Producing Data-Driven Plots for Microsoft Word®
Using SAS/GRAPH® and the SAS® Macro Language

Robert W. Graebner, Quintiles, Inc., Kansas City, MO

ABSTRACT
SAS/GRAPH is a powerful tool that provides a broad range of graphing capabilities. When combined with the power of the SAS macro language, SAS/GRAPH allows users to develop flexible applications that can automate the production of graphs. The ability to output graphs into the most common graphic file formats allows publication quality graphs to be easily created and imported into the commonly used personal computer applications. This paper presents the basics of using the SAS macro language to automate the process of producing graphs and importing them into Microsoft Word. Four basic uses of the macro language are covered; automatically generating plots for all defined subsets of data, tailoring each plot to the data in a particular subset, outputting the graphs to computer graphics metafiles (CGM) for importing into Microsoft Word and generating Visual Basic source code to insert the figures into a Word document and resize them. Examples from a clinical trial study in which several thousand graphs were produced are used to illustrate the key concepts involved. This paper is intended for users with basic SAS/GRAPH experience who are interested in expanding their graphing capabilities through the use of the SAS macro language.

INTRODUCTION
Graphical analysis can be a significant part of many pharmaceutical research studies. This fact, combined with the increasing complexity of studies, can result in the need to produce a large number of complex graphs. In addition, it is often necessary to produce publication quality graphs that can be inserted directly into the study report. This process can be automated using SAS/GRAPH, the SAS macro language, Microsoft Word and Visual Basic. Automating the process increases efficiency while ensuring completeness and consistency.

The examples presented in this paper are based on a Phase I clinical trial in which the client requested both individual and summary plots, with both linear and log-linear scales, for the pharmacokinetic (PK) data collected on the study drug, two of its metabolites and also on several pharmacodynamic (PD) analytes, for each treatment. An example of such a figure is given in Figure one. The product of all combinations of these factors resulted in several thousand figures being required. It was also requested that the figures be provided in a Microsoft Word document. Major graphing tasks such as this justify the up-front cost of developing an automated solution.

The SAS macro language can be used in four basic ways to automate this task. The first is to build a list of combinations of factors (subject, analyte, treatment, etc.) that are present in the data. The macro then loops through each combination found and calls the plotting macro for that combination. The second use for macros is to analyze the data in each subgroup and adjust the plots (title, labels, axis scale, etc.) accordingly. The third...
use is to generate graphic files with filenames containing the figure number. The fourth is to generate Visual Basic source code to be run as a Word macro to automate the process of inserting the figures into a document and resizing them.

GENERATING GRAPHS

The first step in the process was to automate the calling of the plotting routine. Generating thousands of graphs without the use of macros would be very inefficient. The sheer volume of source code would be difficult to manage and ensuring that all subsets of data had been plotted would be tedious. The same plotting routine would be called repeatedly with only a few differences in the data subsetting statement and in a few of the plotting options. Developing a macro was the only approach that made sense.

The clinical study report required both individual and summary plots. In order to generate the individual plots, it was necessary to build a list of study subjects that could be processed sequentially. Constructing an array of subject identification numbers would allow a macro to loop through the list of subjects and call the plotting macro for each. While the SAS macro language does not facilitate arrays directly, it is easy to simulate arrays by defining macro variables that have sequential numbers as a suffix. The macro variables that comprise the array can then be referenced by using two-pass macro resolution. As an example, if you had ten subjects, their identification numbers could be stored in macro variables named SUB0, SUB1 through to SUB9. Using a macro variable as an array index (i for example), you could reference the corresponding array element with the macro reference &&SUB&i. This reference has a double ampersand so it will be resolved in two passes. If for example i = 2, then on the first pass &&SUB&i will resolve to &SUB2. On the second pass, &SUB2 will resolve to a string containing the subject identification number for the third subject. If you define the macro variables within a data step by calling the SYMPUT routine, you can make this process data driven. That is, you can create a pseudo array of subjects for any dataset regardless of how many subjects there are. An example of this approach is given below.

```sas
%let _data = secgenpc.pkdata;
proc sort data= &_data out= subj nodupkey;
  by subject;
run;

data _null_
  set subj end=last;
  call symput('subno' || left(_n_), put(subject, $4.));
  if last then call symput('numsub', _n_);
run;
```

In this example, PROC SORT is used with the nodupkey option to create a temporary dataset containing one observation per subject. A data null step is then used to loop through each subject and the SYMPUT routine is called to store the current subject identifier into a sequentially named macro variable. The macro variable name is derived by concatenating the literal string 'subno' with the automatic data step index _n_. When the last subject is reached, a call to SYMPUT is used to store the final value of _n_ into a macro variable to be used as the upper range of the %DO loop used to process each subject. For each subject, a macro is called that uses the same technique as above to create a pseudo array of analytes that are present for each subject. A portion of this macro is shown below.

```sas
proc sort data= &_data (where=(subject= &_subject)) out=pdanalyt nodupkey;
  by testdesc;
run;

data stuff;
  set pdanalyt end= last;
  call symput('analytes' || left(_n_), analyte);
  if last then call symput('numrec', _n_);
run;
```

This array is also looped through and two macros will be called within each iteration of the loop. The first creates a temporary dataset containing all of the data pertaining to the current subject, analyte and specified study days. In the example below, the macro PDIPREP is used to prepare the PD data for a subject.

```sas
%macro pdiprep(_source=, _subject=, _analyte=, _days=);
  data pktreat;
    set &_source;
    if analyte = "&_analyte" and subject = &_subject and rxdy in (&_days);
run;
%mend pdiprep;
```

The resulting dataset is then used with PROC GPLOT to produce each of the graphs that make up the figure by calling a macro called PKIPILOT as shown below.
The PKIPlot macro then calls PROC GLPOT using the specified parameters and stores the plot within the specified catalog (grafout) with the specified name (linplt).

```
proc gplot data= &_data gout= grafout anno=Linearit;
plot &_yvar * &_xvar = &_catvar / haxis=axis1
   vaxis=axis2
   name='linplt'
   legend=legend1;
run;
```

The ANNO= option specifies the name of a SAS annotate dataset that contains annotation text for the plot. For these figures, the word 'Linear' or 'Log' appeared in the upper right corner. The following data step creates the annotate dataset for the linear plot. The XSYS and YSYS variables determine the units for the X and Y coordinate values, in this case the value '3' sets the units to the percent of the display area of each panel. The default is '4', which would set the units to the number of graphic cells from 0 to the edge of the display. In this case, the word 'Linear' will be displayed 85 percent of the way from the bottom of the panel to the top and from the left to the right, using the CGM hardware font number one. This font will be discussed in the section on importing graphs.

```
Data linearit;
   xsys= '3';
   ysys= '3';
   color= 'black';
   font= 'HWCGM001';
   size= 1.5;
   function= 'label';
   x= 85;
   y= 85;
   text= 'Linear';
output;
run;
```

In this particular study, multiple graphs were placed in each figure and therefore a graphic template was defined that contained three panels. The first panel was used for the figure title, the second for the linear plot and the third for the log-linear plot. The two plots were created using the PKIPlot macro, which used PROC GPLOT to generate the graphs and save them as entries in a SAS catalog. The title section was created using PROC GSLIDE as shown below.

```
proc gslide name='ftitle' gout=grafout;
   title1 h= 1.25 f= HWCGM001 &_title1;
   title2 h= 1.25 f= HWCGM001 &_title2;
   title3 h= 1.25 f= HWCGM001 &_title3;
run;
```

The panels are defined using the TDEF command in PROC GREPLAY to create the template named threeply. The TPLAY command is then used to place the contents of each graph in the catalog named grafout into the appropriate panel. The NOFS option prevents the GREPLAY interactive window from being invoked.

```
proc greplay igout=grafout
   tc=tempcat
   nofs;
   tdef threeply des='redced by 5% 2 boxes w/overal title bar'
   1/llx=0 lly=6
```
TAILORING PLOTS TO THEIR DATA

A key step in automating graph production is to determine the appropriate axis scale for the data being plotted. The PK and PD plots being produced in this study had a standard X-axis because the nominal sampling times were used and they were constant across the study. The range of concentration values plotted on the Y-axis however, could vary significantly between subjects, analytes and treatment groups. There are two basic methods that can be used to determine the scale of an axis in SAS GRAPH. The first is the simplest. If you do not specify an axis scale, SAS will determine an appropriate one for you. While this is an excellent feature when producing ad-hoc graphs, there will be a lot of variability in axis scales when the range of data is highly variable. A second approach is to specify the scale in your program. This approach will be consistent across all graphs and if your range of data is consistent it may work well. In this case however, scaling the Y-axis to accommodate the maximum concentration across subjects and analytes resulted in ‘flat line’ plots when the maximum concentration was low. In order to accommodate the wide range of data while placing some constraints on the number of different scales used, it was decided to use PROC MEANS to determine the maximum value in each subset of data. The maximum was then compared with a series of ranges using a SELECT-WHEN block and a macro variable was defined that contained the text of the appropriate ORDER statement to be used with PROC GPLOT. An example of this is given in the source code below. For the sake of brevity, only some of the ranges used are included but it is easy to adapt this approach to cover any particular case. The OTHERWISE statement sets the macro variable _other equal to a null string so that the SAS default will be used if the maximum value does not fall within any of the specified ranges.

```sas
proc means data= &_data noprint;
   var &_yvar;
   output out= maxy max= maxyval;
run;

data _null_;
   set maxy;
   select;
      when( 0 < maxyval <= 2) call symput('_order', 'order=(0 to 2 by 0.5)');
      when( 2 < maxyval <= 5) call symput('_order', 'order=(0 to 5 by 1)');
      when( 5 < maxyval <= 10) call symput('_order', 'order=(0 to 10 by 2)');
      otherwise call symput('_order', '');
   end;
run;
```

Once you have a macro variable that contains the most appropriate ORDER statement, you can reference it in your AXIS statement.

```sas
axis2 label=(rotate=0 angle=90 f= HWCGM001 h=1.25 &_ytitle )
   minor=(height=0.05 in)
   &_order
   value=(f= HWCGM001 h=1.5);
```

GENERATING GRAPHIC FILES

Several methods of inserting the resulting figures into Word were tried including bitmap (BMP), encapsulated postscript (EPS) and tagged image file format (TIFF), but by far the best results were obtained by creating individual computer graphics metafiles (CGM) using hardware fonts. The output file type is specified using one of the available graphics device drivers. To generate CGM files you use CGM Office 97 device drivers CGMOF97P or CGMOF97L for portrait or landscape graphs respectively. Using hardware fonts is important for obtaining clear text that stands up to resizing. Hardware fonts provide the additional benefits of reduced file size and the ability to edit the text in the final graph in Word. Text created using software fonts is stored as a series of moves and draws rather than text. Text created using hardware fonts will be recognized as text by Word. The hardware font used was HWCGM001, which produces regular Arial text in Word. Font two is Arial bold, three is italic and four is bold italic. The same pattern repeats with fonts five through eight for Times New Roman and with fonts nine
through twelve for Courier New. A complete listing of the fonts available and their designation is provided in the SAS technical
support document TS252X-2, “EXPORTING SAS/GRAPH Output to Microsoft Word, Excel and PowerPoint 97”. This
document is available from the SAS web site and is a valuable reference for those who plan on doing any significant amount
of exporting SAS graphics to Microsoft Office. Another valuable paper is TS674 “An Introduction to Exporting SAS/Graph
Output to Microsoft Office”. Another important fact to keep in mind regarding text is that the default unit for text height in SAS
GRAPH is the hardware character cell and when a graph is imported into Word, a point size must be used. The following
mapping of text size is used during the import process.

<table>
<thead>
<tr>
<th>SAS GRAPH H=</th>
<th>Word Point Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>4</td>
</tr>
<tr>
<td>1.0</td>
<td>7</td>
</tr>
<tr>
<td>1.5</td>
<td>10</td>
</tr>
<tr>
<td>2.0</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>17</td>
</tr>
<tr>
<td>3.0</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>24</td>
</tr>
</tbody>
</table>

The example below shows how the device driver and font are specified using the DEV and FTEXT graphics options.

goptions gsfname = goutcgm
  rotate = portrait
  dev = cgmof97p
  ftext = HWCGM001
  gsfmode = replace
  gaccess = gsasfile
  gsflen = 8092
  display
  ;

GENERATING VISUAL BASIC SOURCE CODE
A simple data _null_ step can be used to generate Visual Basic source code that can be run as a macro in Word. A series of
PUT statements are used to add lines to a text file. Each PUT statement has a literal string which contains the appropriate
Visual Basic code and macro variable references are used for specific code that varies from figure to figure. The Visual Basic
commands were determined by recording a macro while inserting a figure and then resizing it to fit the entire page. The macro
was then opened in the Visual Basic editor in Word and copied into the SAS macro to create the %PUT statements. The
Word macro has two basic functions, insert each figure in proper sequence and then resize them to fit the page. The first four
lines of code specify that we want to add a picture to the current file, the name of the file to insert, that we do not want to link
the picture to its source file and that we want to save the picture with the document. A group of macro variables are joined
together to provide the full path and filename. Once the figure has been inserted, Word will assign it a sequential inline shape
ID number. In Visual Basic, a WITH block can be used to specify more than one property for a given object without repeating
the object specification each time. In this macro, the inline shape just created is specified and the properties of height and
width are set equal to the values that resulted from your initial manual resizing. A global macro variable FN is used to contain
the current inline shape number. Its value is incremented after each figure insert. Because macro variables are always
character, the %EVAL macro function is used to perform the addition. Prior to generating any figures a macro called VBINIT
is called with a parameter that specifies the path and name of the Visual Basic source code file to create. This macro also
creates the global macro variable FN and sets it equal to one.

%macro vbinit(_prgfile= );
  filename vbout "&_prgfile";
  %global fn;
  %let fn = 1;
  %mend vbinit;

%vbinit (_prgfile= %str(I:\GDAY411\out\Figures\figures.vba));

An example of the SAS source code used to generate the Visual Basic codes is given below. Note that in Visual Basic, a
semicolon is not used to end a statement. Each statement is assumed to end at the end if the line unless the last character in
the line is an underscore.

data _null_
  file vbout mod;

The example below shows how the device driver and font are specified using the DEV and FTEXT graphics options.
To create a new macro in Word, select TOOLS, MACRO and MACROS from the menu system and a screen will appear that will allow you to enter the name you wish to give the macro. After you have entered the name, the Visual Basic editor will appear with the SUB statement specifying the macro name and an END SUB statement at the bottom. You can then place the cursor between the two statements and insert the source code that was generated by SAS. Save your new macro and you are ready to run it. It is important to note that the default location for new Word macros is the NORMAL.DOT template. This has the advantage of making your macros available when you create new documents but it can also clutter the template with many macros that are seldom used and macros with the same name will be overwritten. To save your macro within a particular document, use the project explorer in the Visual Basic Editor to select your document before inserting and saving your macro. An example of the Visual Basic source code that is generated is given below.

Sub GDAY002_PK_PD

Selection.InlineShapes.AddPicture _
FileName:="I:\GDAY\PKPlots\pkfig1.cgm", _
    LinkToFile:=False, _
    SaveWithDocument:=True
With ActiveDocument.InlineShapes(1)
    .Height = 650
    .Width = 500
End With

Selection.InlineShapes.AddPicture _
FileName:="I:\GDAY\PKPlots\pkfig2.cgm", _
    LinkToFile:=False, _
    SaveWithDocument:=True
With ActiveDocument.InlineShapes(2)
    .Height = 650
    .Width = 500
End With

End Sub

In the above method, a separate block of code is generated for each plot that needs to be inserted. While this is not the most efficient way to accomplish this task in Visual Basic, it does facilitate a simple, straightforward method of generating the Visual Basic source code as each figure is being produced by SAS/GRAPH. A major drawback of this method is that it produces very long Word macros when the number of figures is large, and Word has a maximum limit on the size of a macro. Because the Word macro runs the same basic statements to insert and resize each figure, the size of the macro can be greatly reduced through the use of FOR-NEXT loops. Fortunately in this study, the scheme of numbering the figures was set up in a logical manner. The figure number was divided into parts that referred to the type of data (PK or PD), the subject, the analyte and the study day. The analyte loop needed to vary its range because there were three PK analytes and 15 PD analytes. This was handled by using an immediate IF statement (IIF) to set the upper range for this loop. If PKPD was equal to two (PK) then N was set to two, otherwise it was set to fourteen. The CStr function is used to convert values from numeric to character.

Sub GDAY002_PK_PD_LOOP

    ilsnum = 0
    For subj = 1 To 22
        For pkpd = 2 To 3
            n = IIf(pkpd = 2, 3, 15)
            For analyte = 1 To n
                For sday = 1 To 2

In the above method, a separate block of code is generated for each plot that needs to be inserted. While this is not the most efficient way to accomplish this task in Visual Basic, it does facilitate a simple, straightforward method of generating the Visual Basic source code as each figure is being produced by SAS/GRAPH. A major drawback of this method is that it produces very long Word macros when the number of figures is large, and Word has a maximum limit on the size of a macro. Because the Word macro runs the same basic statements to insert and resize each figure, the size of the macro can be greatly reduced through the use of FOR-NEXT loops. Fortunately in this study, the scheme of numbering the figures was set up in a logical manner. The figure number was divided into parts that referred to the type of data (PK or PD), the subject, the analyte and the study day. The analyte loop needed to vary its range because there were three PK analytes and 15 PD analytes. This was handled by using an immediate IF statement (IIF) to set the upper range for this loop. If PKPD was equal to two (PK) then N was set to two, otherwise it was set to fourteen. The CStr function is used to convert values from numeric to character.
CONCLUSION
The examples in this paper illustrate some of the key concepts that can be used to automate the production of SAS graphs in Microsoft Word. These methods not only increase the efficiency in producing large numbers of graphs but they also help eliminate errors and ensure consistency. The macro presented in this paper was developed for a study in which over 3,000 figures were produced and it has since been modified for use on several other studies with large numbers of graphs.

REFERENCES


CONTACT INFORMATION (HEADER 1)
Robert Graebner
Quintiles, Inc.
P.O. Box 9708
Kansas City, MO 64134-0708

Email: bob.graebner@quintiles.com
mgraebner@kc.rr.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.