PharmaSUG 2015 - Paper DV02

Creating Sophisticated Graphics using Graph Template Language

Kaitlyn McConville, Rho®, Inc., Chapel Hill, NC
Kristen Much, Rho®, Inc., Chapel Hill, NC

ABSTRACT

Graph Template Language (GTL) is an excellent tool for customizing the underlying attributes of graphics produced by the SGPLOT/SGPANEL procedures. However, many find learning this relatively new (in production since SAS 9.2) language a challenge. This paper will take an example based approach to creating complex single- and multi-cell statistical graphics. Focus will be placed on syntax and options available in GTL, overlaying graphs of different types, and creating graphs with more complex layouts. The examples provided using data from the Immune Tolerance Network (ITN) and Autoimmune Disease Clinical Trials (ADCT) will enable you to take your graphs to the next level using GTL.

INTRODUCTION

Presenting data graphically is often one of the best ways to better understand clinical trial results. The need for succinct and informative graphs becomes particularly important in the manuscript development process. When working on a manuscript, statistical programmers and research investigators work together to identify which data are most important for inclusion in a manuscript graphic. Furthermore, these individuals work collaboratively to define the specific details that will be included in each graph. This is typically a back and forth process where the research investigators often ask the statistical programmers to produce and/or customize the graphs beyond the capabilities of the SGPLOT/SGPANEL procedures. To overcome this issue, statistical programmers often rely on GTL to meet the needs of the research investigators. GTL is a powerful tool that allows for a great deal of customization through plot layering and the ability to arrange multiple plot types in a single graph. This paper will not only describe the basic tools you need to create a graph with GTL but will also explore 3 specific examples of graphs developed for manuscript publication.

GTL OVERVIEW

In GTL, graphs are built by using plot and layout statements. The plot statements determine how data are represented in the graph and the layout statements determine where the plots are drawn on the graph. More advanced layouts can be used to divide the plot area into multiple independent cells. In addition, statements can be nested so multiple plots can be layered to create highly customizable graphs.

Creating a graph in GTL is a two-step process that involves the TEMPLATE and SGRENDER procedures. The code below outlines the basic components needed for graph creation with GTL.

Step 1: Define the Graph with the TEMPLATE procedure

```
proc template;
  define statgraph <template-name>;
    begingraph / <options>;
      <GTL statements>;
    endgraph;
  end;
run;
```

The TEMPLATE procedure defines the structure of the graph and will also compile and save the template. This code alone will not create the graph. The BEGINGRAPH and ENDGRAPH statements define the outermost container for the graph and must contain all of the GTL statements.

Step 2: Produce the Graph with the SGRENDER procedure

```
proc sgrender data = <data> template = <template-name>;
run;
```

The SGRENDER procedure will associate data with the predefined template and create the graph. If necessary, the same template can be used with multiple, compatible datasets to create additional graphs.

The SAS code below combines the steps above to create a scatterplot of height versus weight by sex.

```
proc template;
  define statgraph example;
    begingraph;
      layout overlay;
        scatterplot x=height y=weight / group=sex markerattrs=(symbol=circlefilled)
                                        name="legend";
        discretelegend "legend" / title="Sex";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=datasetname template=example;
run;
```

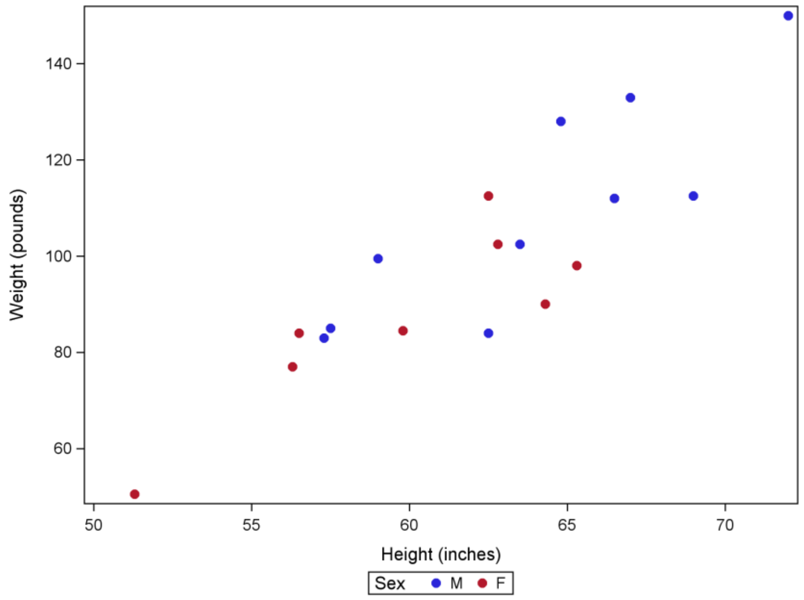

Figure 2. Scatterplot Using GTL

Shortcut!

Still intimidated? One helpful tool for getting started with GTL is to use the TMPLOUT option in the SGPLOT procedure. This creates a SAS program with the GTL code for the specified graph. An example is provided below:

```
proc sgplot data=prep0 tmplout='GTL_Code.sas';
  histogram t2vol;
run;
```

Running the above code with the TMPLOUT option will output a new program called GTL_Code.sas to your current folder directory. This program contains the following GTL code, which will produce the same histogram from the above SGPLOT procedure:

```
proc template;
  define statgraph sgplot;
    begingraph /;
      layout overlay;
        Histogram T2VOL / primary=true binaxis=false LegendLabel="T2 Lesion Volume";
      endlayout;
    endgraph;
  end;
run;
```

This option allows users to start with familiar SGPLOT or SGPANEL code in order to output the GTL framework from which the plot can be further customized.

Now that the basics of GTL have been reviewed, let's look into some examples that highlight the more advanced capabilities of GTL.

EXAMPLE 1: SCATTER PLOT OVERLAID ON BOX PLOT

While you can overlay many different types of plots using SGPLOT/SGPANEL procedures, you cannot overlay scatter plots on box plots. The VBOX and SCATTER statements are incompatible so running both statements within the same SGPLOT procedure shown below will result in the following message:

```
proc sgplot data=datasetname;
  vbox auc / group=trt;
  scatter x=trt y=auc;
run;
```

```
ERROR: Attempting to overlay incompatible plot or chart types.
```

GTL gives users the capability to overlay these plots with fairly simple syntax. Figure 2 below shows scatter plots overlaid on boxplots over time by treatment and demonstrates the following features:

- OVERLAY statement for overlaying the scatterplot on top of the boxplot
- EVAL function for jittering the points along the x-axis
- DRAWTEXT statement for easy annotation
- MERGEDLEGEND statement for combining multiple graph identifiers into one

Braces are used throughout the paper to highlight the nesting in the code. For code that was too long to display within the main text of the paper, <OPTIONS#> was used. The exact code for each <OPTIONS#> is included in the appendix.

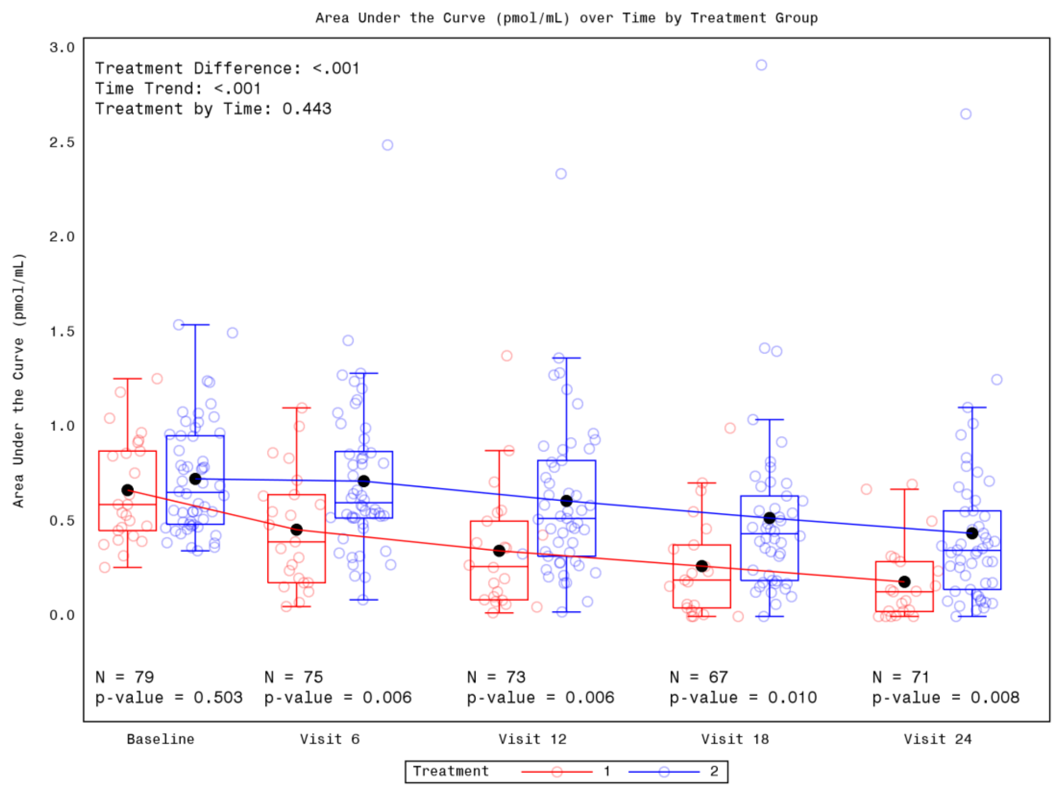

Figure 2. Scatterplot Overlaid on Boxplot

The SAS code to produce Figure 2 can be found below.

```
proc template;
  define statgraph scatterbox;
    begingraph;
      entrytitle "Area Under the Curve (pmol/mL) over Time by Treatment Group";

      layout overlay / xaxisopts=(<OPTIONS1>)
                       yaxisopts=(<OPTIONS2>);
        drawtext textattrs=(size=8pt) "Treatment Difference: &treat_auc." /<OPTIONS3>;
        drawtext textattrs=(size=8pt) "Time Trend: &time_auc." / <OPTIONS4>;
        drawtext textattrs=(size=8pt) "Treatment by Time: &trtbtime_auc." /<OPTIONS5>;
        drawtext textattrs=(size=8pt) "N = &n0_auc." / <OPTIONS6>;
        drawtext textattrs=(size=8pt) "p-value = &p0_auc." / <OPTIONS7>;
        drawtext textattrs=(size=8pt) "N = &n6_auc." / <OPTIONS8>;
        drawtext textattrs=(size=8pt) "p-value = &p6_auc." / <OPTIONS9>;;
        drawtext textattrs=(size=8pt) "N = &n12_auc." / <OPTIONS10>;
        drawtext textattrs=(size=8pt) "p-value = &p12_auc." / <OPTIONS11>;
        drawtext textattrs=(size=8pt) "N = &n18_auc." / <OPTIONS12>;
        drawtext textattrs=(size=8pt) "p-value = &p18_auc." / <OPTIONS13>;
        drawtext textattrs=(size=8pt) "N = &n24_auc." / <OPTIONS14>;
        drawtext textattrs=(size=8pt) "p-value = &p24_auc." / <OPTIONS15>;

        scatterplot x=eval(0.4*rannor(57)+visitn2) y=auc / group=trt name="trt1"
                                                           <OPTIONS16>;
        boxplot x=visitn2 y=auc / group=trt display=(caps mean median connect)
                                  connect=mean name="trt2";

        mergedlegend "trt1" "trt2" / title="Treatment";

      endlayout;

    endgraph;
  end;
run;

proc sgrender data=datasetname template=scatterbox;
run;
```

OVERLAY

In order to overlay the scatter plots on the boxplots, the LAYOUT OVERLAY statement is used. Within LAYOUT OVERLAY, there is a statement for each type of graph to be overlaid. In this case, there are statements for a scatterplot and a boxplot, but this procedure is not limited to overlaying just these two types of graphs.

JITTERING THE POINTS

To randomly jitter the points along the x-axis, the EVAL function specific to GTL is used. In the SCATTERPLOT statement, x is defined as EVAL(0.4*RANNOR(57)+VISITN2). Rather than creating a new x variable in a separate data step, EVAL evaluates the function for use in the graph. This function randomly jitters each point along the x axis by using RANNOR to add a random value from the normal distribution. The multiplier of 0.4 is used to control the width of the jittering so that the points stay within the width of the boxplot.

BOXPLOT FEATURES

Within the BOXPLOT statement, there are several options available to further customize the output. The DISPLAY option specifies the features the boxplot will display. In this case, the MEAN and CONNECT options are used to connect the means over time by treatment group.

ADDING TEXT

One option for graph annotation is to use the DRAWTEXT statement. In this case, the graphs are annotated with p-values and counts. Text can be written anywhere on the graph with just a few options—no need for going through the hassle of defining an annotate dataset! Since the text to be drawn could change as the data change, annotation is done with predefined macro variables.

LEGEND CUSTOMIZATION

The MERGEDLEGEND statement creates a legend for the graph that represents identifiers from both graphs merged into one. In Figure 2 'Treatment 1' is either the red line or red circle, depending on if you are referencing the boxplot or scatter plot. To display both identifiers, both graph statements must have a NAME identified because these names are called in the MERGEDLEGEND statement.

EXAMPLE 2: MULTI-CELL GRAPH WITH NESTED LAYOUT LATTICE

Research investigators often request to combine different types of plots into a single graph. The LAYOUT LATTICE statement can be used to divide the plot area into a multi-cell grid of graphs. Within each cell of the lattice, additional lattice statements can be used to create complex layout configurations. Figure 3 uses a nested lattice statement to display three types of summary graphs for different baseline MRI characteristics and demonstrates the following features:

- LATTICE statement for creating a complex nested layout
- GRIDDED statement to overlay descriptive statistics
- ROWGUTTER and COLUMNGUTTER options for inserting gap space

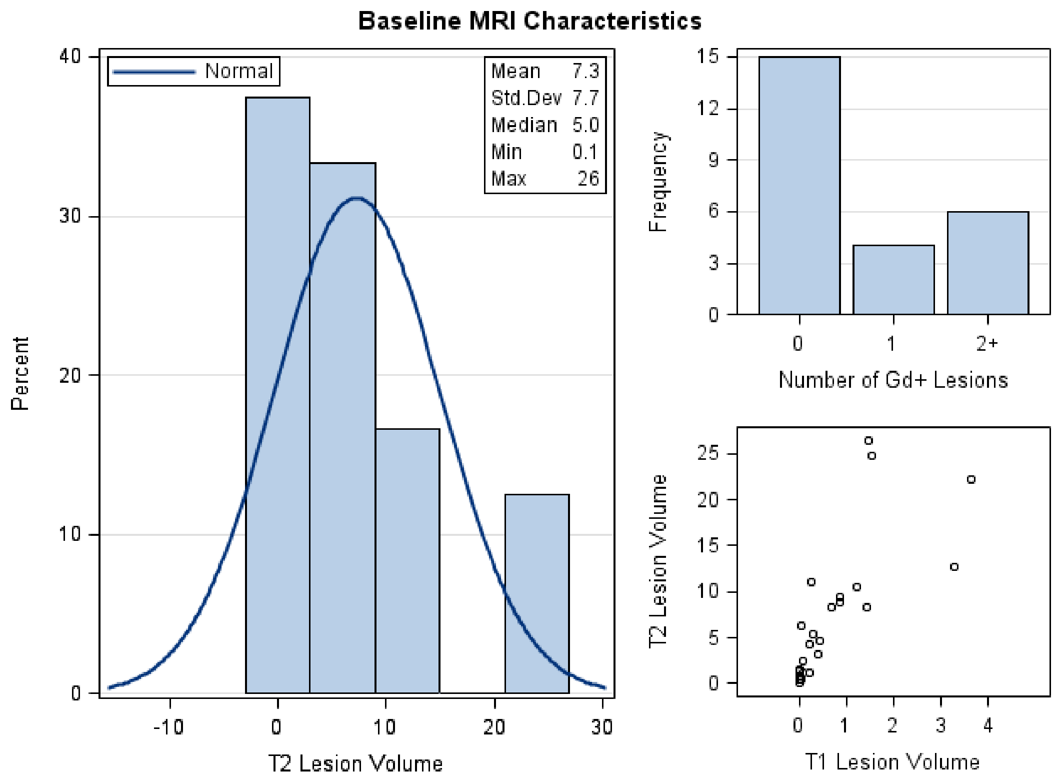

Figure 3. Multi-cell Graph with Nested Layout Lattice

The SAS code to produce Figure 3 can be found below.

```
proc template;
  define statgraph mri;
    begingraph;
      entrytitle 'Baseline MRI Characteristics';

      layout lattice /rows=1 columns=2 columnweights=(0.6 0.4) columngutter=.5cm;
        layout overlay / yaxisopts=(griddisplay=on);
          histogram t2vol / binaxis=false;
          densityplot t2vol / lineattrs=graphfit name='density' legendlabel=’Normal’;
          discretelegend 'density' / location=inside halign=left valign=top;
            layout gridded / rows=5 columns=2 opaque=true border=true
                             autoalign=(topright);
              entry halign=left 'Mean';     entry halign=right "&mean";
              entry halign=left 'Std.Dev'; entry halign=right "&std";
              entry halign=left 'Median';   entry halign=right "&med";
              entry halign=left 'Min';      entry halign=right "&min";
              entry halign=left 'Max';      entry halign=right "&max";
            endlayout;
        endlayout;

        layout lattice / rows=2 columns=1 rowgutter=.5cm;
          layout overlay / yaxisopts=(griddisplay=on linearopts=(tickvaluelist=
                               (0 3 6 9 12 15)));
            barchart x=gadc;
          endlayout;

          layout overlay / xaxisopts=(linearopts=(viewmax=4));
            scatterplot x=t1vol y=t2vol;
          endlayout;
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=datasetname template=mri;
run;
```

CREATING THE COMPLEX LAYOUT

The SAS code above uses a 1 row, 2 column LAYOUT LATTICE statement with a nested 2 row, 1 column LAYOUT LATTICE statement to create a plot with 3 distinct plot areas. The COLUMNWEIGHTS option with the LAYOUT LATTICE statement designates the column widths. In other words, this option specifies the fractional proportion of each cell relative to the overall grid width. The number of entries in this list should directly correspond to the number of columns and the sum of these weights should be 1.0. A similar option for ROWWEIGHTS can be used in cases where there are multiple rows with different row heights desired.

LAYOUT GRIDDED FOR DESCRIPTIVE STATISTICS

Located on the left, the first cell has several plot statements overlaid, which includes a legend for the normal density plot. A nested 5 row, 2 column LAYOUT GRIDDED statement is used to create an inset table of text. It is nested within the LAYOUT OVERLAY along with the other plot statements. Each ENTRY statement in the LAYOUT GRIDDED statement becomes a row in the table. Predefined macro variables are used to display the appropriate descriptive statistic value.

OPTIONS FOR COSMETIC UPDATES

When using a nested LAYOUT LATTICE statement, the default spacing between plots can make the graph appear cramped, particularly when the plots have axis labels. In this case, the COLUMNGUTTER and ROWGUTTER options are used to add vertical and horizontal gap space between the plots.

EXAMPLE 3: MULTI-CELL PANELED SERIES PLOTS

When reporting clinical trial results in manuscripts, there is often a limit to the number of displays allowed. Using a multi-cell layout is crucial for getting multiple graphs into one display. Figure 4 shows the individual trajectory for a handful of subjects over time at different visits for two distinct outcomes and demonstrates the following features:

- Combining the GRIDDED, DATAPANEL, and PROTOTYPE layout statements to create a complex layout
- DISCRETELEGEND statement for customizing the look of the legend

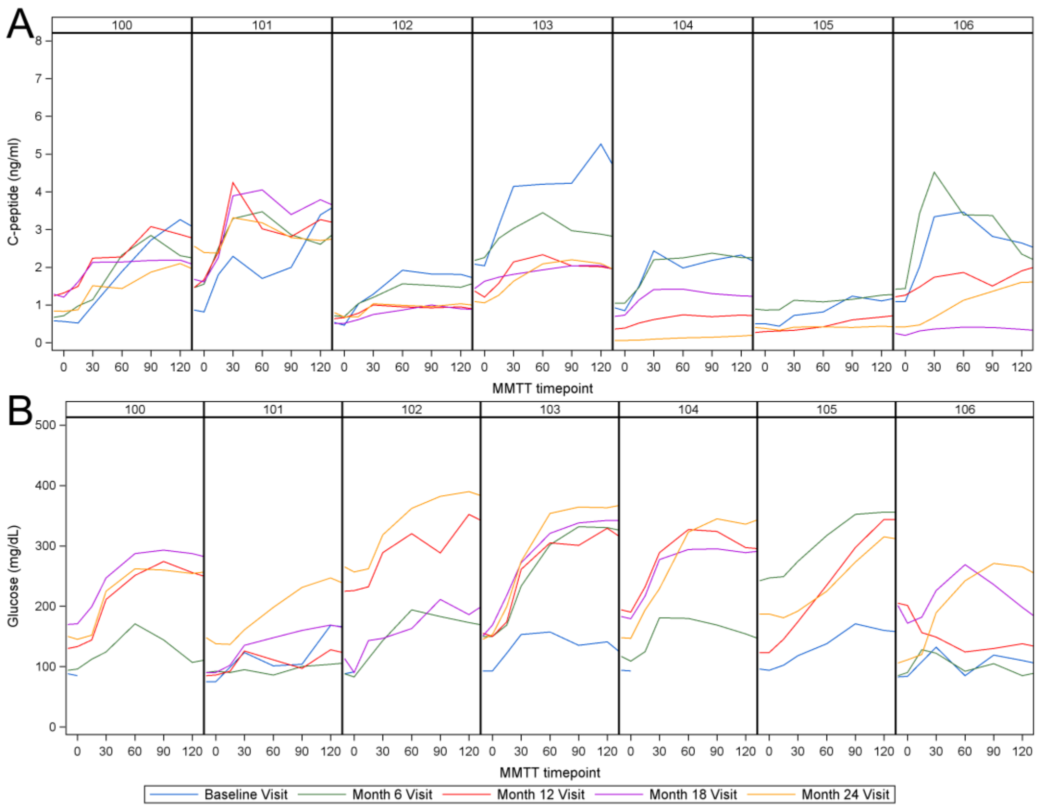

Figure 4. Multi-cell Paneled Series Plot

The SAS code to produce Figure 4 can be found below.

```
proc template;
  define statgraph twobyone;
    begingraph / designwidth=11in designheight=8.5in;
      layout gridded / rows=2 columns=1 rowgutter=15;

      drawtext textattrs=(size=30pt) "A" / <OPTIONS17>;
      drawtext textattrs=(size=30pt) "B" / <OPTIONS18>;

        layout datapanel classvars=(id) / <OPTIONS19>;
          layout prototype;
            seriesplot x=visit0 y=cpep / legendlabel="Baseline Visit" name="y0"
                                         lineattrs=(<OPTIONS20>);
            seriesplot x=visit6 y=cpep / legendlabel="Month 6 Visit" name="y1"
                                         lineattrs=(<OPTIONS21>);
            seriesplot x=visit12 y=cpep / legendlabel="Month 12 Visit" name="y2"
                                          lineattrs=(<OPTIONS22>);
            seriesplot x=visit18 y=cpep / legendlabel="Month 18 Visit" name="y3"
                                          lineattrs=(<OPTIONS23>);
            seriesplot x=visit24 y=cpep / legendlabel="Month 24 Visit" name="y4"
                                          lineattrs=(<OPTIONS24>);
          endlayout;
        endlayout;

        layout datapanel classvars=(id) / <OPTIONS25>;
          layout prototype;
            seriesplot x=visit0 y=glucose / lineattrs=(<OPTIONS26>);
            seriesplot x=visit6 y=glucose / lineattrs=(<OPTIONS27>);
            seriesplot x=visit22 y=glucose / lineattrs=(<OPTIONS28>);
            seriesplot x=visit18 y=glucose / lineattrs=(<OPTIONS29>);
            seriesplot x=visit24 y=glucose / lineattrs=(<OPTIONS30>);
          endlayout;
        endlayout;

      discretelegend "y0" "y1" "y2" "y3" "y4" / across=5 border=true
                                                valueattrs=(size=10pt);

      endlayout;
    endgraph;
  end;
run;

proc sgrender data=datasetname template=twobyone;
run;
```

CREATING THE COMPLEX LAYOUT

There are a few different layout statements that make this output possible—GRIDDED, DATAPANEL, and PROTOTYPE. The GRIDDED statement treats each cell in the DATAPANEL statement independently and enables the stacking of the paneled graphs A and B. The DATAPANEL statement allows users to panel graphs by a classification variable. In this case, the CLASSVAR is subject id. The PROTOTYPE statement must be nested within the DATAPANEL statement because it instructs the DATAPANEL statement what to display in each panel of the classification variable. In this case, it provides instructions for five separate series plots.

LEGEND CUSTOMIZATION

The DISCRETELEGEND statement creates a legend for the graph, much like the MERGEDLEGEND statement. Within a plot statement, a NAME must be specified because it will be required when defining the DISCRETELEGEND. Paired with the NAME statement, a LEGENDLABEL can be used to specify how the data will be labeled within the legend. If LEGENDLABEL is not specified, the label for the variable will be used instead.

CONCLUSION

GTL is an extremely powerful tool allows users to produce highly customizable graphs. It has also been proven to be invaluable when working with research investigators to create graphs for publication. With the ability to nest layout statements and overlay multiple plots of different types, the possibilities are seemingly endless! The examples explored in this paper lay the foundation you need to create your own complex graphs in GTL.

ACKNOWLEDGMENTS

This research was performed as a project of the Immune Tolerance Network and the Statistical and Clinical Coordinating Center for Autoimmune Disease Clinical Trials, which are supported by the National Institute of Allergy and Infectious Diseases of the National Institutes of Health (award numbers NO1-AI-15416, HSN272200800029C and HHSN272200900057C).

RECOMMENDED READING

- Matange, Sanjay. 2014. "Up Your Game with Graph Template Language Layouts." *Proceedings of the PharmaSUG 2014 Conference.* Available at http://www.pharmasug.org/proceedings/2014/DG/PharmaSUG-2014-DG14-SAS.pdf.
- Matange, Sanjay. October 2013. Getting Started with the Graph Template Language in SAS®: Examples, Tips, and Techniques for Creating Custom Graphs. Cary, NC: SAS Institute.

CONTACT INFORMATION

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

Name: Kaitlyn McConville
Enterprise: Rho, Inc.
Address: 6330 Quadrangle Dr.
City, State ZIP: Chapel Hill, NC 27517
Work Phone: 919-408-8000
Fax: 919-408-0999
E-mail: kaitlyn_mcconville@rhoworld.com

Name: Kristen Much
Enterprise: Rho, Inc.
Address: 6330 Quadrangle Dr.
City, State ZIP: Chapel Hill, NC 27514
Work Phone: 919-408-8000
Fax: 919-408-0999
E-mail: kristen_much@rhoworld.com

APPENDIX

EXAMPLE 1 <OPTIONS>

<OPTIONS1>

```
display=(tickvalues) linearopts=(tickvaluelist=(1 6 12 18 24) viewmin=0 viewmax=26
tickvalueformat=vis.)
```

<OPTIONS2>

```
display=(label tickvalues) linearopts=(tickvaluelist=(0 .5 1 1.5 2 2.5 3) viewmin=-.5
viewmax=3)
```

<OPTIONS3>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=-1
y=92 justify=left
```

<OPTIONS4>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=-1
y=89.5 justify=left
```

<OPTIONS5>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=-1
y=87 justify=left
```

<OPTIONS6>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=-1
y=13.5 justify=left
```

<OPTIONS7>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=-1
y=11 justify=left
```

<OPTIONS8>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=4
y=13.5 justify=left
```

<OPTIONS9>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=4
y=11 justify=left
```

<OPTIONS10>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=10
y=13.5 justify=left
```

<OPTIONS11>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=10
y=11 justify=left
```

<OPTIONS12>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=16
y=13.5 justify=left
```

<OPTIONS13>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=16
y=11 justify=left
```

<OPTIONS14>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=22
y=13.5 justify=left
```

<OPTIONS15>

```
anchor=topleft width=80 widthunit=percent xspace=datavalue yspace=graphpercent x=22
y=11 justify=left
```

<OPTIONS16>

```
markerattrs=circle datatransparency=0.7
```

EXAMPLE 3 <OPTIONS>

<OPTIONS17>

```
anchor=bottomleft width=80 widthunit=percent xspace=graphpercent yspace=graphpercent
x=1 y=95 justify=center
```

<OPTIONS18>

```
anchor=bottomleft width=80 widthunit=percent xspace=graphpercent yspace=graphpercent
x=1 y=47 justify=center
```

<OPTIONS19>

```
rows=1 columns=7 rowaxisopts=(label='C-peptide (ng/ml)' linearopts=(tickvaluelist=(0 1
2 3 4 5 6 7 8) viewmin=0 viewmax=8)) columnaxisopts=(label='MMTT timepoint'
linearopts=(tickvaluelist=(0 30 60 90 120) viewmin=-10 viewmax=120))
headerlabeldisplay=value
```

<OPTIONS20>

```
lineattrs=(pattern=1 color=cmykFF970030)
```

<OPTIONS21>

```
lineattrs=(pattern=1 color=cmyk90149580)
```

<OPTIONS22>

```
lineattrs=(pattern=1 color=cmyk00FFFF00)
```

<OPTIONS23>

```
lineattrs=(pattern=1 color=cmyk37FF0026)
```

<OPTIONS24>

```
lineattrs=(pattern=1 color=cmyk0068FF00)
```

<OPTIONS25>

```
rows=1 rowaxisopts=(label='Glucose (mg/dL)' linearopts=(tickvaluelist=(0 100 200 300
400 500) viewmin=0 viewmax=500)) columnaxisopts=(label=’MMTT timepoint’
linearopts=(tickvaluelist=(0 30 60 90 120) viewmin=-10 viewmax=120))
headerlabeldisplay=value
```

<OPTIONS26>

```
lineattrs=(pattern=1 color=cmykFF970030)
```

<OPTIONS27>

```
lineattrs=(pattern=1 color=cmyk90149580)
```

<OPTIONS28>

```
lineattrs=(pattern=1 color=cmyk00FFFF00)
```

<OPTIONS29>

```
lineattrs=(pattern=1 color=cmyk37FF0026)
```

<OPTIONS30>

```
lineattrs=(pattern=1 color=cmyk0068FF00)
```