data used in this paper from the SAS Technical Papers and Presentations site at https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4330-2020-Hebbar. The code in this paper was tested using SAS 9.4M6 and SAS Viya 3.5. The graphs were generated using the Dove style at 200 Dots per Inch (DPI).

WHAT ARE GTL EXPRESSIONS

GTL supports arithmetic and logical expressions like DATA step expressions. The operands can also be dynamic variables or run-time macro variables. Note that logical expressions do not subset observations like WHERE clauses do. Instead, they return a Boolean value for each observation. In GTL, the expression must be enclosed in an EVAL function unless it is a Boolean expression being used as a predicate in an IF or IF-ELSE conditional block.

GTL expressions are evaluated by the Statistical Graphics Engine in the ODS Graphics system. GTL also supports some functions that are not supported in DATA step. For more details, you can refer to the "Runtime Programming Features" section in the SAS® Graph Template Language Reference.

EXAMPLES OF EXPRESSION USAGE

In this section we will go over some typical usage of expressions in GTL. A general rule to keep in mind is that your expression should return a column of values where GTL expects a column and return a single scalar value where a scalar is expected. If these do not match, then the resulting value(s) will not resolve.

CREATING GROUPS USING EXPRESSIONS

In many situations, you might want to group your observations based on conditions in the data instead of an existing group variable. Consider the SASHELP.CLASS data set. If you want to group this data by age into tweens and teens, you could create a new data set and add a new AGE_GROUP variable, as shown here:

```
data work.grp_class;
   attrib age_group length=$8 label='Age Group' ;
   set sashelp.class;
   if (age LT 13) then age_group = 'tween';
   else age_group = 'teen';
run;
```

This data set can then be used with a GTL template and PROC SGRENDER to display, for example, a scatter plot with GROUP=AGE_GROUP. Note that you can also do this by creating a user-defined format (by using the FORMAT procedure), with similar logic and using that format on the age column when creating the graph.

Another approach is to use GTL expressions in the template. You do not need to create a derived data set or a new user-defined format. You can use an expression for the GROUP= option to generate a new column variable instead. Here is a code snippet that demonstrates this:

```
proc template;
  define statgraph age_grp_expr;
    beginGraph;
      entryTitle "Generate Groups by Expression";
      layout overlay;
        scatterPlot x=weight y=height / name="sp1"
              group=eval(ifc(age LT 13, "tween", "teen"));
        discreteLegend "sp1" / title='Age Group' location=inside
              hAlign=right vAlign=bottom;
      endLayout;
      entryFootnote halign=left
                   'group=eval(ifc(age LT 13, "tween", "teen"))';
    endGraph;
  end;
run;

proc sgrender data=sashelp.class template=age_grp_expr;
run;
```

The graph output from this program is shown in Figure 1.

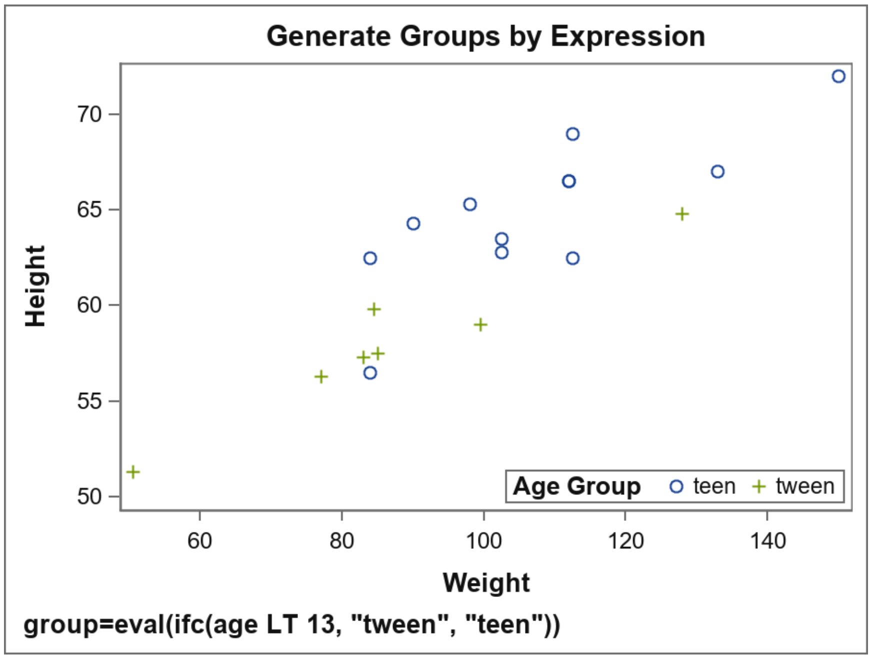

Figure 1. Generate Groups by Expressions

In this example, we used the IFC function to create a character-valued group column, based on the condition `age LT 13`. Similarly, you can also use the IFN function to generate numeric values. These functions are well-documented in the *SAS® Functions and CALL Routines Reference*.

As a further refinement, we can use a template DYNAMIC variable in the expression. This allows us to set and change the boundary age when generating the graph at run time instead of hard coding a value in the template. The following code snippet illustrate this:

```
  proc template;
    define statgraph age_grp_expr;
      dynamic boundary "Cut off age for tweens";
      beginGraph;
       < ... >
          scatterPlot x=weight y=height / name="sp1"
                markerAttrs=(size=12 weight=bold)
                group=eval(ifc(age LT boundary, "tween", "teen"));
       < ... >
      endGraph;
    end;
  run;

  /* Run the first graph with age 12 as the cut-off */
  proc sgrender data=sashelp.class template=age_grp_expr;
    dynamic boundary = 12;
  run;

  /* Create another graph rendering with a different cut-off! */
```

```
proc sgrender data=sashelp.class template=age_grp_expr;
  dynamic boundary = 14;
run;
```

SELECTING SUBSETS OF OBSERVATIONS

Although we cannot truly subset observations using expressions in GTL, we can simulate this behavior by setting values outside our range of interest to missing!

To illustrate this idea, let us say that you want to subset the automobiles in the SASHELP.CARS data set into two: those with high highway miles per gallon (MPG), and those with low highway MPG. You can then plot each MPG subset in its own cell against the Manufacturer's Suggested Retail Price (MSRP). To do this in the DATA step, you would need to create two new columns based on MPG_HIGHWAY by setting their values to missing when their respective condition is not met.

We can do something similar using the IFN function directly in GTL, as shown in the code snippet below:

```
proc template;
  define statgraph selections;
    dynamic upLimit "low end of high mpg"
            lowLimit "upper end of low mpg";
    beginGraph / designHeight=600;
      EntryTitle "Vehicles with Extreme Highway MPG (2004)";
      layout lattice / rows=2 < ... >;
        cell;
          cellHeader;
            Entry "MPG " {unicode '2265'x} " " upLimit;
          endCellHeader;
          scatterPlot x=eval(ifn(mpg_highway >= upLimit, msrp, .))
                      y=eval(ifn(mpg_highway >= upLimit, mpg_highway, .))
                      / group=make dataLabel=model <...> ;
        endCell;
        cell;
          cellHeader;
            Entry "MPG " {unicode '2264'x} " " lowLimit;
          endCellHeader;
          scatterPlot x=eval(ifn(mpg_highway <= lowLimit, msrp, .))
                      y=eval(ifn(mpg_highway <= lowLimit, mpg_highway, .))
                      / group=make dataLabel=model <...> ;
        endCell;
      endLayout;
    endGraph;
  end;
run;
```

We can now generate the graph as follows:

```
ods graphics / reset height=600px labelMax=600;
proc sgrender template=selections data=sashelp.cars;
  dynamic upLimit=44 lowLimit=16 ;
run;
```

This gives us the graph in Figure 2, with the subset of automobiles with a highway mpg ≤ 16 in the lower cell and the subset with a highway mpg ≥ 44 in the upper cell.

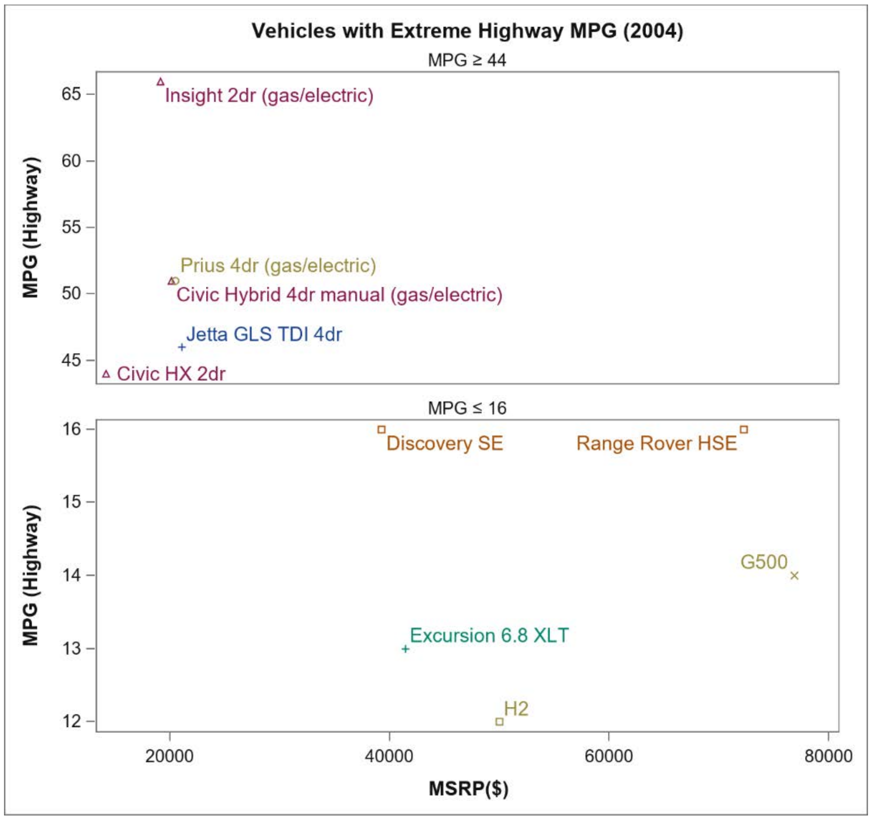

Figure 2. Selecting Observations using Expressions

Note that you might have to apply equivalent conditional expressions on all the coordinate and response roles for a given plot. This is true even though using the expression on just one coordinate role replaces the unwanted observations with MISSING and prevents those observations from being plotted.

In this example, if only X= used the subset expression and Y= retained the original column variable, then the data range for Y would remain the same as the original range. This skews your Y-axis range and leads to inefficient usage of the graph area because the system computes data ranges for a plot independently for each role. Using the same conditional expression in all the coordinate and response roles of a plot avoids this issue.

SIMPLE STATISTICS USING EXPRESSIONS

You can calculate mean, median, standard deviation, and other simple statistics with GTL expressions and use them in the graph. Figure 3 shows a scatter plot of vehicle MPG versus their MSRP from the SASHELP.CARS data set. We have plotted a reference line at the median MSRP and two reference lines at mean ± standard deviation. We have also shown the minimum and maximum weight values in a footnote.

Since reference lines and entry footnote require scalar values, you need to choose expressions that evaluate to single values and not columns.

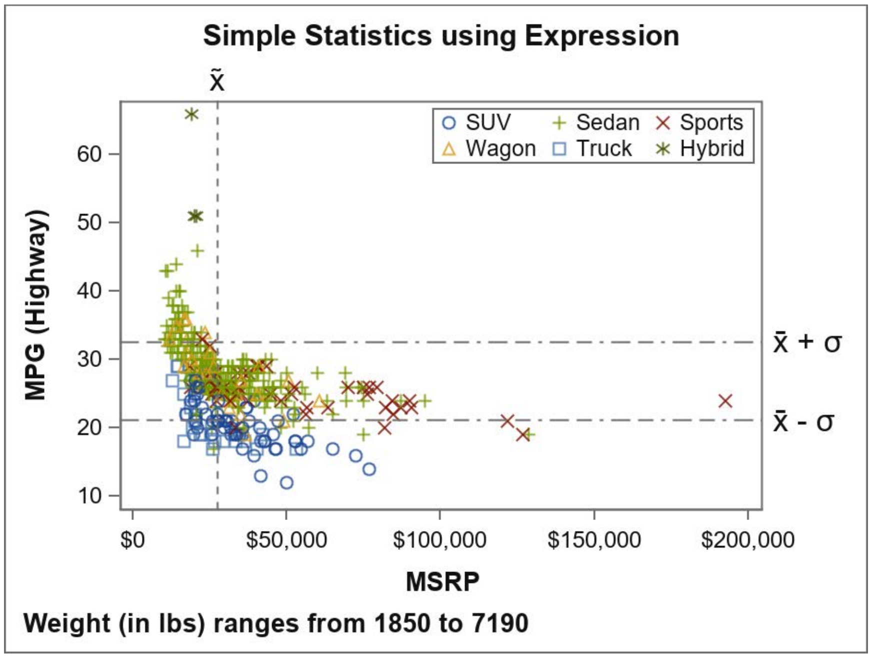

Figure 3. Median, Mean, and STD using Expressions

The template code snippet for this graph is shown below:

```
<proc template...>
    layout overlay;
      scatterPlot x=msrp y=mpg_highway / name="sp1"
            group=type dataTransparency=0.2;
      referenceLine x=eval(median(msrp)) / lineAttrs=(pattern=shortDash)
            curveLabel="x(*ESC*){unicode '0303'x}"
            curveLabelAttrs=(size=12);
      referenceLine y=eval(mean(mpg_highway) + std(mpg_highway)) /
            curveLabel="x(*ESC*){unicode '0304'x} + (*ESC*){unicode sigma}"
            curveLabelAttrs=(size=12) lineAttrs=(pattern=dashDashDot);
      referenceLine y=eval(mean(mpg_highway) - std(mpg_highway)) /
            curveLabel="x(*ESC*){unicode '0304'x} - (*ESC*){unicode sigma}"
            curveLabelAttrs=(size=12) lineAttrs=(pattern=dashDashDot);
      discreteLegend "sp1" / across=3 <...> ;
    endLayout;

    entryFootnote halign=left "Weight (in lbs) ranges from "
            eval(min(weight)) " to " eval(max(weight)) ;
<...>
```

This code also uses ODS escapes for inserting Unicode symbols in the curve labels. Also note that while mean(mpg_highway) and std(mpg_highway) have been specified twice in the template, the Statistical Graphics Engine only evaluates them once per rendering, regardless of the number of open ODS destinations.

USING FCMP FUNCTIONS

If you are an advanced user who needs their own subroutine implementation, Base SAS provides a function compiler using the FCMP procedure. These user-defined functions can be used in a DATA step, as well as many other procedures. Note that CALL routines are not supported in this context. Please see the "FCMP Procedure" section in *Base SAS® Procedures Guide* for more information on FCMP.

The following program implements FCMP functions for the traditional Body Mass Index (BMI), and an alternate proposal by Dr. Nick Trefethen at Oxford University, UK. We also coded two convenience functions for the difference as well as an absolute difference between the old and the alternate (new) equations.

```
proc fcmp outlib=sasuser.cmplib.test;

/* See http://people.maths.ox.ac.uk/trefethen/bmi.html */
  function BMI(height_inch, weight_lb);
    return (703 * weight_lb / (height_inch ** 2));
  endsub;

  function newBMI(height_inch, weight_lb);
    return (5734 * weight_lb / (height_inch ** 2.5));
  endsub;

  function BMIDiff(height_inch, weight_lb);
    return (newBMI(height_inch, weight_lb)
              - BMI(height_inch, weight_lb));
  endsub;

  function absBMIDiff(height_inch, weight_lb);
    return(abs(BMIDiff(height_inch, weight_lb)));
  endsub;

run; quit;
```

You can now use these functions directly in GTL. Let us plot the difference between the new and old BMI values for each student in the SASHELP.CLASS data set as a bubble plot. We indicate the magnitude of the difference with the size of the bubble and the increase or decrease with the fill color of the bubble, as shown below:

```
options cmplib=sasuser.cmplib; /* Location of FCMP functions */

proc template;
  define statgraph bmi;
    beginGraph;

      rangeAttrMap name="ram1";
        range min - 0 / rangeColorModel=(green white);
        range 0 - max / rangeColorModel=(white red);
      endRangeAttrMap;
      rangeAttrVar var=eval(BMIDiff(height, weight)) attrVar=bmiDiff
                                                     attrMap="ram1";
     < ... >
      layout overlay;
        bubblePlot x=weight y=height size=eval(absBMIDiff(height, weight))/
                                 colorResponse=bmiDiff  < ... > ;
        < ... >
      endLayout;
     < ... >
    endGraph;
  end;
run;

proc sgrender data=sashelp.class template=bmi;
run;
```

The graph from this program is shown in Figure 4.

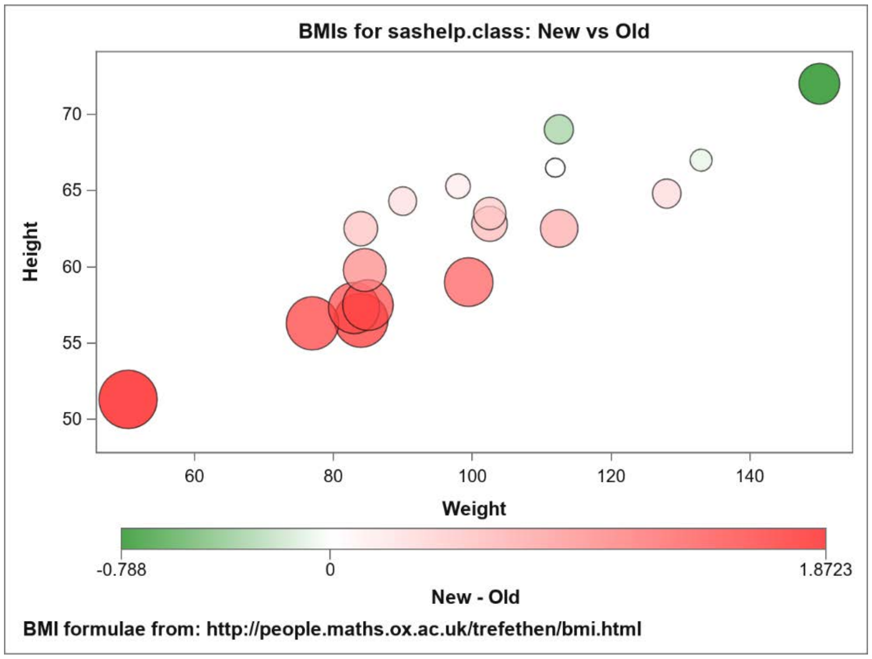

Figure 4. Using FCMP functions in GTL

SPACE GAMES

GTL internally retains leading blanks in text values, but it displays axis tick values and legend item labels after stripping any leading *and* trailing blanks. Use the non-breaking space character (‘A0’x in ASCII, ‘C2A0’x in UTF-8 and ‘41’x in EBCDIC) to work around this because these characters are not stripped, and they render like a blank character.

The 2013 version of this paper was targeted at a SAS audience that used a much earlier version. It had many examples of how to use non-breaking space to retain trailing blanks, to indent text, and to right-justify text when placed to the left of a graphical element. With SAS 9.4 there are better ways to achieve these features.

INDENTED TEXT

In certain visualizations, you might want to display a table of values in the graph. Further, you might also want to demarcate certain rows by indenting them. A forest plot is one such graph, where a table of studies contains subgroups that are indented to clarify the grouping.

In releases earlier than SAS 9.4, you would have used a scatter plot with the MARKERCHARACTER= option to precisely place the text along the Y-axis. Subgroup labels had to be indented by prefixing them with a (non-breaking) space! Starting with SAS 9.4, you can do this painlessly using the AXISTABLE statement.

Here is an example of a forest plot created using an axis table. Notice the indented table values on the left side. This was easily implemented with axis table’s INDENT= and INDENTWEIGHT= options, which let you control the amount of text indentation. The change in font weights (bold vs regular), was implemented with a DISCRETEATTRMAP. In fact, all the text columns in this graph are axis tables!

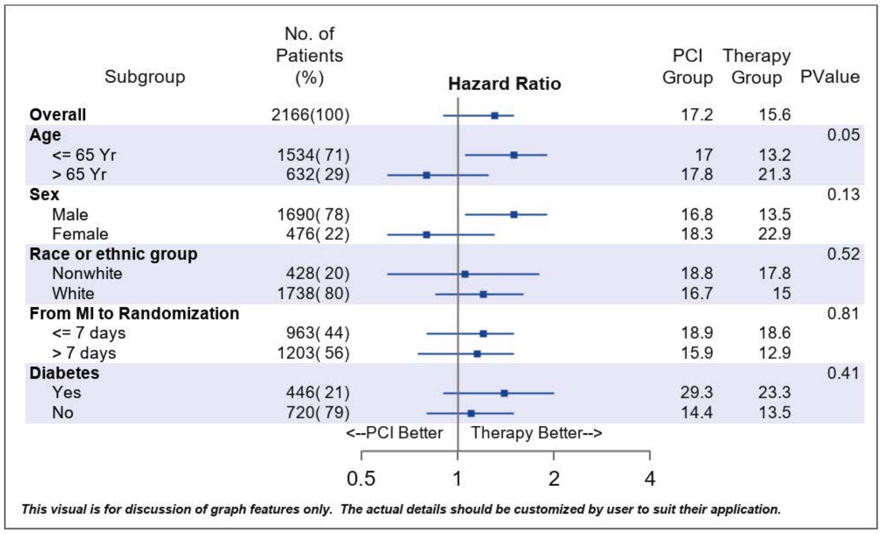

Figure 5. Indented Text using Axis Table

While this example was created using the SGPLOT procedure, you can implement this directly in GTL as well. The full code for this example is available at this paper's GitHub page.

RIGHT-JUSTIFIED TEXT

Have you run into a scenario where you need to place some text on the left side of a graphical element (that is, right-justified text)? Prior to SAS 9.3, high-low plot, scatter plot data labels with an explicit DATALABELPOSITION= option, and the DRAWTEXT annotation statement were not available. The workaround for those older releases was to create a modified data set. You appended non-breaking space characters equal to the length of the text so that the center position of the new text effectively places the original text to the left of the data coordinate! In addition, you had to use fixed width fonts for this trick to work consistently.

Adverse Event plots where you need right-justified text can be easily created using the high-low plot starting with SAS 9.3. Other plots using right-justified text can be drawn using the TEXTPLOT statement, available with SAS 9.4M2.

To illustrate how simple this feature is, let us create a scatter plot with data labels positioned on the left of the marker symbols for the SASHELP.CLASS data set. We use the NAME column for the data labels. The code follows:

```
proc template;
  define statgraph rjText;
    beginGraph;
      entryTitle "Right-justified Text: using DataLabelPosition=Left";
      layout overlay;
        scatterPlot x=weight y=height / group=sex
                dataLabel=name dataLabelPosition=left;
      endLayout;
    endGraph;
  end;
run;

/*--Draw the Graph--*/
proc sgrender data=sashelp.class(obs=10) template=rjText;
run;
```

No data preparation or tricks are required anymore! The output is shown in Figure 6.

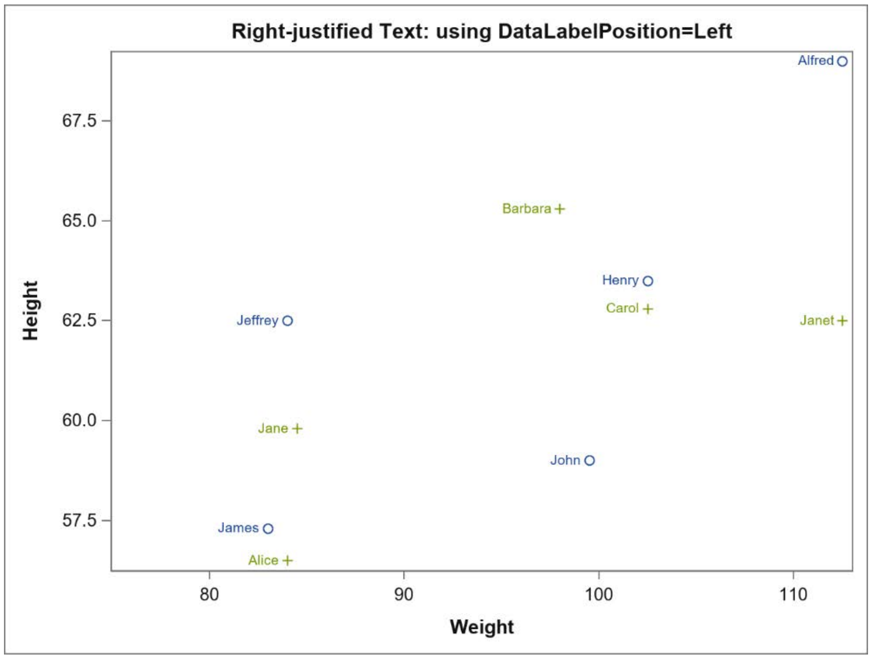

Figure 6. Right-justified Text using Non-Breaking Space

Right-justified text at the start of events in an Adverse Event plot are also easily done with the high-low plot's LOWLABEL= option, which does the justification automatically. Figure 7 shows an adverse event plot created with the high-low plot.

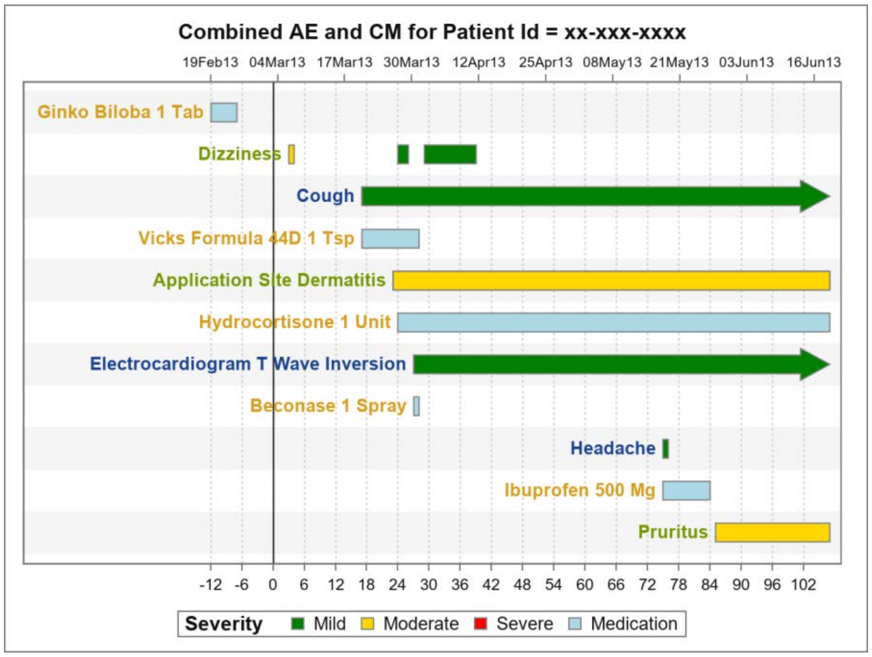

Figure 7. Adverse Event Plot using High-Low Plot.

The code uses the SGPLOT procedure and is available at this paper's GitHub page. But the graph can also be created with GTL.

DISCRETE AXIS IN "CHUNKS"

It is said that a person can hold four or five things in their working memory at once! Often, breaking a busy graph's discrete axis into chunks improves its readability. Here is one such example, where the categorical axis has been partitioned into weeks, quarters, and half-yearly observations.

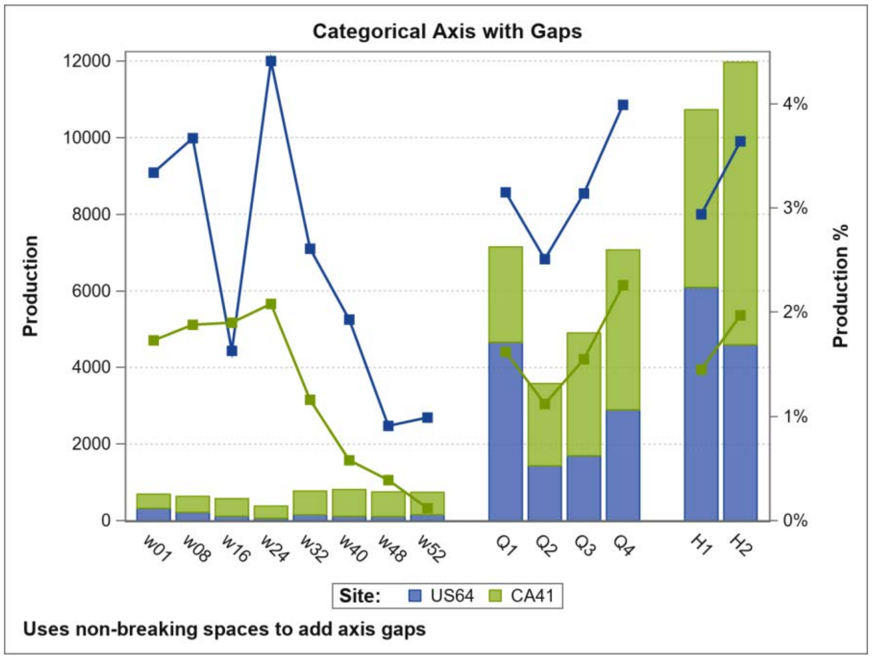

Figure 8. Breaking Graphs into Chunks with Non-Breaking Space

You can use a non-breaking space to achieve this. The basic idea is to insert non-breaking spaces into the categorical variable to correspond to the desired gaps in the axis. Because this renders as a blank and we hide the ticks, it appears as if we have introduced some gap between the ticks! Note that each break uses a different number of these characters to make them all unique and hence map to distinct coordinates in the axis. The response values for all these gap category values is set to MISSING. Here is the code snippet for the data:

```
/* Data for illustration purposes only! */
%let nbsp='A0'x; /* ASCII: 'A0'x, UTF-8: 'C2A0'x, EBCDIC: '41'x */
data plantProd;
  < ... >
  input prod prod_pct Time $ Site $;
  Time=translate(Time, &nbsp., '.');  /* map '.' to non-breaking space */
  datalines;
321   0.0334  w01 US64
373   0.0173  w01 CA41
< ... >
.     .       ..  US64
.     .       ..  CA41
4657  0.0315  Q1  US64
2491  0.0162  Q1  CA41
< ... >
.     .       ... US64
.     .       ... CA41
6091  0.0294  H1  US64
4638  0.0145  H1  CA41
< ... >
;
run;
```

Next, we create the template and render the graph. There is nothing special about this template, except for the **BREAK=true** option in the SERIESPLOT statement. This ensures that the series plot does not connect through the missing observations.

```
proc template;
  define statgraph chunked;
    beginGraph;
     < ... >
      layout overlay / < ... >;
         BarChartParm X=time Y=prod / group=Site name="bar" <...>;
         SeriesPlot X=time Y=prod_pct / group=Site display=all
                   break=true yaxis=y2  < ... > ;
     < ... >
      endLayout;
    endGraph;
  end;
run;

proc sgrender template=chunked data=plantProd;
run;
```

You can tweak this code further – the translate() function calls that map periods to non-breaking spaces can be moved into the template code as expressions for the X= !

UNICODE IN DATA COLUMNS

GTL allows you to set Unicode character values either as keywords or as hexadecimal literals in the template code for options that take string literals. It also supports inline ODS escapes for the UNICODE function in such options. Here is an example of entry title syntax for displaying 'α Values for Ψ Waves':

```
entryTitle {unicode alpha} "Values for (*ESC*){unicode '03A8'x} Waves";
```

You can also use inline ODS escapes in annotate data sets for use with SG Procedures (for the SAS 9.3 release and later). Starting with SAS 9.4, you can use annotate data sets in GTL templates as well.

There are two ways to render Unicode characters from text column variables for graphical elements that are not annotations (for example, for data labels in a scatter plot).

Note that to render Unicode characters correctly, you need to ensure that the font being used supports the characters that you are trying to display. The GRAPHUNICODETEXT style element is an easy way to specify a good Unicode font.

USING USER-DEFINED FORMATS

Starting with SAS 9.4M3, we can use PROC FORMAT to generate user-defined formats that support ODS escapes in the GTL context. These can be used to format data values with Unicode characters.

As an example, Figure 9 shows a simple bar chart of the three most frequent and the three least frequent zodiac signs (based on completely ad hoc data!). The X-axis tick values are rendered using the zodiac symbols.

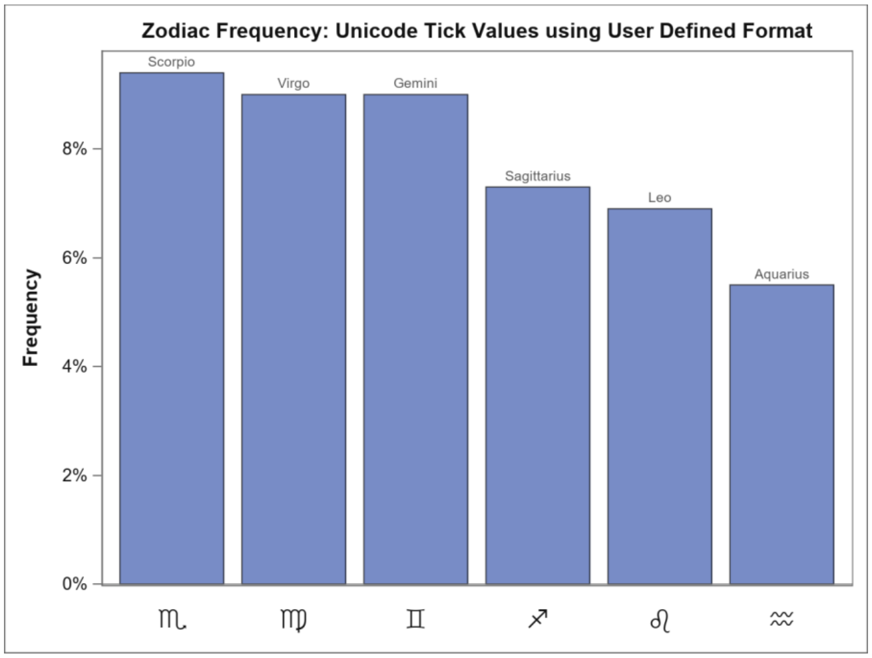

Figure 9. Unicode in Data with User-defined Format

The code snippet for this graph is shown below:

```
data zodiacs;
  length Sign $12;
  input Sign $ Frequency;
 datalines;
 Scorpio      0.094
 Virgo        0.090
 Gemini       0.090
 Sagittarius  0.073
 Leo          0.069
 Aquarius     0.055
 ;
run;

proc format;
  value $ zodiacSymbol
    'Scorpio' = "(*ESC*){unicode '264F'x}"
    'Virgo' =   "(*ESC*){unicode '264D'x}"
    'Gemini' =  "(*ESC*){unicode '264A'x}"
    'Sagittarius' = "(*ESC*){unicode '2650'x}"
    'Leo' =      "(*ESC*){unicode '264C'x}"
    'Aquarius' = "(*ESC*){unicode '2652'x}"
    ;
run;

<proc template ... >
      layout overlay / xAxisOpts=( tickValueAttrs=GraphUnicodeText(size=14)
                         display=(tickvalues)
                         discreteOpts=(tickValueFormat=$zodiacSymbol.) );
        barChartParm x=Sign y=Frequency / dataTransparency=0.3
                         dataLabel=Sign;
      endLayout;
< ... >
```

Observe that we have set the ZODIACSYMBOL user-defined format for the X-axis tick values. Also note the use of the GRAPHUNICODE style element for tick value attributes.

USING UTF-8 SESSION ENCODING

Did you know that in addition to running SAS with your default encoding, SAS also offers a way to run with UTF-8 encoding? On Windows, for SAS 9.4, look under SAS → SAS 9.4 (Unicode Support). UNIX operating system users do not despair: You can find the equivalent command at $SASROOT/bin/sas_u8 . This is not supported on z/OS platforms. Running SAS in this manner does come with some caveats: Please refer to the technical paper, "*Multilingual Computing with SAS® 9.4*" for a good coverage of these issues.

The basic idea of this workaround is to create the data in UTF-8 encoding. Because the SAS session is also running under the same encoding, the values are passed through to the graph renderer without being transcoded.

In your UTF-8 SAS session, you can enter the Unicode data in three ways:

- Directly input the text literals with a UTF-8 aware editor.
- Enter the UTF-8 hexadecimal values.
- If you know the Unicode (UTF-16) values but not the UTF-8 values for your text, you can use the KCVT function. It transcodes the Unicode values into UTF-8

Let us first create some macro variables and a data set using all of these approaches as shown below:

```
data _null_;
  call symput('ulabel', kcvt('03b300200398'x, 'utf-16be', 'utf-8'));
  call symput('udf_v1', kcvt('03b100200393'x, 'utf-16be', 'utf-8'));
  call symput('udf_v2', kcvt('03c000200394'x, 'utf-16be', 'utf-8'));
run;

data uni;
  attrib age label="&ulabel"; /* Data set label in utf-8 */
  attrib sex length=$4;       /* NOTE: length= bytes not chars */
  attrib name length=$32;
  set sashelp.class(obs=6);
  if _n_ = 2 then /* Alice in Katakana: already utf-8 HEX values */
    name='E382A2E383AAE382B9'x;
  if _n_ = 4 then /* Carol in Kannada: utf-8 literals directly typed into
                     a utf-8 capable editor */
    name= 'ಕರೊಲ್' ;
  sex = ifc(sex='M', kcvt('2642'x, 'u16b', 'utf8'),
                      kcvt('2640'x, 'u16b', 'utf8'));
                     /* NOTE: short encoding names */
run;

proc format;
  value utf8_udf
    12 = &udf_v1
    13 = &udf_v2
    OTHER = [Best6.]
    ;
run;
```

A note of caution when using the length specification for text variables in UTF-8 encoding: the length is the number of bytes allocated, *not the number of characters*. So, you need to judiciously estimate the upper bound for your text data byte length. You can use the

KPROPDATA function to handle any partially truncated characters due to incorrect length (a hat tip to You Xie for that suggestion).

Also note that we slipped in a user-defined format. We use this data set and format to display Unicode on axis tick values! Let us now inspect the template and rendering code – nothing tricky here – except for the format override for the AGE column:

```
proc template;
  define statgraph uni_utf8;
    begingraph;
     < ... >
      layout overlay / xaxisopts=(labelAttrs=(size=12 weight=bold));
        scatterPlot x=age y=height / name="sp1" group=sex
              datalabel=name
              dataLabelAttrs=(family=GraphUnicodeText:fontFamily size=16);
        discreteLegend "sp1" / title="Sex" valueAttrs=(size=15 weight=bold)
              < ... >;
      endlayout;
    endgraph;
  end;
run;

proc sgrender template=uni_utf8 data=uni;
 format age utf8_udf10.;
run;
```

Did you notice the font family override in the DATALABELATTRS= above? That is because we want a font family that supports Unicode, but we still want to see the group color effect! The output with Unicode characters in scatter plot data labels, legend entries, X axis tick values, and the X axis label is shown in Figure 10.

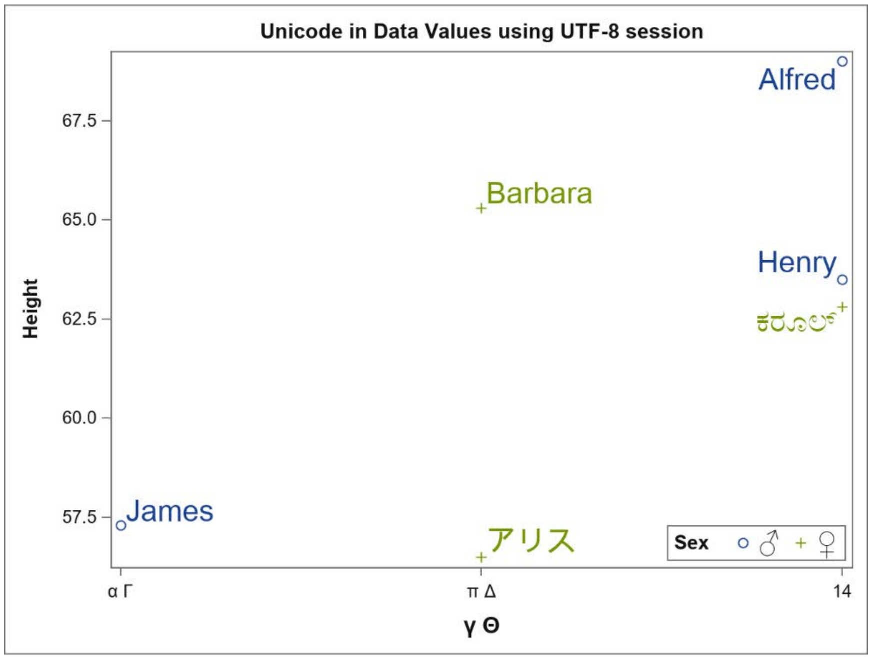

Figure 10. Unicode in Data values using UTF-8 Session Encoding

CONCLUSION

Expression support is built into the Graph Template Language. It lets you be creative in preparing your data for graphing. Use of non-breaking space, text plots, and axis tables are very useful when working with character data. These tips, along with the ways to render Unicode characters in the data, are handy tools for creating novel graphs. Go forth and express yourself!

REFERENCES

- Hebbar and Matange. 2013. "Free Expressions and Other GTL Tips". *Proceedings of the SAS Global 2013 Conference*, Cary, NC: SAS Institute Inc. Available https://support.sas.com/rnd/datavisualization/papers/sgf2013/371-2013.pdf
- SAS Blog "Graphically Speaking". Available at http://blogs.sas.com/content/graphicallyspeaking
- Kiefer, M. "*Multilingual Computing with SAS® 9.4*". Available https://support.sas.com/resources/papers/Multilingual_Computing_with_SAS_94.pdf
- SAS Institute Inc. 2019. *SAS® Functions and CALL Routines Reference.* 5th ed. Cary, NC: SAS Institute Inc. Available https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=lefunctionsref&docsetTarget=titlepage.htm&locale=en
- SAS Institute Inc. 2019. *SAS® Graph Template Language Reference* 5th ed. Cary, NC: SAS Institute Inc. Available https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=grstatgraph&docsetTarget=titlepage.htm&locale=en

ACKNOWLEDGMENTS

Thanks to Sanjay Matange for his contributions to the original paper and to Lingxiao Li and Dave Caira for their review.

RECOMMENDED READING

Matange, S. 2013. *Getting Started with the Graph Template Language in SAS®: Examples, Tips, and Techniques for Creating Custom Graphs,* Cary NC: SAS Institute.

Kuhfeld, W. 2010. *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*. Cary, NC: SAS Institute.

Matange, S., and Heath, D. 2011. *Statistical Graphics Procedures by Example: Effective Graphs Using SAS*, Cary NC: SAS Institute.

CONTACT INFORMATION

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

Prashant Hebbar
SAS Institute Inc.
Prashant.Hebbar@sas.com