Paper 121-2010

Integrating Multiple SAS/GRAPH® Procedures to Generate a Custom Image

Joshua M. Horstman, First Phase Consulting, Inc., Indianapolis, IN

Abstract

SAS/GRAPH® offers a rich set of tools that can be used to programmatically generate custom image files. Using the MWSUG website header as a sample project, this paper provides an introduction to several lesser-known SAS/GRAPH procedures and demonstrates how to use them together. PROC GMAP is used to draw a custom map of the states in the MWSUG region. Text and other graphical elements are produced using PROC GSLIDE. Finally, the various pieces are arranged together with PROC GREPLAY. Various GOPTIONS parameters are discussed as they relate to the output of an image file of the desired type and size. No prior SAS/GRAPH experience is assumed.

Introduction

While the SAS/GRAPH software is typically used to produce graphical representations of data, its capabilities extend far beyond this. The MWSUG website header image (shown at right) was created programmatically using several lesser-known SAS/GRAPH procedures. This paper walks through the process of creating that image step by step.

MWSUG
MidWest SAS® Users Group

This detailed examination highlights key functions of the GMAP, GSLIDE, and GREPLAY procedures as well as various graphics options. We describe the creation of the individual elements that constitute the complete image – first the map, then the text. Next we explain how to bring these elements together into the desired arrangement. Finally, we consider how to direct the output into an external GIF file.

This discussion is based on SAS® version 9.1.3.

Step 1: Drawing a Map with PROC GMAP

We begin by creating the map component of our desired image. To accomplish this task, we employ the GMAP procedure. PROC GMAP produces maps that provide a visual representation of how a response variable changes over a geographic area. Two ingredients are necessary for this: a map data set, and a response data set.

The Map Data Set

The map data set contains the specific coordinates that define the geographic units to be mapped. For our project, we need a map data set that defines the boundaries of each of the 12 states in the MWSUG region. While such a data set could be constructed from scratch, the SAS/GRAPH software includes an extensive collection of map data sets covering the entire world at various levels of political subdivision. These data sets are located in a library called MAPS that is automatically loaded when a SAS session begins.

We will make use of the MAPS.US data set, which includes coordinates that define the boundaries of all 50 U.S. states and the District of Columbia. This data set contains X and Y coordinates based on a projection of the map onto a two-dimensional surface. The MAPS library also includes data sets that contain unprojected longitude and latitude values. Unprojected map data can be useful if, for example, one wishes to create a custom projection using the GPROJECT procedure, but can produce unexpected results if fed directly into PROC GMAP.

Although the MAPS.US data set includes states that are not part of our desired MWSUG region, it is not necessary for us to subset the map data set. Rather, our response data set will contain response values for only those states we wish to include in our map. PROC GMAP will only display those map areas for which response data are available (unless that behavior is overridden using the ALL option).

The Response Data Set

The response data set can contain one or more response variables. It also must contain identification variables that indicate the geographic region with which each response is associated. These must be the same identification variables that are used in the map data set.

The MAPS.US data set uses two identification variables: STATE and SEGMENT. The STATE variable is a numeric code known as a FIPS code, which is a standard code used by the U.S. federal government. The SEGMENT variable is used to distinguish between portions of states that consist of multiple, noncontiguous areas such as Michigan. Since our response will not vary by SEGMENT, we need only include STATE in our response data set.

Our response data set also needs to contain a response variable. However, we do not have an actual response variable because we are not truly interested in representing how some quantity varies by geographic area. We seek only to produce a single-colored map, and so we engage in a ploy. We create an artificial response variable called INCLUDE which happens to take the arbitrary value of 1 for states we desire to include in our map and which is missing otherwise.

We construct our response data set by simply taking a subset of our map data set corresponding to the states we wish to include. Rather than ascertaining the FIPS code for each of the 12 states in our region, we make use of a SAS function called FIPSTATE which converts FIPS codes to the more familiar two-character state postal codes. The following DATA step code creates the response data set.

```
data mwsug_states;
  set maps.us;
  where fipstate(state) in
    ('IA','IN','IL','KS','MI','MN','MO','OH','ND','NE','SD','WI');
  include=1;
run;
```

Invoking the GMAP Procedure

With both map and response data sets in hand, we proceed to call the GMAP procedure. The PROC GMAP statement itself specifies the map and response data sets using the MAP= and DATA= options, respectively. We also use the GOUT= to designate a SAS catalog in which graphics output will be saved. This will be important later when combining graphic elements into the final image. The catalog name is arbitrary. Here we use GRAFCAT.

Before describing the remaining statements in the GMAP procedure, a word about SAS/GRAPH colors is in order. There are several ways to specify colors in SAS/GRAPH. Since our graphic is for a website, we utilize the RGB hexadecimal color-naming scheme commonly used in web design. In SAS/GRAPH, this consists of the prefix CX followed by three two-digit hexadecimal values corresponding to the red, green, and blue components of the color.

PROC GMAP is capable of producing several types of two-dimensional and three-dimensional maps. Each map type is associated with a corresponding statement included in the procedure syntax. A map in which areas are colored or patterned based upon the value of a variable is known as a choropleth map and is produced in the GMAP procedure using the CHORO statement.

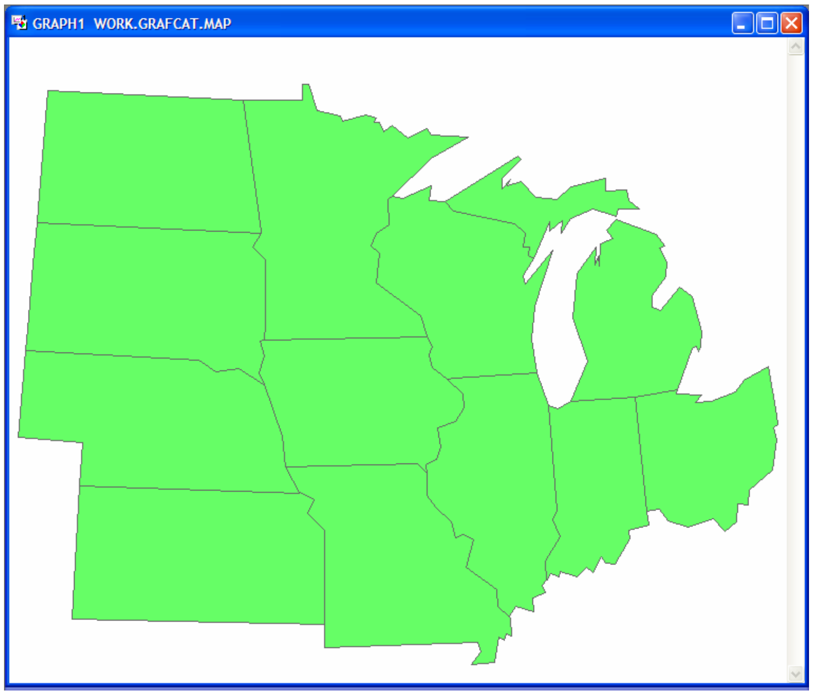
In the CHORO statement, we specify the response variable we wish to use (INCLUDE) as well as several options. We use the COUTLINE= option to choose the color in which map areas will be outlined (CX666666, a medium gray) and the WOUTLINE= option to set the width of the outline to 1. We include the NOLEGEND option to instruct PROC GMAP not to include a legend on our map. Finally, we use the NAME= option to specify an entry name under which the output will be stored in the graphics catalog. Here we use the arbitrary name MAP.

Two additional statements are needed to complete our call to PROC GMAP. The ID statement indicates which variables are used to identify map regions. These variables must appear in both the map and response data sets. Since our response varies only by STATE and not by SEGMENT, we need only include STATE on the ID statement.

We also include the PATTERN1 statement to specify the color and fill pattern that PROC GMAP will use. This statement is not strictly part of the GMAP procedure. It is a general SAS/GRAPH statement used by several procedures. Since our response variable only has one distinct value, we only provide one pattern here. Specifically, we use a COLOR of CX66FF66 (a light green) and a VALUE of MSOLID which indicates a solid color fill.

Putting these pieces together, we arrive at the following PROC GMAP code which produces the output shown below.

```
 proc gmap map=maps.us data=mwsug_states gout=grafcat;
   id state;
   pattern1 color=CX66FF66 value=msolid;
   choro include / coutline=CX666666 woutline=1 nolegend name='map';
run;
quit;
```



Step 2: Formatting Text with PROC GSLIDE

Having completed our custom map, we now turn our attention to drawing the text. This is accomplished using the GSLIDE procedure. PROC GSLIDE is designed for the creation of text slides for presentations. It can be used to generate text and line elements to be combined with other graphics output. However, before calling the procedure, we must configure a few SAS/GRAPH graphics options.

Setting Up the Graphics Output Area with GOPTIONS

SAS/GRAPH will allow us to specify the dimensions of elements we create in one of several different units. These units include assorted absolute units (inches, centimeters, etc.) as well as the PCT unit which allows us to specify sizes as percentages of the graphics output area. We use the latter for simplicity and because it allows us to easily scale our entire graphics output by simply adjusting the size of the graphics area. This will be important later during final assembly of the complete image. We could include this unit designation every time a dimension is specified, but it is easier to simply make it the default graphics unit by including the graphics option GUNIT=PCT on the GOPTIONS statement.

Since we specify the sizes of our text elements as percentages of the graphics output area, it is helpful to specify an absolute size for the graphics output area. Otherwise, the default size of the graphics area is dependent upon the display device. In interactive SAS, this can even vary based upon the current size of the SAS window. The particular size one should specify is dependent upon the desired appearance of the output. Here we use an output area that is 3.9 inches in width and 1.35 inches in height. These values were determined by a process of trial and error and are set using the HSIZE= and VSIZE= graphics options on the GOPTIONS statement.

We also use the CBACK= graphics option on the GOPTIONS statement to set the background color to a dark blue.

Adding Text with the NOTE Statement

Strictly speaking, the PROC GSLIDE statement is the only statement in the GSLIDE procedure. However, the procedure can make use of the TITLE, FOOTNOTE, and NOTE statements, which are general SAS/GRAPH statements, in order to include text on a slide. Here we use the NOTE statement.

The NOTE statement provides several different options that allow us to control the formatting and positioning of the specified text. We use the HEIGHT=, COLOR=, and FONT= options to set the text height, color, and font, respectively. Using the MOVE= option, we can designate the position of the text by providing a pair of coordinates. The coordinates can be either absolute (measured from the origin) or relative (measured from previously drawn text in the same statement). We indicate that coordinates are relative by preceding them with a plus or minus operator.

A single NOTE statement can contain multiple text strings and options. Each option affects any text that follows it in the statement unless it is reset by another option. Consider the following NOTE statement. It draws the text "MWSUG". The MOVE= option positions the text at the left edge of the graphics area and 42% of the way between the bottom and top of the graphics area. Next, it specifies a height of 70 and a font of boldfaced Georgia. It selects a light blue color which is applied to underlining of thickness 3. It then changes the color to white for the text.

```
note move=(0,42) height=70 font='Georgia/bold' color=CX00CCFF u=3
  color=white 'MWSUG';
```

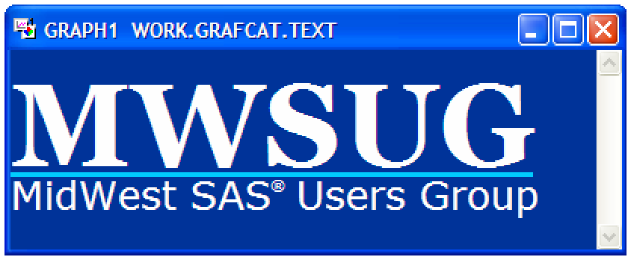
The following NOTE statement provides another example. It first draws the text "MidWest SAS". It then uses relative coordinates to draw the registered trademark symbol in a higher position and smaller font. Another set of relative coordinates return us to the original vertical alignment and font size to complete the line of text with "Users Group".

```
note move=(0,20) height=24.5 color=white font=Verdana 'MidWest SAS'
     move=(+0,+10) height=9 '®'
     move=(+2,-10) height=24.5 'Users Group';
```

Invoking the GSLIDE Procedure

All that remains is for us to embed these NOTE statements within a call to the GSLIDE procedure. We include two options on the PROC GSLIDE statement. As with PROC GMAP, we use the GOUT= option to designate a graphics catalog in which the output will be saved and the NAME= option to assign a name under which the output will be stored. The complete PROC GSLIDE code is exhibited below along with its corresponding output.

```
goptions cback=CX003399 gunit=pct hsize=3.9in vsize=1.35in;
proc gslide gout=grafcat name='text';
  note move=(0,42) height=70 font='Georgia/bold' color=CX00CCFF u=3
    color=white 'MWSUG';
  note move=(0,20) height=24.5 color=white font=Verdana 'MidWest SAS'
       move=(+0,+10) height=9 '®'
       move=(+2,-10) height=24.5 'Users Group';
run;
quit;
```



Step 3: Combining the Elements with PROC GREPLAY

We now have all the pieces we need to construct our desired image. We will use the GREPLAY procedure for this purpose. PROC GREPLAY provides a host of functionality related to displaying and managing graphics output. For this project, we call PROC GREPLAY twice. The first call creates a template which describes how our graphic elements will be positioned together. The second call replays our existing graphics output from the graphics catalog into that template.

Creating the Template

A template consists of one or more rectangular panels, and each panel is defined by the coordinates of its four corners. Within the GREPLAY procedure, we use the TDEF statement to define a template. The TDEF statement requires a name for the template (limited to 8 characters even in SAS version 9). The template can then later be used by reference to this name. Here we use “mytemp”.

After the template name, we define two panels: one for our map, and one for our text. Each panel is assigned a panel number. Any numbers can be used, but it generally makes sense to number the panels sequentially. The panel number is followed by eight options that set the x- and y-coordinates of the lower-left (LLX= and LLY=), lower-right (LRX= and LRY=), upper-left (ULX= and ULY=) and upper-right (URX= and URY=) corners.

Coordinates are specified as a percentage of the graphics output area, regardless of the current setting of the GUNIT= graphics option. We position the map panel with x-coordinates from 2 to 32 and y-coordinates from -13 to 113. These coordinates extend beyond the interval from 0% to 100% of the graphics area, meaning that a portion of the panel will be outside the graphics output area. That portion of the panel will be cropped from the output. We do this intentionally to remove some of the unwanted whitespace from the top and bottom of the GMAP output.

We position our second panel with x-coordinates from 34 to 100 and y-coordinates from 0 to 100. This panel will be used for our GSLIDE output and will occupy approximately the right-hand two-thirds of the graphics area from top to bottom. These values are arbitrary and were obtained through a process of trial and error based on aesthetics.

The TDEF statement is the only statement we need to include in addition to the PROC GREPLAY statement itself. On the PROC GREPLAY statement, we use the TC= option to designate a catalog to be used for templates (which we arbitrarily call “tempcat”) and the NOFS option to indicate that we do not wish to run the GREPLAY procedure in its default interactive window mode. The procedure code is shown below.

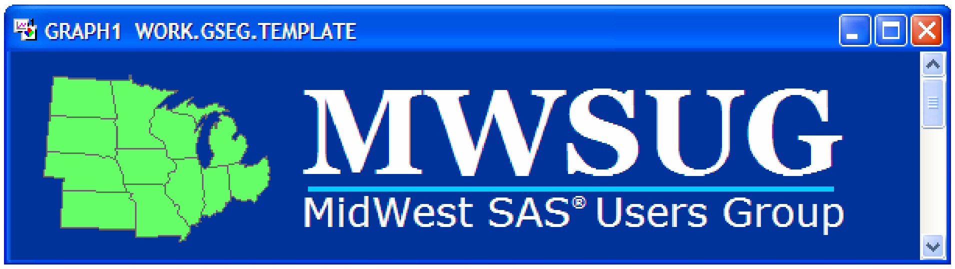
```
proc greplay tc=templib nofs;
  tdef mytemp
    1 / llx=2   lly=-13
        ulx=2   uly=113
        urx=32  ury=113
        lrx=32  lry=-13

    2 / llx=34  lly=0
        ulx=34  uly=100
        urx=100 ury=100
        lrx=100 lry=0
        ;
run;
quit;
```

Using the Template

Everything is now in place to generate the complete graphic. We need only invoke the GREPLAY procedure once again. This time, we use the TEMPLATE statement to identify the template we wish to use and the TREPLAY statement to indicate which graphics output from the graphics catalog should be placed in each of the numbered panels.

We use the IGOUT= option on the PROC GREPLAY statement to specify the graphics catalog from which graphics output should be replayed. We also use a GOPTIONS statement to set the size of the graphics output. The code and output are displayed below.

```
goptions hsize=5.75in vsize=1.35in;
proc greplay igout=grafcat tc=templib nofs;
  template mytemp;
  treplay 1:map 2:text;
run;
quit;
```



Step 4: Writing the Image to a File

Satisfied with the appearance of our graphics output, we now wish to write this image to a file so that it can be used on the MWSUG website. This involves both redirecting the output to a file as well as using a different device driver to achieve the desired format. For this project, we will create a .GIF file.

First, we associate a SAS fileref with the desired external file by adding the following FILENAME statement to the top of the code file.

```
  filename outfile 'c:\mwsug\mwsug_webheader.gif';
```

We accompany this with another FILENAME statement at the bottom of the code file to clear the fileref.

```
  filename outfile clear;
```

In addition, we redirect our graphics output to this fileref by using the GSFNAME= graphics option in a GOPTIONS statement immediately following the FILENAME statement at the top of the code file. We use the DEVICE= graphics option in the same GOPTIONS statement to select the GIF device driver.

We also add the XPIXELS=, YPIXELS=, XMAX=, and YMAX= graphics options to our GOPTIONS statement. XPIXELS= and YPIXELS= specify the size of the display area (i.e. the external file) in pixels. We set XMAX= and YMAX= to the size of our GREPLAY output in inches. This ensures that the output will be scaled appropriately to match the dimensions of the image file.

We begin the GOPTIONS statement with the RESET=ALL option to ensure we are starting with all options set to their default values. We also relocate the GUNIT= and CBACK= options to this initial GOPTIONS statement for clarity.

```
goptions reset=all device=gif gsfname=outfile
  xpixels=575 ypixels=135 xmax=5.75in ymax=1.35in gunit=pct cback=CX003399;
```

The complete SAS code is shown in Appendix A. In the final code, we use the NODISPLAY graphics option with the GMAP and GSLIDE procedures so that these intermediate outputs are not rendered.

Conclusion

In this paper, we've explored the basic functionality of the GMAP, GSLIDE, and GREPLAY procedures and surveyed several important graphics options. We've seen a relatively straightforward way to employ these procedures to programmatically draw an image to our exact specifications. This method has many benefits over typical image drawing tools. The precise dimensions and layout of the image can be controlled at a very fine level, and variations in fonts and colors can be made with only slight modifications to the code.

Contact Information

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

Joshua M. Horstman
First Phase Consulting, Inc.
8921 Nora Woods Drive
Indianapolis, IN 46240
Phone: 317-815-5899
E-mail: j.horstman@firstphaseconsulting.com

Appendix A: SAS Code

```
/*** Open the file ***/

filename outfile 'c:\mwsug\mwsug_webheader.gif';

goptions reset=all device=gif gsfname=outfile gunit=pct
  xpixels=575 ypixels=135 xmax=5.75in ymax=1.35in cback=CX003399;

/*** Generate the map ***/

data mwsug_states;
  set maps.us;
  where fipstate(state) in
    ('IA','IN','IL','KS','MI','MN','MO','OH','ND','NE','SD','WI');
  include=1;
run;

goptions hsize=1.35in vsize=1.35in nodisplay;
proc gmap map=maps.us data=mwsug_states gout=grafcat;
  id state;
  pattern1 color=CX66FF66 value=msolid;
  choro include / coutline=CX666666 woutline=1 nolegend name='map';
run;
quit;

/*** Generate the text ***/

goptions hsize=3.9in vsize=1.35in nodisplay;
proc gslide gout=grafcat name='text';
  note move=(0,42) height=70 font='Georgia/bold' color=CX00CCFF u=3
    color=white 'MWSUG';
  note move=(0,20) height=24.5 color=white font=Verdana 'MidWest SAS'
       move=(+0,+10) height=9 '®'
       move=(+2,-10) height=24.5 'Users Group';
run;
quit;

/*** Create the template ***/

proc greplay tc=templib nofs;
  tdef mytemp
    1 / llx=2   lly=-13
        ulx=2   uly=113
        urx=32  ury=113
        lrx=32  lry=-13

    2 / llx=34  lly=0
        ulx=34  uly=100
        urx=100 ury=100
        lrx=100 lry=0
        ;
run;
quit;

/*** Put it all together ***/

goptions hsize=5.75in vsize=1.35in display;
proc greplay igout=grafcat tc=templib nofs;
       template mytemp;
       treplay 1:map 2:text;
       run;
       quit;

/*** Close the file ***/
filename outfile clear;
```