

Green Graphics: Intelligent Graphical Layout

Paul Hamilton, Amgen Inc.

Abstract

SAS/Graph[®] has the ability to produce a wide variety of output objects. In many applications, the same type of graph may be produced for each of a number of linked data elements. This paper will examine how to seamlessly blend different graphical images into a desired layout. Basic knowledge of the graphical procedures is assumed, but the paper will examine how to store graphical output, how to use prebuilt grids for the layout, how to specify your unique layout, and how to flow graphics into the layout. Finally, a couple of minor tricks will greatly enhance the readability of the whole image.

Introduction

SAS/Graph[®] can produce many types of graphic output images (bars, line plots, 3D surfaces, etc) through the use of the **Proc Gxxx** procedures. SAS/graph has been around for a very long time (although it has been greatly enhanced recently) and probably many SAS programmers are familiar with the basics of plotting. Like most procedures, the SAS/Graph procedures support the use of the **By** statement, so that separate graphs are produced for each combination of by variables. These could include (among many) a separate graph for each unique:

- airplane being built
- subject in an experiment
- patient in a clinic
- time unit (day, month, etc.)

Of course, the usual conditions apply to the use of a **by** statement: either the data must first be sorted (with or without Proc Sort), or an index must exist on the data set before the by processing will work properly.

A basic idea in producing graphs is the *device driver*, which lets SAS know what graphical format to use to produce the image. The choice of a device driver also often determines the file extension on the output file (if one is created). SAS/Graph

provides dozens of file output types, some going back nearly three decades (and some look like it!) This paper will focus on modern, high quality drivers (i.e. PNG, PDF, and SVC).

If SAS/Graph code is executed in most environments (even the mainframe if you have the proper terminal or emulator), graphical images will automatically pop up on the screen. *While* these are often of low to moderate resolution, they are quick and easy ways to garner feedback on how the graphical statements are defining the desired image. SAS/Graph statements (specified on the **goptions** statement) define where permanent output file containing one or many graphical images may be stored.

Behind the Scenes

It is not at all obvious, but every time a graphical image is produced, it is stored in a temporary storage location; a *graphics catalog*. This is not an unusual outcome in the SAS System; running Proc Format usually sends a format or informat specification into a format catalog. The actions of SAS/Graph are quite analogous. Before combining different images onto one layout, one needs to know how the intermediate results are stored.

When you run **Proc Gplot**, it creates a catalog entry for each graphic image produced; normally once per procedure call, unless you are using a **by** statement. By default these entities are stored in the work library, in a catalog called **GRSEG**. These entries have default names; the first is called **GPLOT**, then **GPLOT1**, **GPLOT2**, etc. It is possible to use SAS/Graph successfully for years, and never notice or care that these entries exist. However, once you know they exist, we can do marvelous things with them.

One can also control the name of the graphical catalog entries, and also a description (much like a SAS data set name and label). However, since these entities live

Green Graphics: Intelligent Graphical Layout

Paul Hamilton, Amgen Inc.

inside a catalog, they have a different naming convention than SAS data sets do. SAS data sets have a two level name: [*library name*].[*member name*]. Entries in a catalog start the same, but also add a catalog entry name and a type (this is also true for formats and informats). So if you run **Proc Gplot** once, the default entity it creates will be called [**work.gseg.gplot.grseg**]. (Actually even data sets have a “hidden” third name: [Data] or [View]. This is why you can create a View with the same name as an actual data set without SAS becoming confused.) If you wish to override the default name and descriptions, utilize code like the following:

```
proc gplot data = sashelp.shoes;
  plot sales * returns /
    name = 'Try1'
    description = 'First Try';
quit;
```

After executing this procedure, the entity [**work.gseg.try1.grseg**] will be ready for you to use. Unfortunately, executing a graphic procedure with a **by** statement does not allow the specification of the myriad entities being created, unless you use the macro language to completely control the workflow (as a subsequent example will demonstrate).

Examine Shoes

The **SASHELP** library contains a handy sample data set named **shoes**. It contains data on retail sales of shoes, with Region and Subsidiary as nice grouping fields. First we can examine how the various Regions compare to one another:

```
proc gchart data = sashelp.shoes;
  hbar region /
    sumvar = sales;
  title1 h=16pt
    'Overall Sales by Region';
run;
quit;
```

Next, we focus on a specific set of regions and examine how their subsidiaries perform:

```
proc gchart data = sashelp.shoes;
  where (region not in ('Canada'
```

```
    'United States')));
  where also (region not contains
    'Europe');
  by region;
  hbar subsidiary /
    sumvar = sales;
  title1 h=16pt
    'Overall Sales by Subsidiary'
    'Within Region';
  title2 h=14pt
    'Region: #byval(region)';
run;
quit;
```

Now we experience our first problem: the Region/Subsidiary plots each take up a separate page, and several of the regions are fairly small and do not use much of the page. Additionally, it is difficult for the eye to scan back and forth across the various regions to make visual comparisons. A traditional method of coping with this issue is to print each of the separate plots on a separate page, and then paste the resulting mass of graphs on the wall of a dedicated room. Of course, as the data changes and the graphs are updated, this process gets expensive and tedious.

The Graphics Layout

By default, each graphical image uses the entire graphical area (device dependent) to display an image. SAS/Graph provides 3 tools to help the sophisticated graphic developer more control, all through the use of one procedure:

1. **Proc Greplay** can play graphic images into specified areas of graphic templates
2. **Proc Greplay** contains multiple pre-built graphic templates, and
3. **Proc Greplay** can be used to define your own custom templates.

We need to understand how graphic templates are defined before using them, so first consider the built-in templates SAS provides. In the **sashelp.templt** catalog are 21 preexisting graphic templates. The names are descriptions are self-defining; **H2** is 2 horizontal boxes; **whole** is the whole

Green Graphics: Intelligent Graphical Layout

Paul Hamilton, Amgen Inc.

image area. A common variation is to add space, so **H2S** is 2 horizontal boxes with additional space.

After one has populated the **work.gseg** catalog with some graphical images, the next trick is to “pour” them into a graphics template. This is accomplished by code:

```
proc greplay
  igout = work.gseg nofs;
  tc sashelp.templt;
  template H2S;
  treplay 1: plot1
          2: plot4;
run;
quit;
```

The **igout** statement tells **greplay** where the graphical images will come from. The **tc** statement identifies the source of the Template Catalog to “hold” the images. The **template** statement specifies which template in the catalog will be used (**h2s** in this case). Finally, the **Treplay** statement identifies which saved graph is placed into each region segment. Since **h2s** has 2 regions, this statement has two specifications (plots 1 and 4 were chosen at random).

Executing the code above will produce one graphical image, with two separate graphs replayed into horizontal columns. Examination of the resulting image reveals a problem that will be corrected later; the images are clear but somewhat distorted (compressed horizontally) visually.

Custom Templates

Now that we have used pre-built templates provided with SAS/Graph software, we can explore additional features of Proc Greplay that enable us to create our own custom template layouts.

The question of many separate graphs can be combined reasonably into one layout depends on two things:

- The quality of the final output image: different SAS/Graph device drivers

differ greatly in their resolution and clarity.

- The complexity of the graphs themselves – not just the plotting symbols inside the frame, but also the text annotations and titles required.

Given graphs of moderate complexity and simple text annotation, placing six graphs onto one page is reasonable. This is accomplished by using **Proc Greplay** twice: first to build our custom template, then once again to replay our chosen graphs into the template.

```
proc greplay nofs tc = work.tcat;
  tdef sixbox des = '6 Boxes'
    1/LLX= 0 LLY= 45
      ULX= 0 ULY= 90
      URX= 34 URY= 90
      LRX= 34 LRY= 45
    2/LLX= 35 LLY= 45
      ULX= 35 ULY= 90
      URX= 66 URY= 90
      LRX= 66 LRY= 45
    3/LLX= 67 LLY= 45
      ULX= 67 ULY= 90
      URX=100 URY= 90
      LRX=100 LRY= 45
    4/LLX= 0 LLY= 0
      ULX= 0 ULY= 44
      URX= 34 URY= 44
      LRX= 34 LRY= 0
    5/LLX= 35 LLY= 0
      ULX= 35 ULY= 44
      URX= 66 URY= 44
      LRX= 66 LRY= 0
    6/LLX= 67 LLY= 0
      ULX= 67 ULY= 44
      URX=100 URY= 44
      LRX=100 LRY= 0
    7/LLX= 0 LLY= 0
      ULX= 0 ULY=100
      URX=100 URY=100
      LRX=100 LRY= 0;
run;
quit;
```

The first line of code invokes the procedure and specifies with the **tc** keyword (*template catalog*) a catalog used to store the template we are creating. In this case it is being stored in the **work** library, but it could easily be stored in any permanent SAS library and shared by many users in your organization.

Green Graphics: Intelligent Graphical Layout

Paul Hamilton, Amgen Inc.

The **nofs** keyword stands for *No Full Screen*, and prevents Proc Greplay from starting up in interactive mode.

The next statement defines the name (*sixbox*) and description of the template being created. This name and description are arbitrary but obviously meaningful names are helpful. Next follow a series of sub-clauses, one for each panel being defined. The panel number is followed by a slash, and then a set of 8 coordinates. These represent the X,Y pairs of the Lower and Upper Left, Lower and Upper Right of each panel. The numbers for each of the 8 coordinates represent percent of the image. Here we have defined two rows of three columns. Despite the catalog entry name, there are in fact 7 boxes defined – the last one is the whole graphic image. A single semicolon terminates this long statement.

Remember that we will be placing six separate images into the *sixbox* template, and each one will have its own axis labels and subheading. However, it is handy to be able to place a standard title centered over the entire graphic page; this is the purpose of the final panel. The code below will make a nice looking title, to be placed later.

```
proc gslide
    name = 'Title'
    description = 'Main Title';
    title1 h=18pt "Shoe Regions With
6 Plots/Page";
run;
quit;
```

Finally we need to actually create the combined image. Following whatever **goptions** you prefer to set up your final destination, the following code will create the final product.

```
proc greplay igout = work.gseg
nofs;
    tc work.tcat;
    template sixbox;
    treplay 1:plot1
            2:plot2
            3:plot3
            4:plot4
            5:plot5
            6:plot6
```

```
7:Title;
```

```
run;
quit;
```

This code initiates **proc greplay** again, and the **tc** statement identifies the source of our custom template. The **template** statement calls out the name of the actual template to use (*sixbox*), and finally the **treplay** places the entry **plot1** into the first panel, **plot2** into the second, and so on until the **title** entry is placed into region # 7 (whole page).

The code sequence covered above works as intended and produces a perfectly serviceable plot (depending on the quality of your final output destination). However, careful inspection of the graphical image may leave one discontent. While legible, the sub-plots are subtly distorted, and the text inside them can be difficult to read.

However, a helpful Technical Support note explained all the details necessary for resolving these issues. We will only need to tweak a couple of statements to greatly increase the visual quality and appeal of the final result.

Made to Size

When we built each of the images that were later placed into a much smaller template panel, it appeared at the time that it was intended to fill the entire page. So when the images are placed into a much smaller area (with a different aspect ratio), then they are naturally distorted. The solution is not obvious, but turns out to be simple to achieve.

Once your final combined output has been constructed, measure the resulting width and height of a panel. Then, rebuild the images again, but add to your previous code just a couple of extra options:

```
goptions
    nodisplay
    vsize = 3.75in
    hsize = 3.25in;
```

The vertical and horizontal sizes are specified on the **vsize** and **hsize**

Green Graphics: Intelligent Graphical Layout

Paul Hamilton, Amgen Inc.

parameters. The **nodisplay** option simply prevents the graphs from appearing on the screen while we are storing them in the work graphics catalog. Now that SAS/Graph knows that the images are destined for specific size and shape, it more builds them more intelligently. Continue on to build the subplots and the graphic title.

Before the final step of creating a higher quality output, we need just a few more graphics options:

```
goptions
    hsize = 0
    hpos = 0
    vsize = 0
    vpos = 0
display;
```

Everything else following this is the same as before; use **proc greplay** to combine the 7 graphical elements into one output file. The size and position elements above are set to zero, which has the effect of restoring their values to the default for the device driver being employed. The **display** option indicates to SAS/Graph that the resulting graph should be written to the output destination, not just stored in a graphics catalog.

Conclusion

SAS/Graph[®] provides many tools to combine similar or disparate graphical images into one cohesive whole. Doing so has a number of benefits:

- Improves the ability to compare across grouping variables
- Permits customization of grouping of like images
- If printed to hard copy, reduces the amount of printed output while retaining all the original information.

The techniques presented in this paper are very generic, and can serve a wide variety of graphical reporting needs. They should work well in any modern (V8+) version of The SAS System (no need to wait for V9.2) and are stable and robust.

References

The final tip to vastly improve the quality of multi-panel templates came from TS499:

support.sas.com/techsup/technote/ts499.html

As usual, there is no attribution to a specific SAS employee for this tip. The author would like to express profound gratitude to the anonymous employee.

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 brands and product names are registered trademarks or trademarks of their respective companies.

Contact Information

Contact the author with questions or comments at:

Paul Hamilton
Amgen, Inc.
hamiltonp@amgen.com

For a copy of the complete program containing all the code in the presentation, please contact the author and request "ShoeGraphs.sas".