Making ODS Graphics Templates Generic

Bob Newman, Amadeus Software Limited

ABSTRACT

The usefulness of an ODS graphics template can be greatly increased by making it more flexible in its behaviour. This paper shows how a graphics template can be parameterised, how the parameters can be made optional and given default values, and how the high-level behaviour of the template can be made to depend on the parameter values specified. Along the way, some insights are given into the workings of the Graphics Template Language (GTL).

It is assumed that the reader has some knowledge of the ODS Statistical Graphics procedures, and will not take fright at the sight of GTL code. Knowledge of GTL syntax is not a pre-requisite, however.

INTRODUCTION

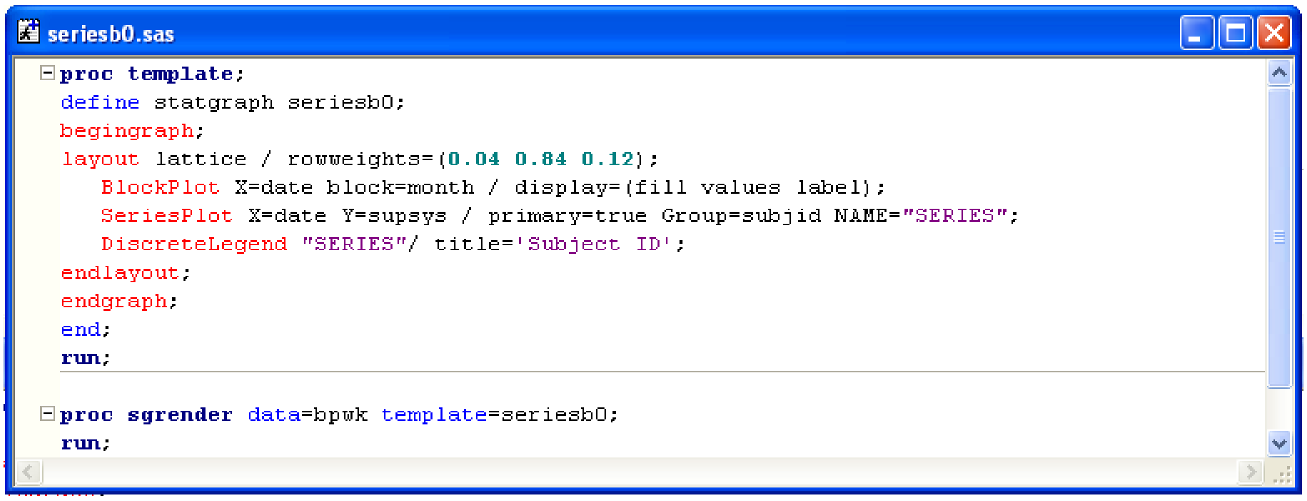
Procedures such as SGPLOT, SGPANEL and SGSCATTER (which were introduced in SAS 9.2) enable high-quality graphics to be generated with relatively little effort, and all support a TMPLOUT option that enables a graphics template definition to be written to a file, for re-use using Proc SGRENDER. A graphics template specifies a plot, in a way that is independent of the data set being plotted.
This paper shows how variability can be introduced into such a graphics template, thereby greatly increasing its usefulness to the user community.

The techniques demonstrated begin with the use of dynamic variables and macro variables to specify parameters, then move on to the possibilities offered by use of expression evaluation (including the use of some functions specific to GTL) and conditional logic. These possibilities include making parameters optional, and/or giving them default values. Finally an example is given of storing and using a permanent template.

No attempt is made here to teach GTL syntax, but the GTL code used is straightforward and not hard to understand. By the end of the paper, the reader who has not used GTL before may feel emboldened to delve into the documentation and try writing some of his/her own GTL code.

A BASIC TEMPLATE

We begin with an ODS graphics template in which everything is explicitly specified, and use it to generate a graph.



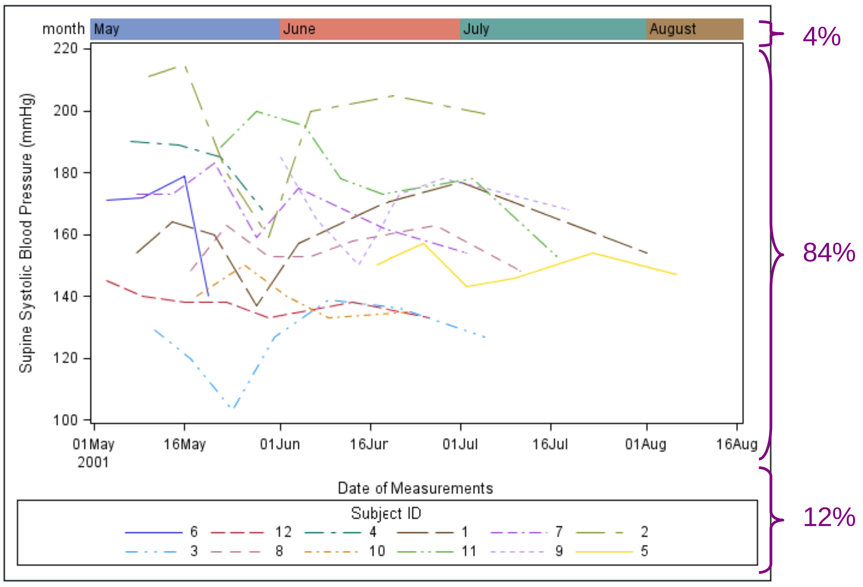
```
proc template;
define statgraph seriesb0;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   BlockPlot X=date block=month / display=(fill values label);
   SeriesPlot X=date Y=supsys / primary=true Group=subjid NAME="SERIES";
   DiscreteLegend "SERIES"/ title='Subject ID';
endlayout;
endgraph;
end;
run;

proc sgrender data=bpwk template=seriesb0;
run;
```

Normally a good starting point for this kind of exercise is a template that has been created using one of the SG procedures, such as Proc SGPLOT, using the TMPLOUT option. However, the above template, which is called SERIESB0, uses a little more of the Graphics Template Language (GTL). It combines a grouped series plot with a block plot and a legend; block plots are one of the GTL features not available using Proc SGPLOT. In the example below, the block plot is the coloured band across the top.

The LAYOUT statement used here divides the plot area vertically into a LATTICE of three cells, allocating 4% of the available space to the block plot, 84% to the series plot, and 12% to the legend. (Other forms of LAYOUT are available, one of the most common being LAYOUT OVERLAY, which enables plots to be superimposed.)

The series plot is named purely so that the DISCRETELEGEND statement can refer to it; the name “SERIES” does not appear anywhere in the output. (It might be better style to have the legend statement inside a SIDEBAR block, but the code used here is shorter and good enough for illustrative purposes.)

The graph is then created using Proc SGRENDER, whose two inputs are the name of a data set containing data to be plotted, and the name of the template. The result looks like this:

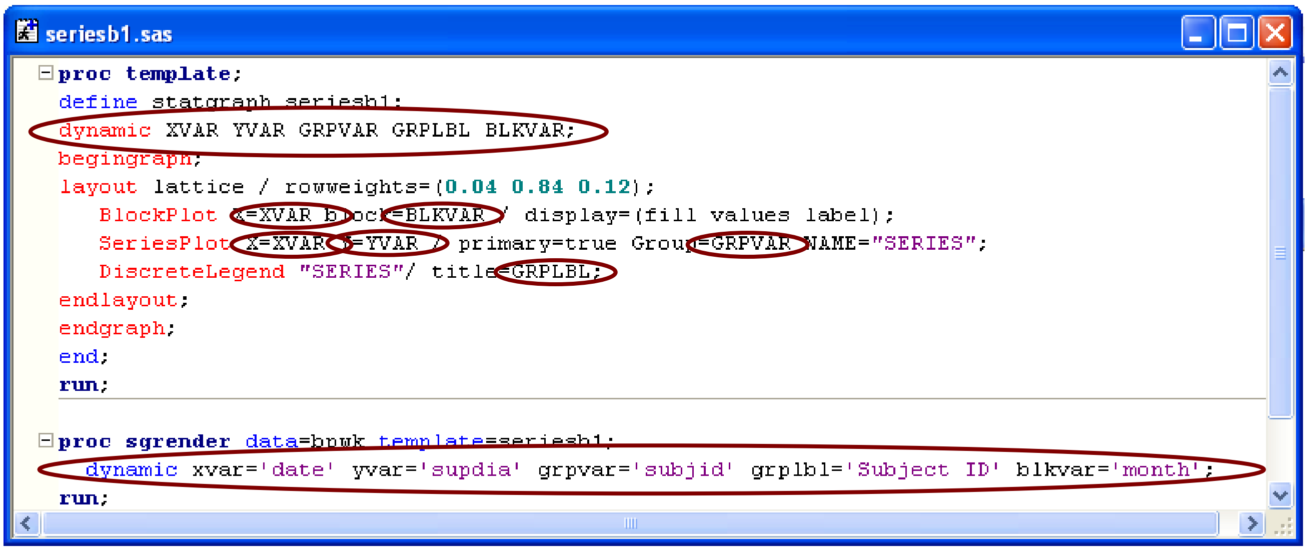


The block plot is the coloured band containing the month names. Notice that although this template can, in principle, be used with *any* data set, it will fail unless the data set contains numeric variables called DATE, SUPSYS and SUBJID, and a character variable called MONTH. The block plot further requires that the data set be sorted by DATE.

The Subject ID values are not well-ordered in the legend. This point will be considered in a later section on “sorting the data”.

USING DYNAMIC VARIABLES

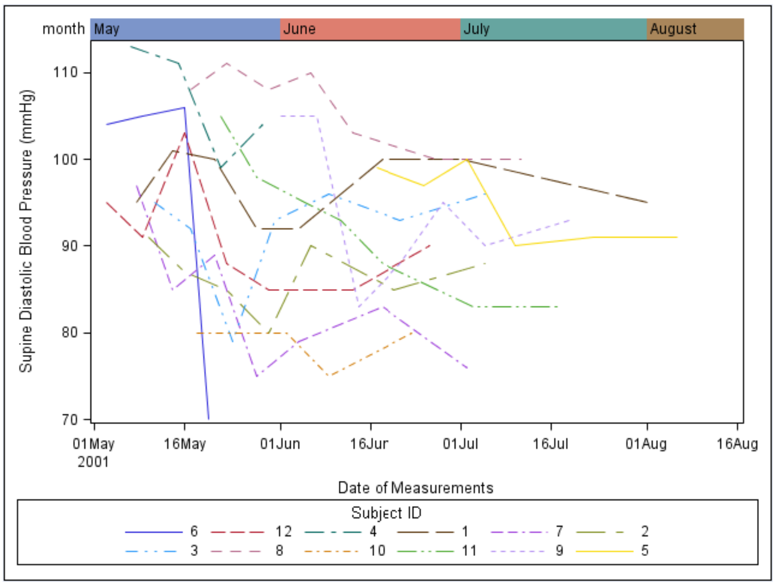
In making this template more flexible, the first thing we would like to do is to make parameters of a few things that are currently hard-coded, such as the variable names. This can be done using *dynamic variables*:



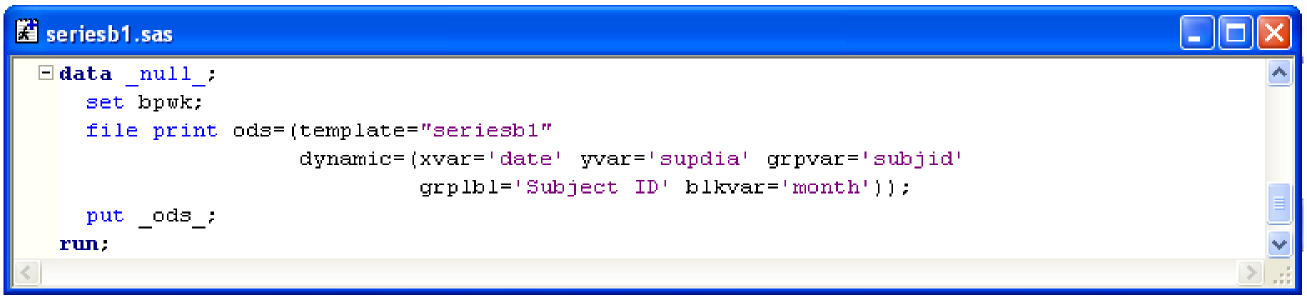
```
proc template;
define statgraph seriesb1;
dynamic XVAR YVAR GRPVAR GRPLBL BLKVAR;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   BlockPlot X=XVAR block=BLKVAR / display=(fill values label);
   SeriesPlot X=XVAR Y=YVAR / primary=true Group=GRPVAR NAME="SERIES";
   DiscreteLegend "SERIES"/ title=GRPLBL;
endlayout;
endgraph;
end;
run;

proc sgrender data=bpwk template=seriesb1;
  dynamic xvar='date' yvar='supdia' grpvar='subjid' grplbl='Subject ID' blkvar='month';
run;
```

This new template SERIESB1 uses a DYNAMIC statement to declare five dynamic variables, for the four variable names and the label for the group variable. Upper case has been used for all these names; this is not essential, but it is a useful convention. In the remainder of the template, the names of these dynamic variables appear where previously there were fixed values. Notice that there is no special syntax associated with these substitutions, and in the DISCRETELEGEND statement, GRPLBL does not even need any quotes around it.

Now when a plot is generated, values have to be specified for these dynamic variables. This is done using Proc SGRENDER’s own DYNAMIC statement. All the values assigned here have to be quoted, whether they are variable names, fixed character strings, or fixed numeric values. In this example, we have chosen to use the same data set, but to plot a different Y-variable, SUPDIA rather than SUPSYS.



The same graph could have been produced, incidentally, using a special form of data step instead of Proc SGRENDER. The equivalent code would be:



```
data _null_;
  set bpwk;
  file print ods=(template="seriesb1"
                 dynamic=(xvar='date' yvar='supdia' grpvar='subjid'
                          grplbl='Subject ID' blkvar='month'));
  put _ods_;
run;
```

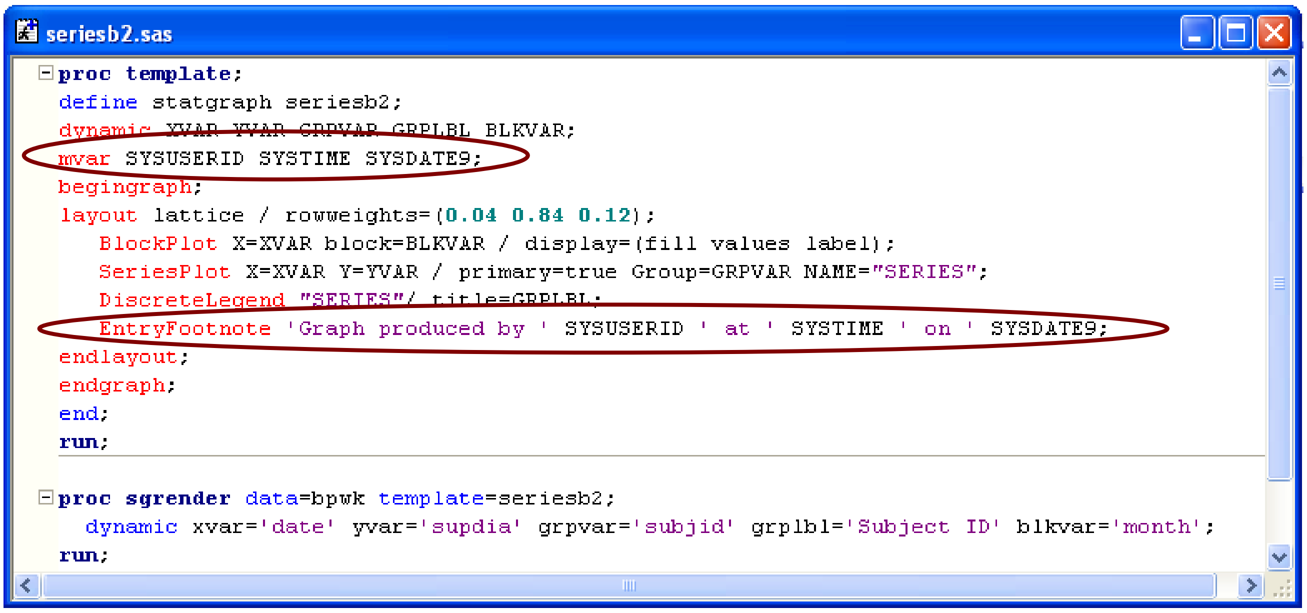
Proc SGRENDER will continue to be used for the remainder of this paper, but the possibility of using this kind of data step instead is always available.

USING MACRO VARIABLES WITHIN TEMPLATE DEFINITIONS

Within template definitions, macro variables behave a lot more like dynamic variables than like standard SAS macro variables. They are:

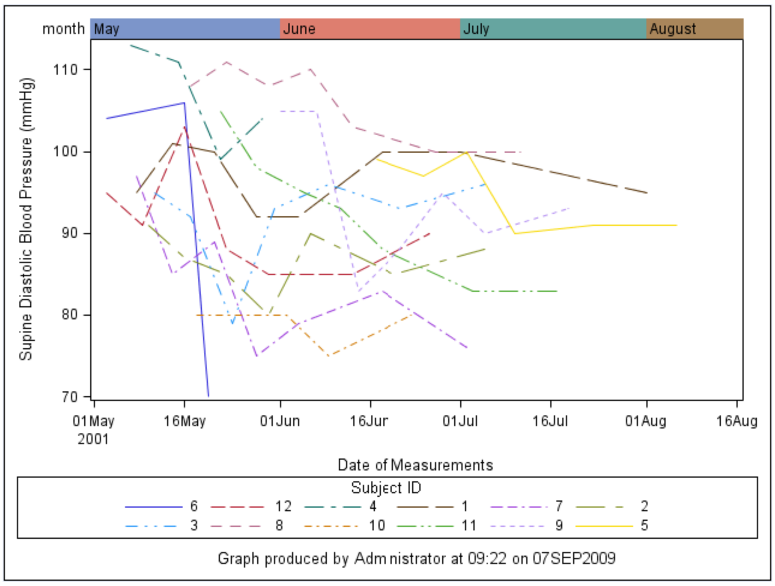
- Declared using the MVAR and NMVAR statements. MVAR is used to declare character macro variables, and NMVAR to declare numeric macro variables.
- Substituted in simply by naming them (as with dynamic variables) NB the "&" symbol is ***not*** used.

It would be feasible to use macro variables in the template definition in all the places where dynamic variables were used in the previous example. There would be no advantage in doing that, and as a general rule when defining parameters it is preferable to use dynamic variables. Dynamic variables require a special syntax when values are assigned to them, so it is always clear that they relate to a graphics template. In contrast, there are several ways to set the values of a macro variable, and many places other than graphics templates where they can be used. However, the macro facility can be particularly useful when using *automatic* macro variables such as SYSDATE, for example:



```
proc template;
define statgraph seriesb2;
dynamic XVAR YVAR GRPVAR GRPLBL BLKVAR;
mvar SYSUSERID SYSTIME SYSDATE9;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   BlockPlot X=XVAR block=BLKVAR / display=(fill values label);
   SeriesPlot X=XVAR Y=YVAR / primary=true Group=GRPVAR NAME="SERIES";
   DiscreteLegend "SERIES"/ title=GRPLBL;
   EntryFootnote 'Graph produced by ' SYSUSERID ' at ' SYSTIME ' on ' SYSDATE9;
endlayout;
endgraph;
end;
run;

proc sgrender data=bpwk template=seriesb2;
  dynamic xvar='date' yvar='supdia' grpvar='subjid' grplbl='Subject ID' blkvar='month';
run;
```

GTL's ENTRYFOOTNOTE statement is used here to specify a footnote giving the username, time and date. The three macro variables used, all of which are automatic macro variables, are declared in an MVAR statement. The result is:



NOTE ON TITLES AND FOOTNOTES IN GTL

Titles and footnotes in graphs produced by Proc SGRENDER are specified using the ENTRYTITLE and ENTRYFOOTNOTE statements within the template definition. ENTRYTITLE and ENTRYFOOTNOTE statements are not numbered; their precedence is determined by the order in which they appear in the GTL code. *The standard SAS statements TITLE and FOOTNOTE are ignored by Proc SGRENDER.*

When an SG procedure such as SGPLOT or SGPANEL runs, it creates a graphics template "on the fly", and this will include ENTRYTITLE and ENTRYFOOTNOTE statements equivalent to the TITLE and FOOTNOTE statements in force at the time. If the TMPLOUT option is used to write the template definition to a file, these ENTRYTITLE and ENTRYFOOTNOTE statements will be included, and they will not be altered if a graph is subsequently produced from the template using Proc SGRENDER.

A possible approach is to use dynamic variables to generate titles and footnotes. An example of this is given later.

EVALUATING EXPRESSIONS

Within GTL, expressions can be evaluated using the EVAL function, but this facility is not available at all points in the template definition. The detailed GTL syntax (in the SAS reference documentation) states exactly where expressions are supported and where they are not. In general, expressions are supported for:

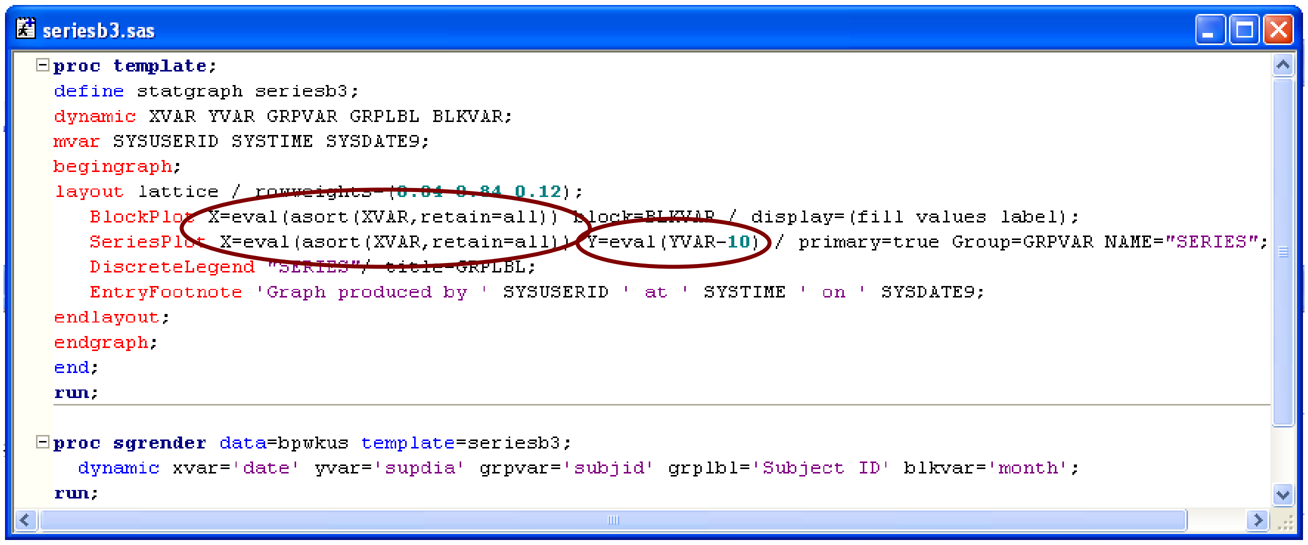
- Options (not suboptions) where the value normally required is a (single) constant, or the name of a variable

- Text elements within ENTRYTITLE, ENTRYFOOTNOTE and ENTRY statements

Within an expression you can use data step functions, arithmetic operators, and other special functions supported by the GTL.

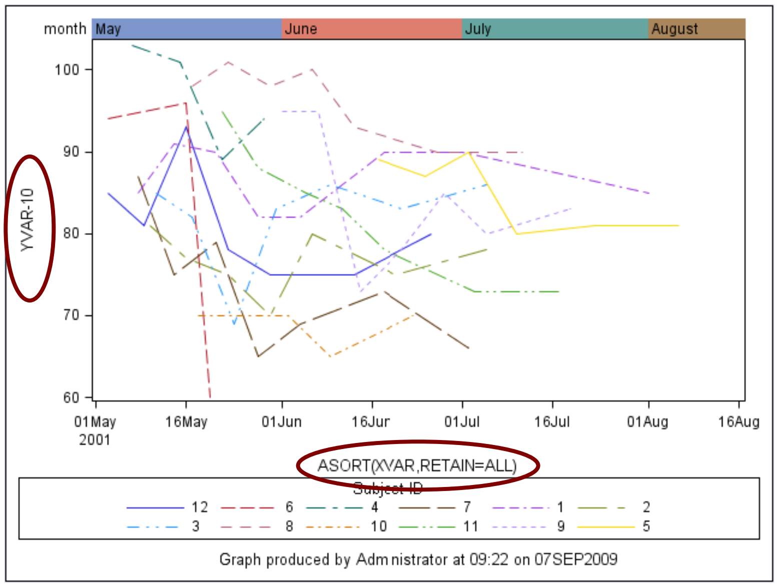
This example uses expressions on the X and Y specifications in the BLOCKPLOT and SERIESPLOT statements. For Y, we are simply subtracting 10 from the recorded value (perhaps the instruments were consistently misreading). Most use of the EVAL function will be as simple as this.

The treatment of the X variable is more elaborate and interesting. It uses ASORT, which is one of GTL's own functions. This sorts the XVAR values into ascending order. (There is also a DSORT for descending order). Specifying the template in this way means that it can handle a dataset which is unsorted. The option "RETAIN=ALL" is essential here – without it, *only* the XVAR values would be sorted, and correspondence with other columns would be lost.


seriesb3.sas

```
proc template;
define statgraph seriesb3;
dynamic XVAR YVAR GRPVAR GRPLBL BLKVAR;
mvar SYSUSERID SYSTIME SYSDATE9;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   BlockPlot X=eval(asort(XVAR,retain=all)) block=BLKVAR / display=(fill values label);
   SeriesPlot X=eval(asort(XVAR,retain=all)) Y=eval(YVAR-10) / primary=true Group=GRPVAR NAME="SERIES";
   DiscreteLegend "SERIES"/ title=GRPLBL;
   EntryFootnote 'Graph produced by ' SYSUSERID ' at ' SYSTIME ' on ' SYSDATE9;
endlayout;
endgraph;
end;
run;

proc sgrender data=bpwkus template=seriesb3;
  dynamic xvar='date' yvar='supdia' grpvar='subjid' grplbl='Subject ID' blkvar='month';
run;
```

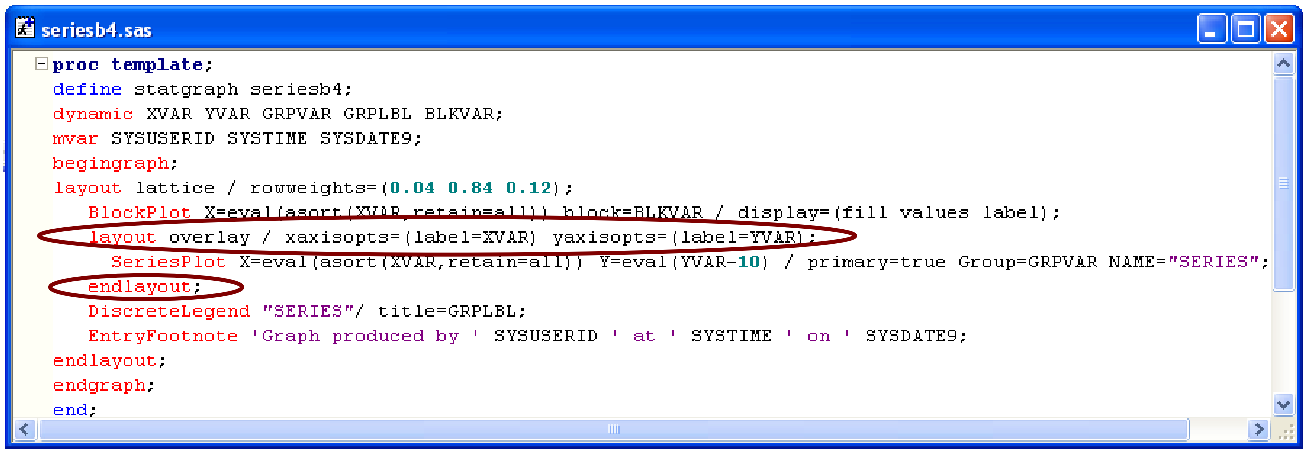
Using this new template, we are able to generate plots from the data set BPWKUS, in which the data is unsorted.



Notice that the axis labels have changed. Previously they contained the labels of the variables being plotted. Now instead they contain the formulae for the expressions being plotted. Putting this right is quite difficult. The moral is that using GTL's sorting functions can bring problems, and it may be better to stipulate that the template requires the data set to have already been sorted. However, for the moment we will pursue those problems, and see what can be done.

FIXING THE AXIS LABELS

It is really only the axes for the series plot that need to be altered. Trying to specify axis properties at the level of the LAYOUT LATTICE will have undesirable effects on the block plot and the legend. However, there are no axis-related options to the SERIESPLOT statement. A little trick can solve this problem: we wrap the SERIESPLOT up inside its own LAYOUT OVERLAY block, and use the axis options of the LAYOUT OVERLAY statement. An overlay layout is designed to enable multiple plots to be superimposed, but it also works if used with a single plot. Notice that we now have nested layouts. This is fine, and in fact quite common.



```
proc template;
define statgraph seriesb4;
dynamic XVAR YVAR GRPVAR GRPLBL BLKVAR;
mvar SYSUSERID SYSTIME SYSDATE9;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   BlockPlot X=eval(asort(XVAR,retain=all)) block=BLKVAR / display=(fill values label);
   layout overlay / xaxisopts=(label=XVAR) yaxisopts=(label=YVAR);
     SeriesPlot X=eval(asort(XVAR,retain=all)) Y=eval(YVAR-10) / primary=true Group=GRPVAR NAME="SERIES";
   endlayout;
   DiscreteLegend "SERIES"/ title=GRPLBL;
   EntryFootnote 'Graph produced by ' SYSUSERID ' at ' SYSTIME ' on ' SYSDATE9;
endlayout;
endgraph;
end;
```

Even now, the axes are labelled with the *names* of the variables being plotted. We would have preferred to use the *labels* of those variables, and might have expected to be able to obtain those using GTL's COLLABEL function, but unfortunately GTL does not support expressions at this point. (We are specifying a value here for the LABEL *suboption* within the XAXISOPTS option; in general, expressions are not supported for suboption values.) This deficiency could be remedied by defining new dynamic variables to specify the axis labels.

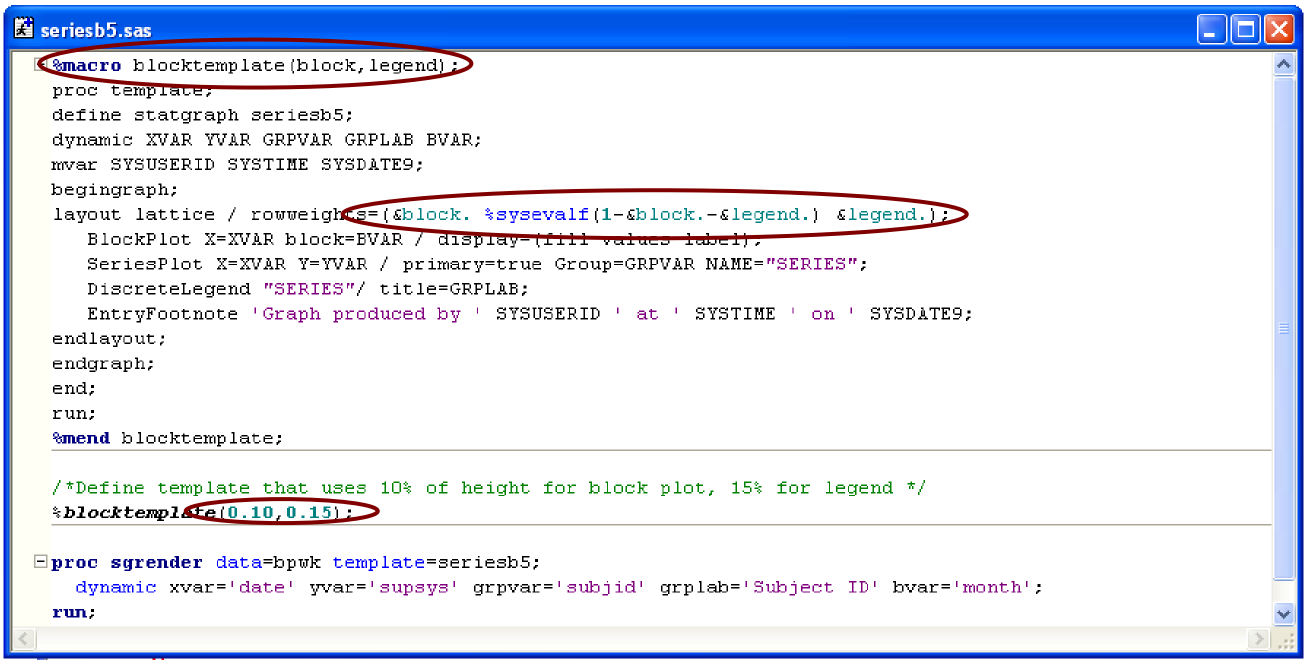
SORTING THE DATA

A fundamental principle of GTL is that, however complex a graphics template may be, it can only use data from one data set. A less obvious principle is that, during the generation of the graph, it can only use the data *in one order*. Functions such as ASORT and DSORT can be used to make the template sort the data for itself, but once it is sorted, that is the order in which the data will be used for all components of the graph. If different statements within the template attempt to sort the data in different ways, only one of them will get its own way.

The example demonstrated here is problematical because the block plot and the series plot have different requirements for the ordering of the data. The block plot requires it all to be sorted by date alone. The series plot would produce a better-ordered legend if the data were sorted by date within subject ID, but this would prevent a block plot from being produced. Resolving this incompatibility would require a degree of cunning in the construction of the input data set, and further modifications to the template; this will not be demonstrated here.

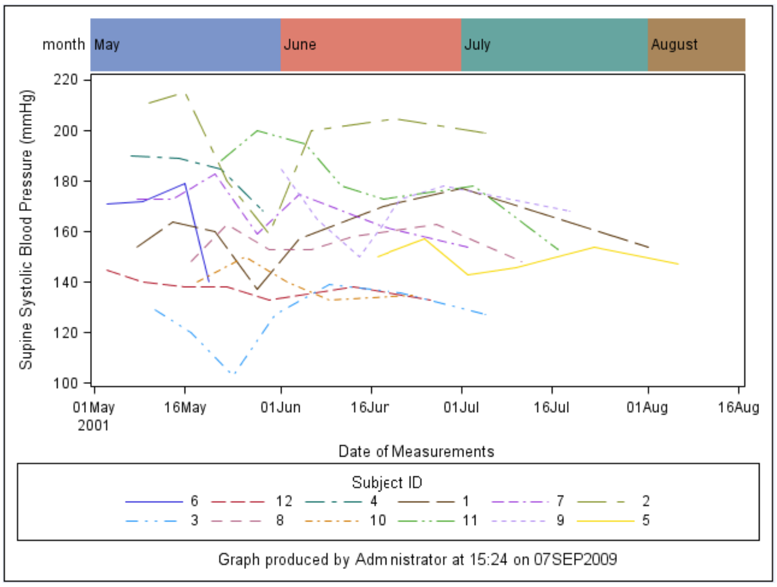
USING MACRO VARIABLES IN DEFINING THE GRAPHICS TEMPLATE

In our example, parameterising the width of the block plot (hitherto 0.04) and the legend plot (0.12) is tricky, because they occur in places within the template definition where GTL's EVAL function is not available. This could however be done using macro variables in the standard, non-GTL, way, when generating the Proc TEMPLATE code, for example:



```
%macro blocktemplate(block,legend);
proc template;
define statgraph seriesb5;
dynamic XVAR YVAR GRPVAR GRPLAB BVAR;
mvar SYSUSERID SYSTIME SYSDATE9;
begingraph;
layout lattice / rowweights=(&block. %sysevalf(1-&block.-&legend.) &legend.);
   BlockPlot X=XVAR block=BVAR / display=(fill values label);
   SeriesPlot X=XVAR Y=YVAR / primary=true Group=GRPVAR NAME="SERIES";
   DiscreteLegend "SERIES"/ title=GRPLAB;
   EntryFootnote 'Graph produced by ' SYSUSERID ' at ' SYSTIME ' on ' SYSDATE9;
endlayout;
endgraph;
end;
run;
%mend blocktemplate;

/*Define template that uses 10% of height for block plot, 15% for legend */
%blocktemplate(0.10,0.15);

proc sgrender data=bpwk template=seriesb5;
  dynamic xvar='date' yvar='supsys' grpvar='subjid' grplab='Subject ID' bvar='month';
run;
```

Here the template itself is generated dynamically using a macro. Both substitution and expression evaluation are performed using the SAS macro language, at the time the Proc TEMPLATE code is generated. Later, when the compiled template is used to generate a plot, substitution is performed using dynamic variables (and the automatic macro variables, as before) within GTL. The result is a plot with the proportions modified as specified:



CONDITIONAL LOGIC

GTL supports conditional logic, in the form of IF/THEN/ELSE/ENDIF blocks. These can be nested. The syntax is straightforward, though different from that of Base SAS.

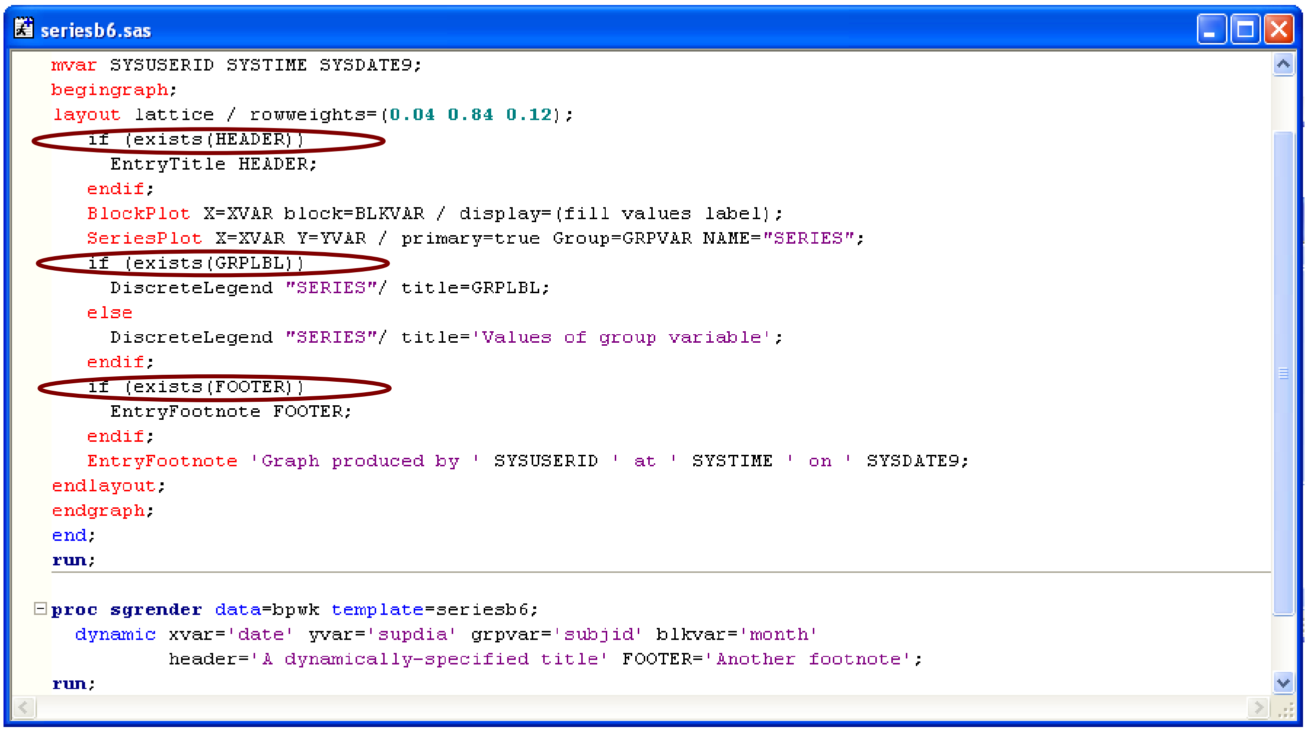
The conditions can involve arithmetic operators, logical operators, comparison operators, Boolean operators, concatenation operators, data step functions and/or GTL's own functions.

The code that is conditionally generated should be one or more complete GTL statements. Conditional generation of partial lines does not work.

This facility allows a great deal of flexibility to be introduced into graphics templates, although sometimes at the cost of repetitive coding. Some examples will now be given,

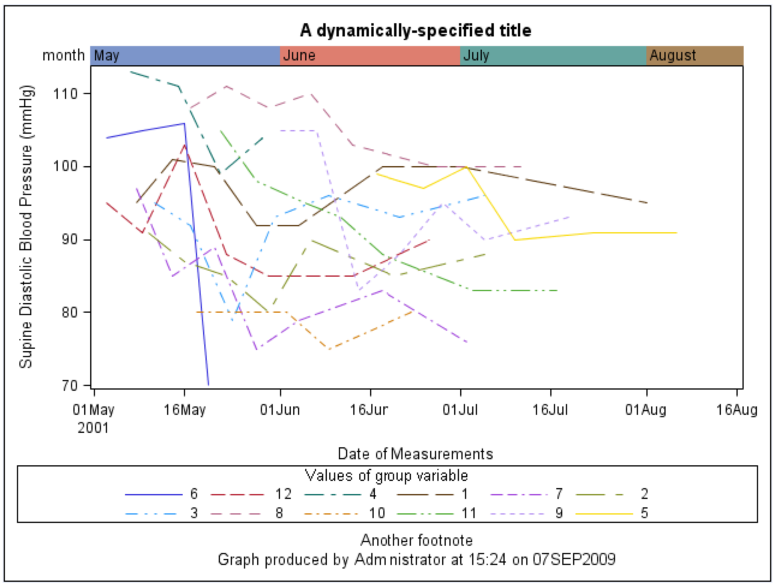
OPTIONAL PARAMETERS AND DEFAULT VALUES

In this example, we make the GRPLBL parameter optional, with a default value of "Values of group variable", and introduce new optional parameters HEADER and FOOTER which add a user-supplied title and a user-supplied additional footnote.



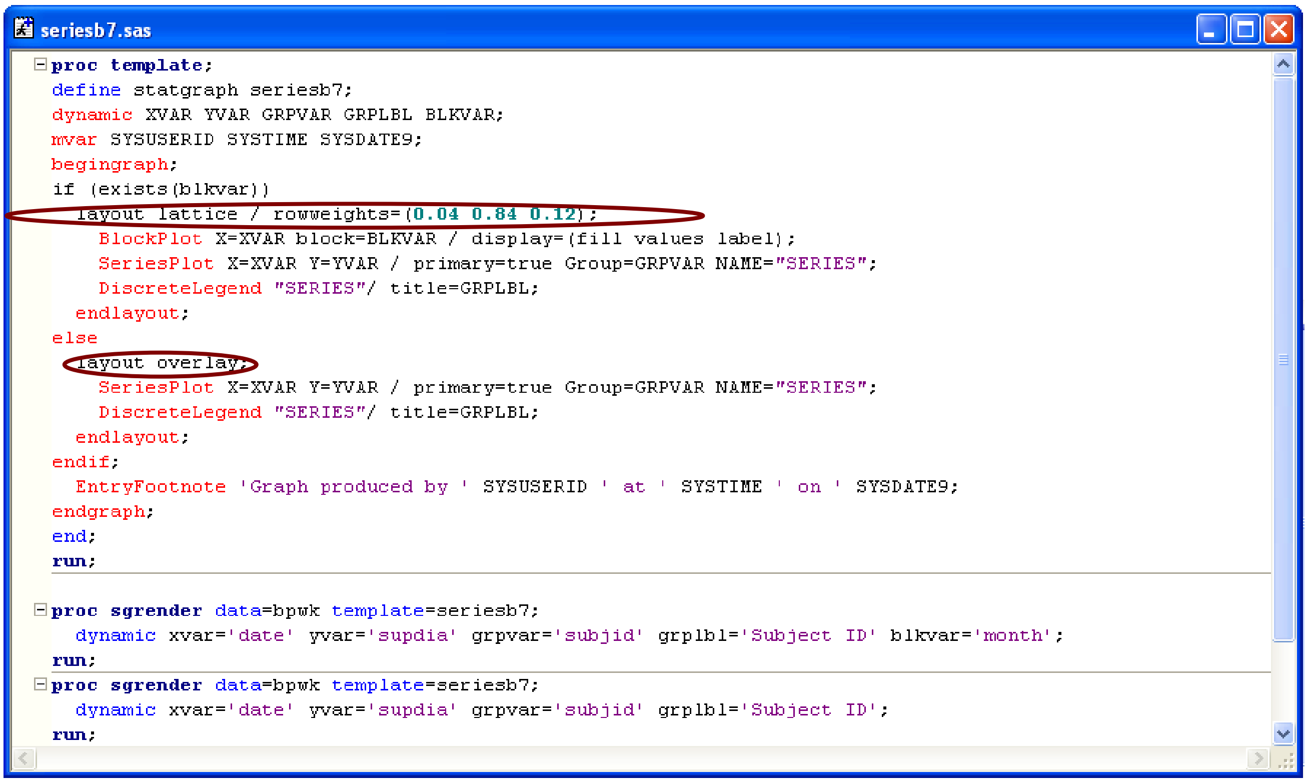
```
mvar SYSUSERID SYSTIME SYSDATE9;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   if (exists(HEADER))
     EntryTitle HEADER;
   endif;
   BlockPlot X=XVAR block=BLKVAR / display=(fill values label);
   SeriesPlot X=XVAR Y=YVAR / primary=true Group=GRPVAR NAME="SERIES";
   if (exists(GRPLBL))
     DiscreteLegend "SERIES"/ title=GRPLBL;
   else
     DiscreteLegend "SERIES"/ title='Values of group variable';
   endif;
   if (exists(FOOTER))
     EntryFootnote FOOTER;
   endif;
   EntryFootnote 'Graph produced by ' SYSUSERID ' at ' SYSTIME ' on ' SYSDATE9;
endlayout;
endgraph;
end;
run;

proc sgrender data=bpwk template=seriesb6;
  dynamic xvar='date' yvar='supdia' grpvar='subjid' blkvar='month'
          header='A dynamically-specified title' FOOTER='Another footnote';
run;
```

This example makes use of the GTL function EXISTS, which tests whether an item exists. The item can be (as here) a dynamic variable; or it can be a macro variable, or a column in the input data set. NB the GTL function EXISTS is different from the EXIST function of base SAS.



HIGHER-LEVEL VARIATION IN TEMPLATES

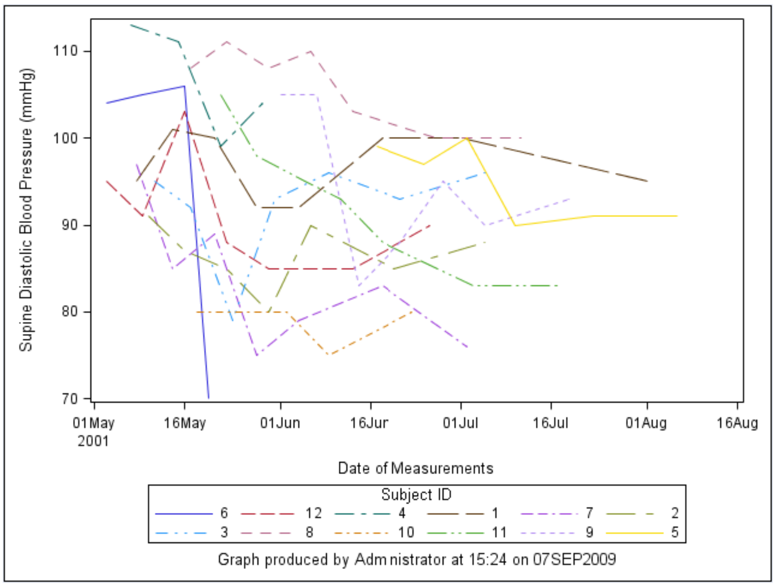
It may sometimes be desirable for a template to produce quite different behaviour dependent on the values of the parameters. For example, the same template could produce either a quadratic regression plot or a loess plot, depending on the circumstances. Remember though that a template can only use data from a single data set, which limits the scope for variation in behaviour to some extent.
In the example below, the BLKVAR parameter is made optional. If it is not supplied, then a simple grouped series plot is produced, with no block plot.



```
proc template;
define statgraph seriesb7;
dynamic XVAR YVAR GRPVAR GRPLBL BLKVAR;
mvar SYSUSERID SYSTIME SYSDATE9;
begingraph;
if (exists(blkvar))
  layout lattice / rowweights=(0.04 0.84 0.12);
    BlockPlot X=XVAR block=BLKVAR / display=(fill values label);
    SeriesPlot X=XVAR Y=YVAR / primary=true Group=GRPVAR NAME="SERIES";
    DiscreteLegend "SERIES"/ title=GRPLBL;
  endlayout;
else
  layout overlay;
    SeriesPlot X=XVAR Y=YVAR / primary=true Group=GRPVAR NAME="SERIES";
    DiscreteLegend "SERIES"/ title=GRPLBL;
  endlayout;
endif;
  EntryFootnote 'Graph produced by ' SYSUSERID ' at ' SYSTIME ' on ' SYSDATE9;
endgraph;
end;
run;

proc sgrender data=bpwk template=seriesb7;
  dynamic xvar='date' yvar='supdia' grpvar='subjid' grplbl='Subject ID' blkvar='month';
run;
proc sgrender data=bpwk template=seriesb7;
  dynamic xvar='date' yvar='supdia' grpvar='subjid' grplbl='Subject ID';
run;
```

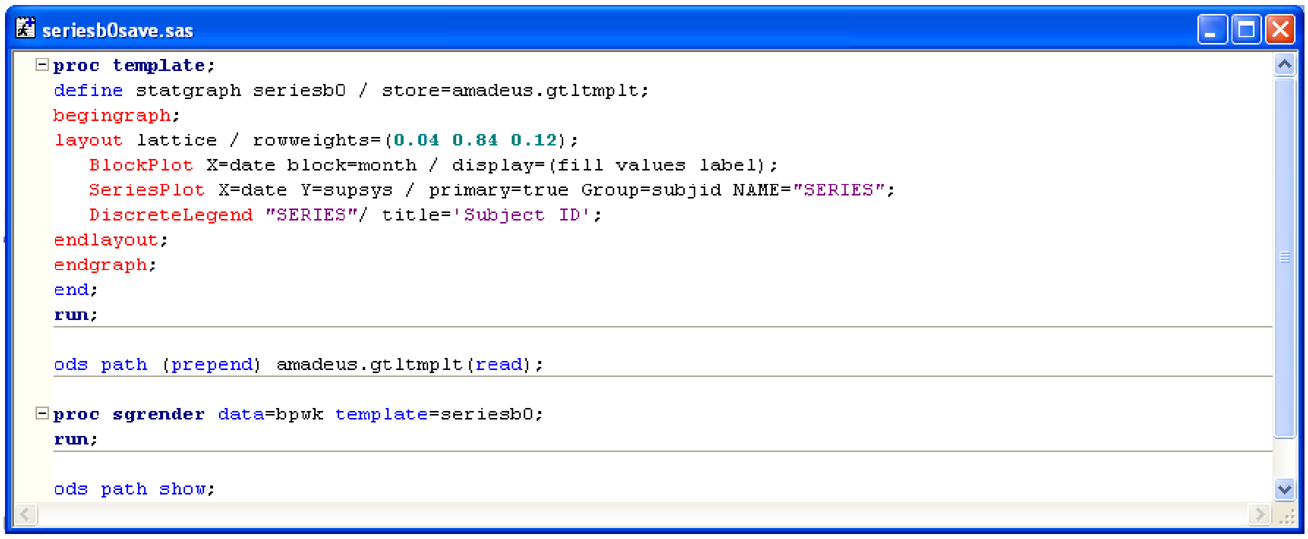
Notice that each branch of the condition here generates a complete LAYOUT block. When no block plot is being generated, an overlay layout is appropriate, rather than the lattice layout we have been using up till now. Conditionally-generated LAYOUT statements with an unconditionally-generated ENDLAYOUT would confuse GTL. Notice also that the ENTRYFOOTNOTE statement has been moved outside the LAYOUT blocks – though it remains within the GRAPH block.

The second call to Proc SGRENDER uses the overlay layout and produces the following plot:



STORING AND USING PERMANENT TEMPLATES

In all the above examples, the templates have been stored in the default item store SASUSER.TEMPLAT. It is possible however to store and use permanent templates that are available to other users. The following code demonstrates this, using the simple form of the template with which we began this paper.



```
proc template;
define statgraph seriesb0 / store=amadeus.gtltmplt;
begingraph;
layout lattice / rowweights=(0.04 0.84 0.12);
   BlockPlot X=date block=month / display=(fill values label);
   SeriesPlot X=date Y=supsys / primary=true Group=subjid NAME="SERIES";
   DiscreteLegend "SERIES"/ title='Subject ID';
endlayout;
endgraph;
end;
run;

ods path (prepend) amadeus.gtltmplt(read);

proc sgrender data=bpwk template=seriesb0;
run;

ods path show;
```

The STORE option to the DEFINE STATGRAPH statement specifies that this template is to be saved to an item store called GTLTMPLT in the AMADEUS library. The first ODS statement adds this item store to the ODS search path, so that the template will be found by Proc SGRENDER. The final ODS statement displays the ODS search path in the log.

ODS GRAPHICS DESIGNER AND PROC SGDESIGN

The ODS Graphics Designer is an alternative method of producing ODS statistical graphics plots, and a feature known as “shared variable (SV) plots” has been specified which will allow plots produced in this way to be parameterised, when replaying saved plot definitions using Proc SGDESIGN. However at the time of writing the current release of the ODS Graphics Designer is designated “pre-production”, and SV plots have not yet been implemented.

CONCLUSION

With care, it is possible to add a great deal of flexibility to ODS graphics templates, thereby simplifying the production of high-quality graphics by the user community.

CONTACT INFORMATION

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

Author Name:	Bob Newman
Company:	Amadeus Software Limited
Address:	Mulberry House, 9 Church Green, Witney, Oxon OX28 4AZ
Work Phone:	+44 (0) 1993 848010
Email:	bob.newman@amadeus.co.uk
Web:	www.amadeus.co.uk