Paper AD12

Let SAS Create SPLUS Graphs

Jacksen Lou, Merck & Co. Inc. North Wales, PA, USA

ABSTRACT

To combine the superb graphic visualization of SPLUS with the powerful data manipulation and analysis features of SAS, this paper introduces a new technical approach and presents a practical application that allows the reader to generate SPLUS graphs from within a SAS environment. The approach creates a software interface between SAS and SPLUS that merges SAS macro language and SPLUS scripting language.
Key Words: Graph, interface, SAS, SPLUS, script.

INTRODUCTION

While SAS is widely celebrated for its robust data manipulation and powerful statistical analysis capabilities, SPLUS is well recognized for its superb graphing and powerful visualization features. Therefore, when graph creation is needed as part of analysis and reporting, a streamlined use of SAS and SPLUS is very desirable in the statistical programming industry.

In recognizing such an industry wide need, the Insightful Corporation[1] assigned a team to build the S-PLUS® Clinical Pack for SAS® users. This pack is able to generate specific graphic displays utilizing a collection of SAS macros interfacing with SPLUS command language. Within the same technical concept as the Clinical Pack, Chris Thornton, et al [2], Robert Treder and Jagrata Minardi[3] have further demonstrated the extended use of SPLUS Trellis graphics in SAS, specifically in SAS ODS.

Almost at the same time, several technical approaches were investigated and put into practice. One of these approaches is to use the specified or fixed SPLUS scripts that are generated from a manually created GUI graph[4]. This approach simplifies the SAS macro input required to control the graphic components. However, due to using the fixed script template the user must first manually create a GUI graph and convert it into a script file as a template every time a slightly different graph is requested. Therefore, when only one or a few similar graphs are needed, the user may spend much more time creating a script than directly creating a graph. In addition, the user needs to have a basic understanding of SPLUS scripting language.

The approach demonstrated in this paper is to operate on a full scale SPLUS script. Because the script is in full scale, we only need to manually create one GUI graph for each graph type (line, scatter, stacked bar, etc.). The user creates a graph by passing the desired values into the pre-selected key parameters built in a SAS macro for all the graphic presentation. Compared with the previously described methods, such a full script approach greatly reduces the manual work necessary to generate template scripts but still allows the user all the graphic flexibility offered by SPLUS. Another advantage is that the users are not exposed to SPLUS language while they are producing SPLUS graphics.

METHODS

PREREQUISITES

The approach assumes that (1) all the results data required to feed the SAS-SPLUS macro are available without a need for further data manipulation or analysis; (2) the layout and / or structure of the input data is what the macro expects; (3) the attributes (e.g., numeric or character) of data variables are formatted to meet the requirements of the SPLUS scripts.

CONSTRUCTING A FULL SCALE SPLUS SCRIPT

SPLUS offers a variety of graphic choices. In SPLUS 6.0, there are more than 50 types of 2-dimension, and more than 30 types of 3-dimension graphs which are available for the user to choose from. Each of these graphic types has its own system defined range of graphical features.

A graph of a certain type can be created under the Graphic User Interface (GUI) manually or programmatically. Once it is created, it can be converted into GUI scripting language in SPLUS's Script

window by copy-pasting a GUI graph to the program window, or by simply dragging the graph into the program window.

A script generated in this manner contains code for all of the graphic components that this type of graph should have, whether a graphic component is utilized in the current GUI graph or not.

UNDERSTANDING SPLUS SCRIPTING LANGUAGE

The approach does not require the user to understand SPLUS scripting language. However, to gain a better understanding of how the scripting methodology works, selected scripting components are presented below.

A SPLUS GUI script is organized in modules. Each module controls a logical section of the graphic features. The following is a partial list of the scripting headers for individual modules and their brief elucidation.

SECTIONAL HEADERS

```
guiCreate("Graph2D"…       <Overall Setting>
guiCreate("MainTitle"…     <Graph Title>
guiCreate("Subtitle"…      <Graph Subtitle>
guiCreate("Legend"…        <Legend Overall>
guiCreate("LegendItem"…    <Legend Item n>
guiCreate("ReferenceLine"…<Reference>
guiCreate("LinePlot"…      <Plot Line n>
guiCreate("YAxisTitle"…    <Y Axis Title>
guiCreate("Axis2DLabelY"… <Y Axis Label>
guiCreate("Axis2dY"…       <Y Axis>
guiCreate("XAxisTitle"…    <X Axis Title >
guiCreate("Axis2DLabelX"… <X Axis Label>
guiCreate("Axis2dX"…       <X Axis>
etc.
```

A header row indicates a start of a functional section. Each section contains a few dozens of lines of scripting code. To drill down further for a little taste of details, the examples of the most critical component sections are further presented, as follows.

OVERALL SETTING

```
guiCreate("Graph2D", Name ="SSBP$1",
   HorizontalShift = "0.2",
   VerticalShift = "0.199999999999999",
   DisplayWidth = "10.6",
   DisplayHeight = "8.1",
   InteriorMarginX = "Auto",
   InteriorMarginY = "Auto",
   Width = "8.48",
   Height = "5.91757",
   xPlotOrigin ="1",
   yPlotOrigin ="1",
   …
```

This section of a script controls overall settings of a graph, such as display area, axis origin, number of panels, graphic borders, filling and line colors, etc.

PLOT LINE

```
guiCreate("LinePlot", Name ="SSBP$1$3",
   PlotNumber = "1",
   DataSet ="SYSTOLb",
   xColumn ="19",
   yColumn ="20",
   zColumn = "",
   wColumn = "",
   …
```

This section of a script defines the plotting line, symbol, sequence, break, etc.

LEGEND

```
guiCreate("LegendItem", Name ="SSBP$1$1$LegendItem1",
   Hide = F,
```

```
   Text ="100mg (100mL) n=6",
   AutoUpdate = F,
   LineStyle ="Solid",
   BreakAtSymbols = T,
   AutoUpdateSymbol = F,
   SymbolStyle ="Circle, Solid",
   …
```

This section of a script defines legend features such as line, symbol, font, border, filling, etc.

AXIS

```
guiCreate("YAxisTitle", Name ="SSBP$1$Axis2dY1$1",
   UseDate = F,
   UseTime = F,
   Title ="Change from Baseline (mmHg)",
   xPosition = "0.505495",
   yPosition = "2.71868",
   …
```

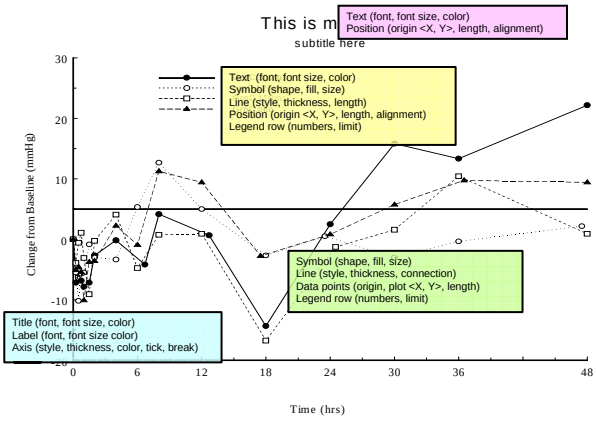
This section of a script controls the X- or Y-axis title in its position, justification, vertical or horizontal margin, etc. Similarly, there are other axis-related sections: for example, Axis2DLabelY to define the Y-axis label, and Axis2DY to define the axis position and style, etc.

The above are just brief examples of code in the scripting sections. There are in fact many more sections in a full script.

IDENTIFYING PLOTTING SECTIONS TO ENABLE FLEXIBILITY

A graph consists of hundreds or thousands diminutive attributes. This usually makes the controlling of graphical attributes a challenging task in terms of what should be made flexible and what should be fixed. Therefore, prior to a making a plan for graphic automation, one needs to understand how the graphic elements are organized in their logical groups. Such an understanding will enable the building of a graphic automation package to be accomplished with much less effort. The following figure presents a sample graph with its basic skeleton.

Figure 1. Analyzing Graphic Components



As illustrated, a simple but typical line-plot graph consists of only a limited number of high level sections - title, axis, plotting area and legend, etc. In each of these sections, there are only several component parameters that are assumed to be of common interest for change, as described in Figure 1. Based on such an investigation, we should be able to work out a specification to enable graphic flexibility. With a workable list of parameters, we can efficiently build an automation package that permits the modification of a limited number of graphic components, while maximizing graphic flexibility with minimal resources. For most routine graphing activities, in fact, we only need a few controlling parameters to generate many rather complicated, yet different graphs.

BUILDING A SAS MACRO TO IMPLEMENT GRAPHICAL FLEXIBILITY

After a full scale script has been created, and a specification for flexibility has been developed, the next step is to build a SAS macro with a series of parameters that offer users the potential to modify any of the identified plotting components (title, axis, plotting lines, etc.) when creating a new graph.

This SAS macro has a full SPLUS script, for the graph type of interest, embedded within it thus the user can easily pass parameter values into the macro to alter the graphic features that already exist in the template script.

The macro consists of a series of parameters which control the key graphic components, such as XAxis= for horizontal axis control. These parameters are organized, by the nature of their tasks, in the following groups:

1> Import SAS data;
2> Deliver the user specified parameters values to the targeted portions of SPLUS script, therefore, altering the graphic display;
3> Call SPLUS software to run the altered script in background;
4> Generate the SPLUS graph;
5> Convert the SPLUS graph into another graphic file format, such as RTF, CGM, etc.

The following SAS macro definition shows an example of how a simple macro can generate many different line-plot graphs.

```
%macro gline
       (DataIn=,        /* input data               */
        GraphName=,     /* output graph name        */
        OutScript=,     /* output script path/name  */
        SPLUSexe=,      /* path/name of SPLUS.exe   */

        MainTitle=,     /* graph main title         */
        SubTitle=,      /* graph sub-title          */

        PlotOrigin=,    /* original point for X|Y   */
        Xvars=,         /* X axis variables         */
        Yvars=,         /* Y axis variables         */
        XMinMax=,       /* plot range for X axis    */
        YMinMax=,       /* plot range for Y axis    */
        Xtitle=,        /* X axis title             */
        Ytitle=,        /* Y axis title             */
        Xlabel=,        /* X axis label             */
        Ylabel=,        /* Y axis label             */
        XYscaling=,     /* X, Y axis scaling        */

        ReferenceLine=, /* reference line           */
        LineStyle=,     /* plot line style          */
        LineConnType=,  /* plotline connection      */
        SymbolStyle=,   /* plot symbol style        */

        LegendText=,    /* legend text              */
        LegendFont=,    /* legend font              */
        LegendPos=      /* legend position          */
       );
```

INTERFACING WITH SPLUS AND EXECUTING THE SPLUS SCRIPT

The following code is a vital part of the SAS macro, and was specifically used to confirm a connection with SPLUS, for importing data, interfacing with the scripts and running the SPLUS script altered via the SAS macro parameters in a batch mode.

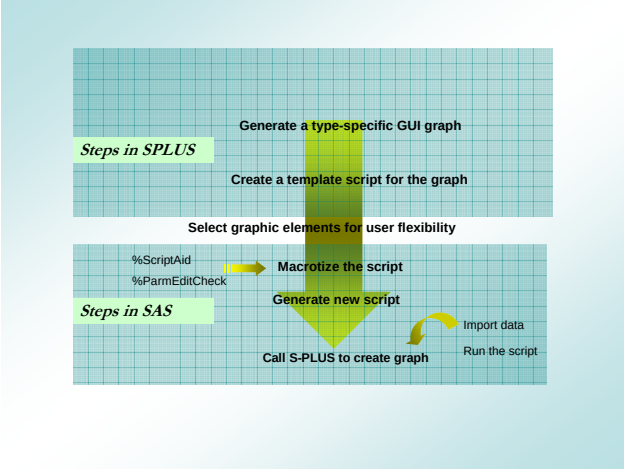
```
*-----------------------------------*;
* Check if SPLUS exists             *;
*-----------------------------------*;
%if %length(&SPLUSexe)=0 %then %let SPLUSexe=C:\Program
Files\Insightful\splus6\cmd\SPLUS.exe;
%if %sysfunc(FILEEXIST(&SPLUSEXE)) eq 0 %then %do;
   %put ERROR: Error message from macro;
   %put ERROR: SPLUS is not installed at specific directory;
   %put ERROR: Error message from macro;
   %goto MacEnd2;
```

```
%end;

*-----------------------------------*;
* Import data                       *;
* open SPLUS and run the script     *;
*-----------------------------------*;
*---  Invoke SPLUS in interactive mode ---*;
x "C:\JZL\work\macros\SPPlot\DEMO\Line\lineplot.ssc && exit.com";
options noxwait noxsync;
data _NULL_;
    call system("&SPLUSexe && EXIT.COM");
    tim=sleep(5);
ruun;

*-----------------------------------------*;
*  Invoke SPLUS in batch mode             *;
*-----------------------------------------*;
filename runssc 'C:\temp\work\macros\SPPlot\lineplot.ssc';
data _null_;
  file "&xptroot\spgraph.bat" ls=3000;
       put "cd &splusexe";
       put "%scan(&splusexe,1,:):";
       put "SPLUS /BATCH " "&splusexe.\%trim(&outscript)_import.ssc" @;
       put " /BATCH " "&outscript" @;
       put " /BATCH " "&splusexe.\%trim(&outscript)_export.ssc";
run;
systask command "&OutScript" wait;
```

SUMMARIZING THE PROCESS FLOW

In order for a reader to understand the process flow, Figure 2 illustrates the major steps utilized in this technical approach.

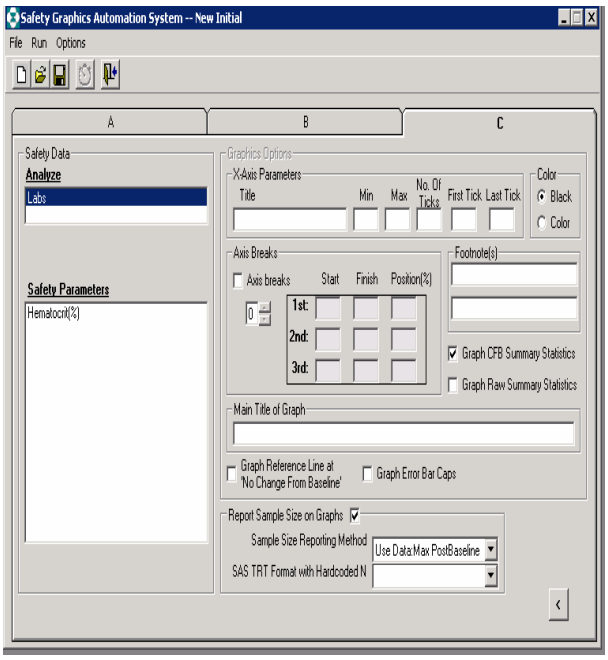
Figure 2. Steps to Generate a Graph using a Full scale SPLUS Script



SAMPLE VISUAL BASIC USER INTERFACE

After a SAS-SPLUS graph automation package has been developed, you may try to reduce the user's confusion (very frequently seen in using graphic software) by building a user interface. Such a user interface can effectively eliminate many errors due to the user's option selection. It has been demonstrated that a Visual Basic user interface built in our system has not only prevented avoidable errors, but also made the graphic production much faster. Below is an example of such an interface that has been used in our SAS-SPLUS graphic system.

Figure 3. A Visual Basic User Interface for a SAS-SPLUS Graphic Automation System



DISCUSSION

The concept and technical solution contained in this paper has been successfully turned into a graphic automation system that has been used in business practice. Using this technology, thousands of graphs have been plotted in a short time (in weeks, if not in days), at ever faster speeds (especially for repeated graphs). However, a SAS programmer who has never been exposed to SPLUS GUI and its scripting language may find it very challenging at the initial stages to gain understanding of the technical concept and acquires the specific skills required to build such a graphic system.

Since the technical approach is based on the SPLUS GUI scripting language, the availability of SPLUS script language and the technical support from Insightful Corporation are critical for its future use.

While using such a graph system does not require any knowledge regarding SPLUS GUI and scripting language, the building such a graph system does require high level skills in several major areas – SPLUS GUI, SPLUS scripting language, SAS macro language, and understanding the differences in the language expressions, and interactions between SAS and SPLUS.

CONCLUSION

The approach introduced in this paper has been proven as a technically feasible and practical method in the industry to create routine graphs. The approach can quickly generate a large amount of high quality SPLUS graphs within a short timeframe. Compared with previously existing similar methods, this approach offers more flexibility with easier user control. The approach waived the user of having any skills or knowledge with SPLUS GUI or GUI scripting language. However, developing a graph system with this approach has proven to be challenging task in both SAS and SPLUS.

REFERENCES

[1] Insightful Corporation, 2004. S-PLUS® Clinical Pack for SAS® Users. http://www.insightful.com/PDF/S-PLUS_SAS_Clinical_Pack.pdf.

[2] Robert Treder and Jagrata Minardi, 2004. Extending ODS Output by Incorporating Trellis™ Graphics from S-PLUS. PharmaSUG, PO14.

[3] Chris Thornton, Sven Knudsen and Michael O'Connel, 2005. Extending the SAS® Environment with Trellis™ Graphics. PharmaSUG, PO18.

[4] Qian Wang and Carl Herremans, 2006. Graph Automation Using Integrated SAS/S-PLUS Solution. SUGI, TS03.

ACKNOWLEDGEMENT

This work was part of an effort of the Merck Global Graphic Committee, and also part of an in-life clinical project. Many people, more than 20, have partially contributed in system development, discussions and decision makings. The author would like to thank all these diligent Merck employees. In particular, my thanks go to Cindy Song (currently with Sanofi-aventis US) and Frederic Coppin, for their enthusiasm to promote this approach as a uniform standard graphic method in Merck a few years back.

CONTACT INFORMATION

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

Jacksen Lou
Merck & Co., Inc.
UG1CD-14, PO Box 1000,
North Wales, PA 19454-1099
(267) 305-7482
jacksen_lou@merck.com