

Paper 371-2013

## Free Expressions and Other GTL Tips

Prashant Hebbar and Sanjay Matange, SAS Institute Inc., Cary, NC

### ABSTRACT

The Graph Template Language (GTL) provides many powerful features for creating versatile graphs. The Statistical Graphics Engine in GTL provides novel ways of using expressions to simplify the task of data preparation.

This presentation covers some new ways of using DATA step functions for creating grouped plots based on conditions and to select a subset of the observations. It also illustrates usage of PROC FCMP functions in GTL expressions. Novel uses of non-breaking space for creating chunked graphs and graphs with indented text, as well as workarounds for passing Unicode characters in data columns are discussed. Learn how to express yourself with ease, graphically!

### INTRODUCTION

Output Delivery System (ODS) Graphics allows you to create effective analytical graphs. It comprises the SG Procedures (also called the *ODS Graphics Procedures*) and the Graph Template Language (GTL), as well as the ODS Graphics Designer and the ODS Graphics Editor. This system is now a part of Base SAS starting with SAS® 9.3.

All graphics output in this system requires a STATGRAPH template created using GTL and the TEMPLATE procedure in addition to a procedure that requests a rendering of the template with your data. The SGRENDER procedure allows you to simply associate your data with a template to render a graph. Other procedures offer varying levels of abstraction toward the same purpose.

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, HighLow, BoxPlots, BarCharts, Fit Plots, and more.
- Text and Legends: EntryTitle, EntryFootnote, Entry, DiscreteLegend, ContinuousLegend, and more.
- 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. We also show creative uses of non-breaking space in graphs. Last, we describe 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 <http://support.sas.com/rnd/papers/index.html>. Find the entry "Free Expressions and Other GTL Tips" under the section for SAS Presentations at SAS Global Forum 2013 and download the examples. The code in this paper was tested using SAS® 9.2 TS2M3 software.

### A LITTLE ABOUT GTL EXPRESSIONS

Expression support is built into GTL: the expressions are free! GTL expressions are enclosed within EVAL( ) and are evaluated by the Statistical Graphics Engine in the ODS Graphics system. They are similar to WHERE expressions. However, logical expressions do not subset observations: they return a Boolean value for each observation instead. GTL also supports some functions that are not supported in WHERE syntax.

GTL's IF-ELSE conditional block also supports Boolean expressions in the IF predicate without an explicit EVAL.

For more details, you can refer to the "Runtime Programming Features" section in the *SAS® Graph Template Language Reference*.

### CREATING GROUPS USING CONDITIONS IN EXPRESSIONS

In many situations, you might want to group your observations based on conditions in the data instead of using a specific group variable. Consider the Sashelp.Class data set. If we want to group this data by age into tweens and teens, we 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 < 13) then age_group = 'tween';
  else age_group = 'teen';
run;

```

This data set can then be used with a GTL template to display, say, a scatter plot with `group=age_group`. You can also do this by creating a user-defined format (by using the `FORMAT` procedure) for the `age` column. Next, you could create a template using `PROC TEMPLATE` and then use `PROC SGRENDER` to render the graph.

Using GTL expressions, we can eliminate the first step. Shown here is a code snippet that demonstrates this:

```

proc template;
  define statgraph age_grp_expr;
    beginGraph;
      entryTitle "Groups using Expression";
      layout overlay;
        scatterPlot x=weight y=height / name="sp1"
          markerAttrs=(size=12 weight=bold)
          group=eval(ifc(age < 13, "tween", "teen"));
        discreteLegend "sp1" / title='Age Group' location=inside
          hAlign=right vAlign=bottom;
      endLayout;
      entryFootnote halign=left 'group=eval(ifc(age < 13, "tween", "teen"))';
    endGraph;
  end;
run;

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

```

The graph output from this program is shown here:

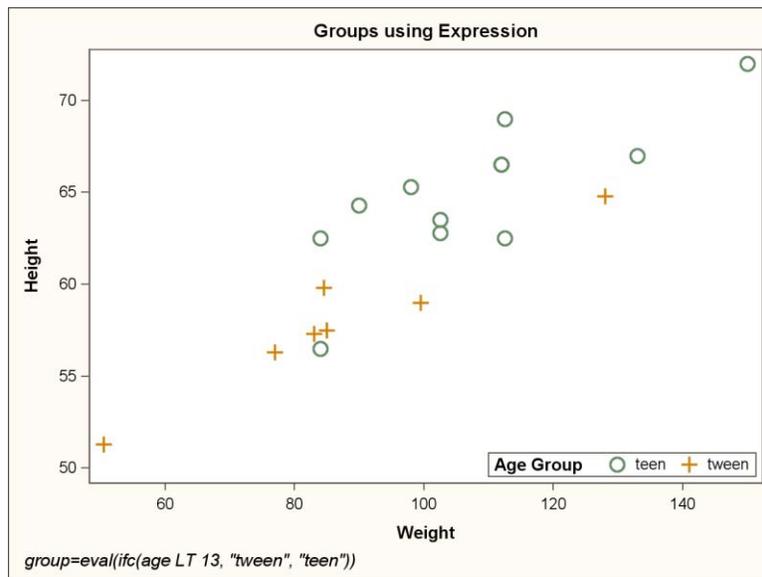


Figure : Creating Groups using Expressions.

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 the boundary age when generating the graph (at run-time as opposed to template compile-time). The following code snippets 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 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 subset observations using expressions in GTL, we can simulate this behavior by setting values outside our subset of interest to missing!

To illustrate this idea, let us consider a way to subset automobiles in the Sashelp.Cars data set with high highway MPGs and those with low highway MPGs, each plotted in its own cell against the MSRP. To do this in the DATA step, we would need to create two new columns based on MPG\_HIGHWAY by setting their values to missing when their respective condition is not met.

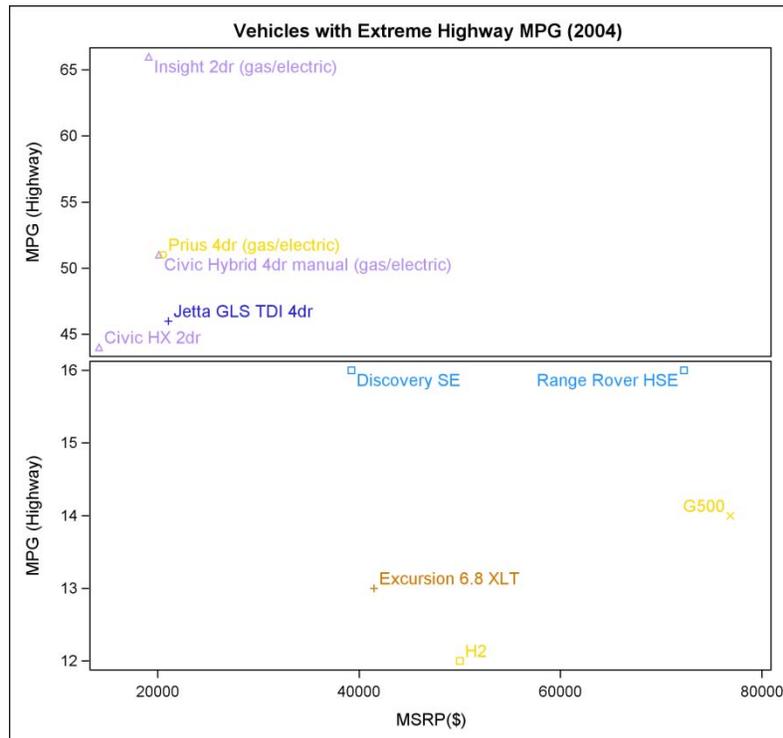
We can do something similar using the IFN function, as shown in the following template code snippet:

```
proc template;
  define statgraph selections;
    dynamic upLimit "low end of high mpg" lowLimit "upper end of low mpg";
    beginGraph / designHeight=600;
      < ... >
      layout lattice / rows=2 < ... >;
        < ... >
        scatterPlot x=eval(ifn(mpg_highway >= upLimit, msrp, .))
          y=eval(ifn(mpg_highway >= upLimit, mpg_highway, .))
          / group=make
          dataLabel=model dataLabelAttrs=(size=10);
        scatterPlot x=eval(ifn(mpg_highway <= lowLimit, msrp, .))
          y=eval(ifn(mpg_highway <= lowLimit, mpg_highway, .))
          / group=make
          dataLabel=model dataLabelAttrs=(size=10);
      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 following graph, with the lower cell showing the subset of autos with a highway mpg  $\leq 16$  and the upper cell showing the subset with a highway mpg  $\geq 44$ .



**Figure : Selecting Observations using Expressions.**

Note: When using this approach, you might have to apply equivalent conditional expressions on all the coordinate options for a given plot. In this example, both X= and Y= arguments in the SCATTERPLOT statement need to use the same condition, but apply it to their respective column variable. Failure to do so will retain the original data ranges for the coordinates of your plot. This is because the data ranges for a plot are computed independently for each coordinate!

## USING FCMP FUNCTIONS

For the 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. For details, please refer to the "FCMP Procedure" section in *Base SAS® Procedures Guide*.

The following program implements FCMP functions for the traditional BMI and one recently proposed by Dr. Nick Trefethen at Oxford University, UK. We also coded two convenience functions for the difference as well as absolute difference between the two.

```

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;

```

We can then use these functions directly in GTL expressions, as shown here. (CALL routines are not supported here.)

```

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 here:

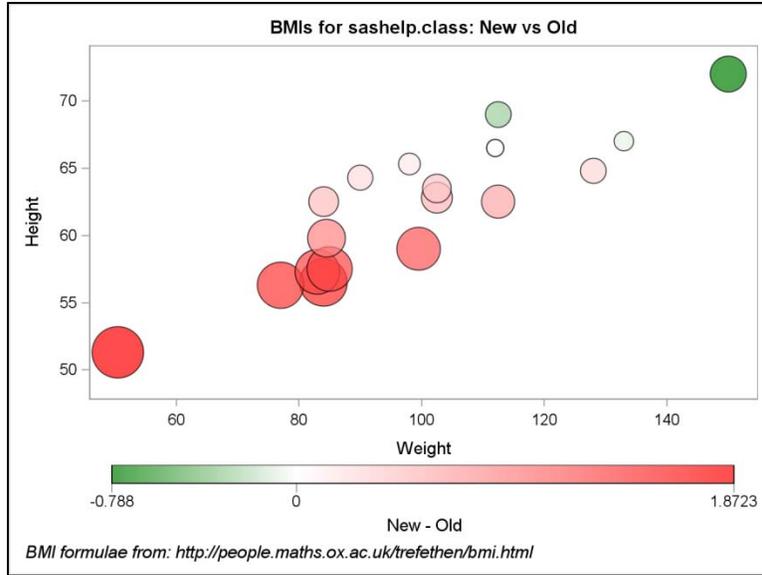


Figure : Using FCMP functions in GTL.

### THE MAGIC OF NON-BREAKING SPACE

Our next bag of tricks deals with retaining the blank character. In most cases, GTL renders your strings after stripping leading *and* trailing blanks. SAS® 9.4 software will have an option to control stripping of labels. In previous releases, you could prevent losing the blanks by using the non-breaking space character 'A0' x – these are not stripped!

### INDENTED TEXT

In certain visualizations, you might want to display a table of values in the graph. In these cases, 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 SAS® 9.4, you will be able to do this painlessly using the AXISTABLE statement. To do this in earlier releases, you could use a scatter plot with the MARKERCHARACTER= option to precisely place the text along the Y axis. Subgroup labels could be indented by prefixing them with non-breaking space!

Here is an example of a forest plot created using this approach. Notice the indented table values on the left side.

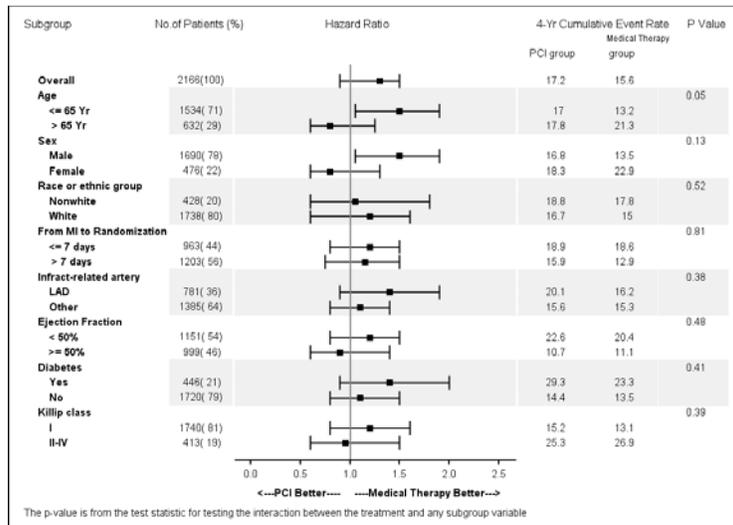


Figure : Indented Text using Non-Breaking Space.

For more details and the full code for this example, please visit the SAS blog at <http://blogs.sas.com/content/graphicallyspeaking/2012/08/12/non-breaking-space/> .

## 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)? In releases prior to SAS® 9.3, you could not access a high-low plot, an explicit position for scatter plot data labels, or even the DRAWTEXT statement.

Suppose that you are creating an Adverse Event plot with SAS® 9.2 software using the scatter plot's MARKERCHARACTER= option to place the labels on the left edge of the bars. Because font sizes are specified in point or pixel units, not in terms of the data space, conjuring up a precise placement for the center of the text label is an exercise in patience and an extremely fragile operation when the data for the graph changes.

Here is what we can do with non-breaking space:

1. Strip the label text of all leading and trailing blanks.
2. Append this label text with blanks equal to its current stripped length – you are essentially doubling the number of characters in the label, with the right half being all blanks!
3. Replace the blanks with non-breaking space (to prevent them from being stripped during rendering).
4. Position the label at the left edge of the graphical element.
5. Ensure that you are using a fixed-width font.

As an illustration, let us create a scatter plot with data labels positioned on the left of the marker symbols for the Sashelp.Class data set. We will use the NAME column for the data labels. As a preliminary step, let us store the current length of this column in the macro variable NAME\_COLLEN as follows:

```
/* Get the name column's allocated length into name_colLen */
data _null_;
  set sashelp.class (obs=1);
  call symputx("name_colLen", vlength(name));
run;
```

Next, we can create the modified data set by appending an equal length of non-breaking space to every name value, including an extra space for padding:

```
data rjClass;
  drop tmpLen;
  attrib name length = $ %eval(2 * &name_colLen); /* double the column length! */
  set sashelp.class;

  if name ne '' then do;
    tmpLen = length(strip(name));

    /* Append an equal length of non-breaking space, + 1 for pad */
    name = cats(name, repeat('A0'x, tmpLen+1));
  end;
run;
```

With the data ready, we create a template and render it as shown here:

```

proc template;
  define statgraph rjText;
    beginGraph;
      < ... >
      layout overlay;
        scatterPlot x=weight y=height;
        scatterPlot x=weight y=height / markerCharacter=name
                   markerCharacterAttrs=(family="SAS Monospace" size=10);
      endLayout;
      < ... >
    endGraph;
  end;
run;

proc sgrender data=rjClass(obs=8) template=rjText;
run;

```

Did you notice the override for fixed-width font in the second scatter plot? Our trick assumes that all characters have the same width. If the font in effect for the text has variable-width characters, the text offsets will be incorrect.

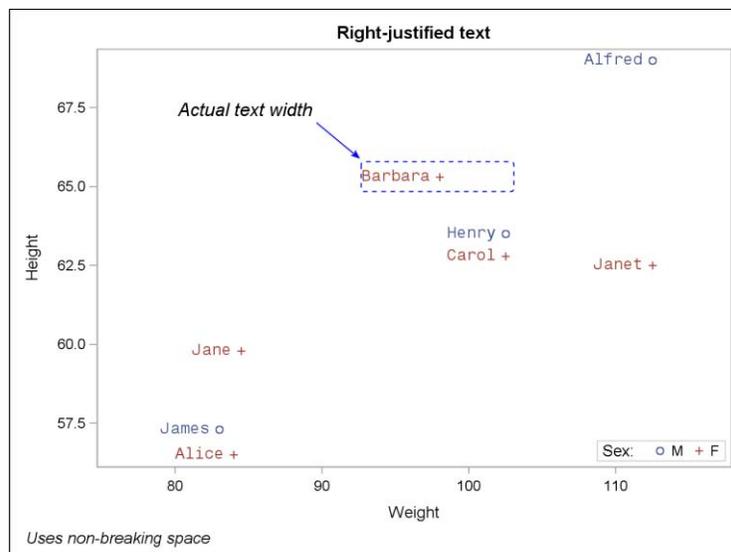


Figure : Right-justified Text using Non-Breaking Space.

A non-trivial example created with SAS 9.2 is shown next.

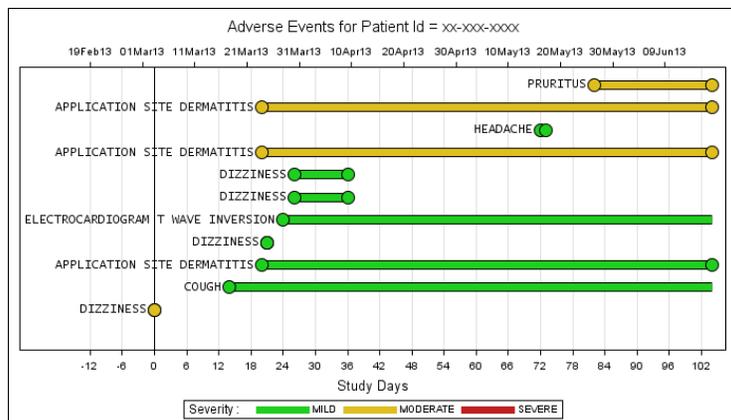


Figure : Adverse Event Plot using Non-Breaking Space.

You can read more about this example and access the full code at the SAS blog here:  
<http://blogs.sas.com/content/graphicallyspeaking/2012/10/21/a-better-ae-timeline/> .

## CHUNKED GRAPHS

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

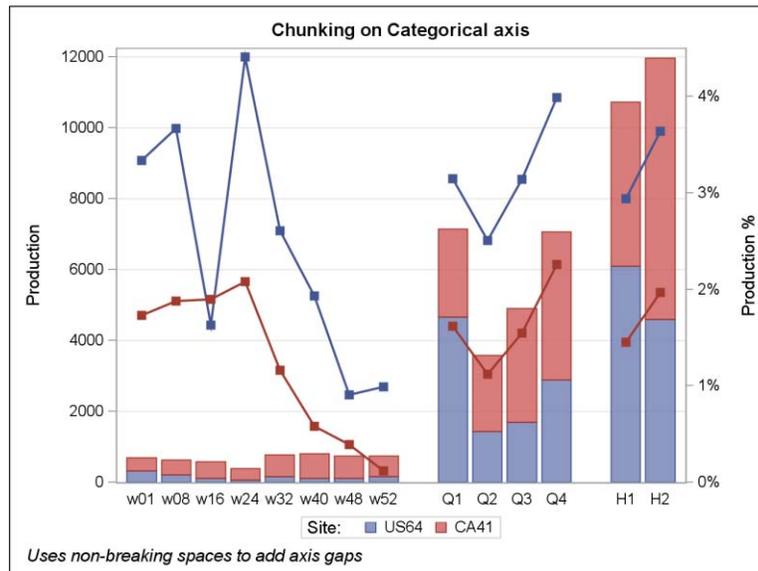


Figure : Breaking Graphs into Chunks with Non-Breaking Space.

Once again, we used non-breaking space to achieve this. The basic idea is to insert non-breaking spaces into the categorical variable corresponding to each desired axis break. Each break uses a different number of these characters. This prevents all these special values from being mapped into the same coordinate when rendering the categorical axis, unlike strings with varying numbers of blanks! The response values for these chunking category values are all missing. Here is the code snippet for the data:

```
/* Data for illustration purposes only! */
data plantProd;
  < ... >
  input prod prod_pct Time $ Site $;
  Time=translate(Time, 'A0'x, '.'); /* map '.' to non-breaking space 'A0'x */
  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;
```

There is one more tweak that can be done here. You can move the mapping of periods to non-breaking spaces into the template code using expressions. We leave that as an exercise for the reader!

## 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 SAS 9.3 and newer). With SAS 9.4, you will be able use annotate data sets in GTL templates as well.

However, there is no support (yet) for using Unicode characters in data columns that are not processed as annotation data (for example, for data labels in a scatter plot). We discuss two workarounds in this section to achieve this.

## MIND YOUR SESSION ENCODING

Did you know that, in addition to your default language version, SAS also offers a version that supports UTF-8 encoding? On Windows, for SAS® 9.2, look under SAS ► SAS 9.2 License Renewal & Utilities ► SAS 9.2 (Unicode Server). In SAS® 9.3, you will find it under SAS ► Additional Languages ► SAS 9.3 (Unicode Support). **UNIX users**, do not despair: You can find the equivalent command at **!SASROOT/bin/sas\_u8** . This is not supported on platforms other than Windows and UNIX. Running SAS in this manner does come with some caveats: Please refer to the technical paper, "*Processing Multilingual Data with the SAS® 9.2 Unicode Server*" for a good coverage of these issues.

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

Assuming you have access to such a SAS session with UTF-8 encoding, let us first generate some macro variables with UTF-8 values in them. Here we have used the KCVT function to transcode UTF-16 values into UTF-8.

```
data _null_;
  call symput('ulabel1', kcvf('03b300200398'x, 'utf-16be', 'utf-8'));
  call symput('udf_v1', kcvf('03b100200393'x, 'utf-16be', 'utf-8'));
  call symput('udf_v2', kcvf('03c000200394'x, 'utf-16be', 'utf-8'));
run;
```

Next, we create the data set for the graph. We insert inline Unicode escapes in the label for the AGE column, a couple of values in the NAME column, and all the values in the SEX column, as follows:

```
data uni;
  attrib age label="&ulabel";
  attrib sex length=$4;
  set sashelp.class(obs=6);
  if _n_ = 2 then
    name=kcvt('03b3002003b203b1'x, 'utf-16be', 'utf-8');
  if _n_ = 4 then
    name=kcvt('06DE002003A3'x, 'u16b', 'utf8'); /* short encoding names */
  if (sex = 'M') then sex = kcvt('2642'x, 'u16b', 'utf8');
  else sex = kcvt('2640'x, 'u16b', 'utf8');
run;

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

Also note that we have slipped in a user-defined format. We will 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;
    beginngraph;
      < ... >
      layout overlay / xaxisopts=(labelAttrs=(size=12 weight=bold));
        scatterPlot x=age y=height / name="sp1" group=sex
          datalabel=name dataLabelAttrs=(size=16);
        discreteLegend "sp1" / title="Sex" valueAttrs=(size=15 weight=bold)
          < ... >;
      endlayout;
    endngraph;
  end;
run;

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

Here is the output with Unicode characters in scatter plot data labels, legend entries, x-axis tick values, and the x-axis label:

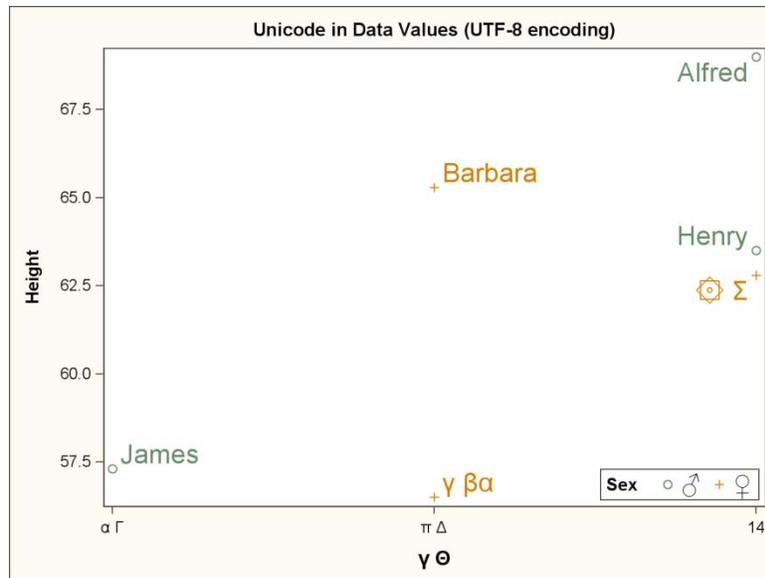


Figure : Unicode in Graph Data using UTF-8 Session Encoding.

## THE .SGE TWO-STEP

In SAS® 9.2, you can generate output in SGE format that is intended for post-processing the graph output using the ODS Graphics Editor. Due to implementation idiosyncrasies, ODS inline Unicode functions in character data columns are correctly rendered in this editor! All you need to do is

1. Generate the .sge file for your graph (using inline Unicode functions in the data).
2. Open the .sge file with the ODS Graphics Editor.
3. Save the graph as an image file.

Let us look at an example of setting up the data for this method:

```
data class_unicode;
  attrib name length=$80 /* Ensure space for long escapes! */
        sex length=$40
        age label='(*ESC*){unicode "03B1"x} - (*ESC*){unicode "03B5"x}';
  set sashelp.class(obs=8);
  if (_n_ = 2) then name = '(*ESC*){unicode "03B1"x} Alice';
  if (_n_ = 4) then name = "(*ESC*){unicode '03A6'x}(*ESC*){unicode '00B2'x}";
  if (sex = 'M') then sex = "(*ESC*){unicode '2642'x}";
  else sex = "(*ESC*){unicode '2640'x}";
run;
```

Notice that, just as in the UTF-8 workaround, we are inserting inline Unicode escapes in the label for the AGE column, a couple of values in the NAME column, and for all the values in the SEX column. (For the extremely curious, this approach does not yet work well for Unicode in user-defined format, hence the omission!)

There is nothing remarkable in the template we are going to use. It is basically a simple a scatter plot with a discrete legend:

```
proc template;
  define statgraph sge_unicode;
    begingraph;
      < ... >
      layout overlay / xaxisopts=(labelAttrs=(size=12 weight=bold));
      scatterPlot x=age y=height / name="sp1"
        group=sex dataLabel=name dataLabelAttrs=(size=14 weight=bold);
      discreteLegend "sp1" / title="Sex" valueAttrs=(size=15 weight=bold)
        < ... >;
    endlayout;
  < ... >
end;
```

```
endgraph;
end;
run;
```

We can now run the SGRENDER procedure, requesting SGE output, which is generated in addition to your standard output.

```
ods listing sge=on;

proc sgrender data=class_unicode template=sge_unicode;
run;
```

Running this code creates a file with a .sge extension along with your listing output file with the same base name. In SAS Display Manager System (DMS), you can navigate to your output node in the Results window and double-click on the SGE item, which launches the ODS Graphics Editor as shown here:

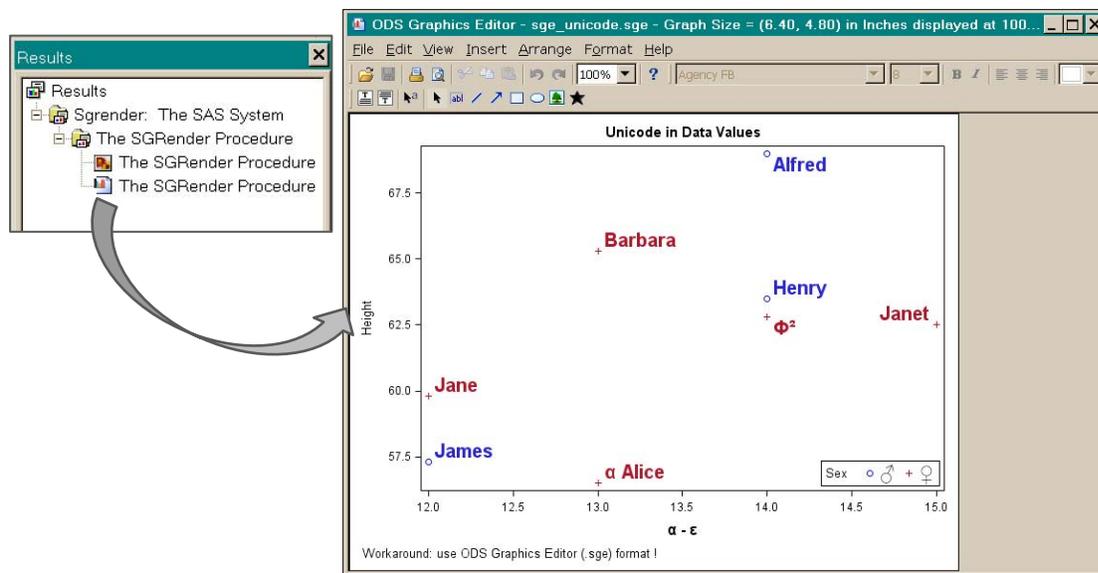


Figure : Unicode in Graph Data using ODS Graphics Editor.

From here, you can save the graph as an image or make further tweaks. For more information about using SGE files, please refer to the *SAS® ODS Graphics Editor: User's Guide*. While this approach is more of a roundabout than a workaround, it does do the trick!

## 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 is a handy trick when working with character data.

These tips, along with the workarounds for rendering Unicode characters in the data, are handy tools for creating novel graphs with ease: go forth and express yourself freely!

## REFERENCES

- SAS Technical Papers. Available at <http://support.sas.com/rnd/papers/index.html>.
- SAS Blog "Graphically Speaking". Available at <http://blogs.sas.com/content/graphicallyspeaking>.
- Kiefer, M. "Processing Multilingual Data with the SAS® 9.2 Unicode Server". Available at <http://support.sas.com/resources/papers/92unicodesrvr.pdf>

## ACKNOWLEDGMENTS

A hat tip to Rick Langston for his ideas and help on working with Unicode in the DATA step.

## RECOMMENDED READING

Cartier, Jeff. 2006. "A Programmer's Introduction to the Graphics Template Language." *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/events/sasglobalforum/previous/index.html>.

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

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

## CONTACT INFORMATION

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

Prashant Hebbar  
SAS Institute Inc.  
Cary, NC 27513

[Prashant.Hebbar@sas.com](mailto:Prashant.Hebbar@sas.com)

Sanjay Matange  
SAS Institute Inc.  
Cary, NC 27513

[Sanjay.Matange@sas.com](mailto:Sanjay.Matange@sas.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.