PharmaSUG 2023 - Paper AP-048

Documenting your SAS programs with Doxygen and automatically generated diagrams.

Philip Mason, Wood Street Consultants Ltd, England

ABSTRACT

Doxygen has been used to document programs for over 25 years. It involves using tags in comments to generate HTML, RTF, PDF and other forms of high-quality documentation. It supports the DOT language for making diagrams from simple text directives.

PROC SCAPROC can be used to generate a trace of a SAS® programs' execution. My SAS® code can then analyse the trace and produce DOT language directives to make a diagram of the execution of that SAS program. Those directives can then be put into the Doxygen tags to add the diagram to your documentation. And the analysis can also show the performance of the SAS program to be used for tuning purposes.

This paper shows how to use Doxygen with SAS and provides the code to automatically produce diagrams for that documentation or tuning purposes.

INTRODUCTION

Part 1 – Doxygen has been around for a long time and used to document many other languages other than SAS, although in recent years people have started to see the value in using Doxygen for SAS. Although there is not built in support for SAS as a language you can make use of the many extensive parameters to adapt it to do a very good job at documenting the language. In this paper I will be sharing things I have learnt through trial and error, as well as with help from Allan Bowe in his documentation of the SASjs code base with Doxygen. This should be enough to get you using Doxygen effectively to make great documentation of your code automatically.

Part 2 - PROC SCAPROC has been available in SAS since SAS 9.2. It implements the SAS Code Analyzer which provides information about input tables, output tables and macro symbols in a SAS program. Its easy to read this information in with SAS, which gives you a nice view of how the program runs. Having that information means it is possible to generate a diagram showing how that program runs. We will look at the following steps:

1. Generating information about the flow of a program
2. Reading that into SAS
3. Defining how to produce a diagram
4. Making the diagram

PART 1 – DOXYGEN

DOCUMENTING SAS CODE

There are many ways to document SAS code. Often SAS programmers have their own favorite method, but at times a company will have a published standard or SOP to follow that lays out precisely how it should be done. We can document in a minimal way perhaps with a header like this:

```
* program: test.sas
  purpose: to do a test ;
```

Or we can go into lots of detail with extensive headers covering macros used, datasets created/modified/read, macro variables used, etc. Additionally, we liberally scatter comments throughout the code to describe what blocks of code do or to explain complex sections of logic.

To use Doxygen we need to adapt our comments to be done in the required format, since Doxygen will read all our SAS code to extract the information it needs to make the documentation. Fortunately, the

format they must be in is a good and flexible layout that provides lots of possibilities. When we see what it supports it will also encourage us to document more about our programs since that information will be extracted and displayed in a very user-friendly form for others.

Doxygen uses specific tags to identify different parts of the documentation. A simple header might look like this:

```
/** @file a.sas
 *  @brief This is a test file
 *  @author John Doe
 *  @date 2023-12-31
 */
```

Tags are prefixed by an '@' symbol. The header block must start with '/**' and ends with '*/'. This is great as that is compatible with how our standard SAS comment blocks work. You have to at least have @file specified if you want documentation generated, but other tags are all optional. Some simple tags used in the example are:

- file – file name of the SAS program (you can leave it blank and Doxygen will fill it in for you)
- brief – brief description of the program
- author – author of the program
- date – date the program was created.

There are many tags available, and we will look at some that I have found most useful as we move on.

Another thing to keep in mind is that it works with most other languages, so if you have code in R, Python, C, etc. then you can generate documentation for those programs as well.

Another powerful feature of Doxygen is that it supports markup and html, so you can include that in your comments which opens a lot of possibilities for making nice tables (for example). Or you can add markup files with a '.md' extension which will show up in the documentation too.

SETTING UP DOXYGEN

You will need to get the Doxygen program to run it against you SAS code to generate documentation in any of the supported output formats. I always generate a web site, which can then be moved to a server of your choice to make it available to users. A trick we use is to generate the web site with **.aspx** extensions for the html files, rather than .html extensions. This means we can copy the web site to Microsoft SharePoint, and it will work perfectly from there (html files don't work).

You can download Doxygen from here https://www.Doxygen.nl/download.html

Install Doxygen by running the setup program, or via one of the other methods described on the web site.

You can use Doxygen in two ways, either by using the Doxygen wizard or creating a parameter file and running Doxygen from the command line with that. I recommend using doxywizard at least until you become familiar with it. You can choose all the settings you want to use in the wizard and then save them into a parameter file which can later be used without the wizard. If you want a copy of my doxyfile (parameters that you can use) then drop me an email.

There are some parameters that I have found essential to get Doxygen to work with SAS code. I have described the ones that I use and find useful in working with SAS.

Tab	Topic	Comments
Wizard	Project	Enter a project name, synopsis, and version/id (which will appear in your documentation prominently). I also grab a company logo and then enter that as the company logo, which makes it look far more professional. You can enter a source code directory here if you only have one. Otherwise wait for the expert section where you can select many. Choosing to scan recursively is great if you just want to scan through lots of directories under another.

		Use the select button to easily pick where you want output to be put.
Wizard	Mode	It's important to choose **Optimize for Java or C# output**, since that works best with SAS code.
Wizard	Output	I usually generate HTML with a navigation panel and search function, which I find very useful. If you want PDF documentation, then choose LaTeX which can later be converted into PDF easily. If you want Microsoft Word documentation, then choose RTF.
Wizard	Diagrams	I usually select to use the dot tool from Graphviz package. This will allow you to specify diagrams with the simple DOT language in your SAS documentation that will be converted into high quality SVGs in your final documentation. You can use my method to automatically generate diagrams and then put them in as text.
Expert	Project	Any options that are changed from defaults are shown in red, which is very helpful. Hovering over an option will show a description of what it is used for. **Create_subdirs** is useful if producing a large amount of documentation. I usually have **repeat_brief** unticked, or else I get brief descriptions repeated. I find **Javadoc_banner** ticked useful. **Optimize_output_java** should be ticked as it works best with SAS. **Extension_mapping** is one of the most important options, as you need to enter **sas=Java** and press the plus button to add it. That means that it will treat SAS code like it was java code, which for our purposes works well.
Expert	Build	**Extract_all** should be ticked. **Case_sense_names** should be set to No since SAS isn't case sensitive.
Expert	Input	You can use the plus button to add directories that you want to process and build documentation from. In **file_patterns** you can use the plus to add ***.sas** so that SAS files are processed from the selected directories. You can also control what other files will be scanned as it tries to build documentation. You can tick recursive if you want subdirectories to be scanned too. **Use_mdfile_as_mainpage** is an option I like to use, which lets me put the name of a markdown file in and then use that as the main starting page. You can customize this in any way, including linking to other parts of your documentation.
Expert	Source Browser	I suggest you tick **source_browser** since it allows you to browse the actual source code from your documentation, as well as cross reference links being generated where a source program name is mentioned. I suggest unticking **strip_code_comments** since that will prevent the comment blocks from being removed when browsing the source code – my preference is to see them.
Expert	HTML	You can use the button to browse for a folder that you want to save your **HTML_OUTPUT** into. I usually change the **HTML_COLORSTYLE_*** attributes too. I use this https://www.w3schools.com/colors/colors_picker.asp to browse for a color that matches the corporate colors and then enter the values for the color style.

		I tick **generate_treeview** to get a tree-like structure for navigation through the documentation.
Expert	Dot	If you are using DOT diagrams in your documentation, then you can play with these settings. I tick **have_dot** to state that I have Doxygen installed locally. Then I can click on the button next to **dot_path** to browse to the directory that contains Doxygen. I also tick **interactive_svg** so I can clickable graphics for navigation in some parts of doc.
Run		Show configuration is handy since it will show all the options selected. If you tick condensed, then it only shows the options you have changed from default values. Clicking on Run Doxygen will do so. If you were generating html output, you will then be able to click on show HTML output to see it.

You can run Doxygen to produce your documentation once you are happy with your choices. It doesn't take long to run so it is easy to try different things out and run it again and again until you get the output you are happy with. You can save your settings from the File/save as menu at the top of window. You can also choose to use the current settings at startup from the settings menu.

You can even generate multiple outputs from one run, handy if you want PDF and Web site documentation.

DOXYGEN TAGS I USE

I have looked through all the tags available thinking about which are useful for SAS programmers and these are the ones I have ended up using, although there are many others (see them here https://www.doxygen.nl/manual/commands.html) so it is worth you having a look at what is available in case there is a use case for that you have for any others.

Tag	Comments
file	Filename can go here – leave blank for Doxygen to fill it in. This is required.
brief	Brief description of program that appears in summary and detailed documentation
details	Detailed description of program. I try to avoid putting things in here which fit more neatly under another tag, e.g. author. Details can go over multiple lines. Indenting code several spaces will format it as a code block.
author	Author of the program
@b @c @e @li	Format the next word **bold**, format the next word as `program code` (non-proportional), format the next word as *emphasized text* (italics), shows a • list item
@param @param[in] @param[out]	Define a general parameter, define an input parameter, define an output parameter
@date	Date that program was written (or last updated, etc.)
@pre	Pre-condition for using program, e.g., macro variables must be set in a particular way
@todo	Things left to do, which are then all collected in a special section so you can see everything to do across all your programs, as well as in each individual piece of documentation
@bug	Note any bugs that need addressing, which like todo are collected in a special section so you can see all the bugs in your code base.

EXAMPLE CODE TEMPLATES

I have put a small collection of Doxygen templates in GitHub for you to examine and potentially use as a starting point for your own. They are located here: https://github.com/philipmason/Wood-Street-Consultants/tree/main/Doxygen.

SOME MORE TIPS

- You can make Doxygen skip over looking at sections of your code by using the tags **@endcond** and **@cond**. We use this when we want to have a code header at the start of our SAS code, but then list examples of how to use the code at the bottom. So, at the end of the code header, we switch off processing with **@endcond**, and then in the comment block with examples we turn processing back on with **@cond**. In the generated documentation the examples follow on directly from the other info.
- Diagrams can be included in your documentation, which means you can either manually create them using DOT syntax or use the method in part 2 of this paper to automatically generate diagrams, which can then be put into the header, so they appear in documentation. **@dot** is used to start a section of text with DOT commands, **@enddot** signifies the end of the section. e.g.,

```
@dot
digraph example {
in -> out
out -> report
}
@enddot
```

- Sections of code can be put in between **@code** and **@endcode**, and they will be clearly laid out in non-proportional font as program code. Special characters or tags won't be interpreted in in the code section.
- Standard markdown can be used with Doxygen, e.g., **SAS** would be shown in bold as **SAS**, ## Welcome ## would be shown as a level 2 title.
- Standard HTML can be used with Doxygen, e.g., <u>SAS</u> would be shown underlined as <u>SAS</u>.
- Mentioning another program in your documentation will automatically make a link to it. e.g. I might say "this program called dm.sas" which would make a clickable link to dm.sas documentation.

PART 2 – AUTOMATICALLY GENERATING DIAGRAMS OF SAS CODE EXECUTION

HOW IT'S DONE

You can read all about the SCAPROC procedure, but I will tell you what you need to know for this process to work. The way it works is that you start it running to **record** the activity of your SAS program, and then when your program has finished you use PROC SCAPROC to **write** that information out. So, a simple use of it would be like this…

```
proc scaproc ;
   record "%sysfunc(pathname(work))/example1.txt";
run ;
data test ;
   set sashelp.prdsale sashelp.prdsal2 ;
run ;
proc scaproc ;
   write ;
run ;
```

and the output produced is like this (note the bold items of interest) …

```
/* JOBSPLIT: JOBSTARTTIME 06MAR2023:07:02:51.92 */
/* JOBSPLIT: TASKSTARTTIME 06MAR2023:07:02:51.92 */
/* JOBSPLIT: DATASET INPUT SEQ #C00001.PRDSALE.DATA */
/* JOBSPLIT: LIBNAME #C00001 V9 'C:\Program Files\SASHome\SASFoundation\9.4\core\sashelp' */
/* JOBSPLIT: CONCATMEM #C00001 SASHELP */
/* JOBSPLIT: LIBNAME SASHELP V9 '( 'C:\Program Files\SASHome\SASFoundation\9.4\nls\en\SASCFG' 'C:\Program
Files\SASHome\SASFoundation\9.4\core\sashelp' 'C:\Program Files\SASHome\SASFoundation\9.4\aacomp\sashelp' 'C:\Program
Files\SASHome\SASFoundation\9.4\cas\sashelp' 'C:\Program Files\SASHome\SASFoundation\9.4\cmp\sashelp' 'C:\Program
Files\SASHome\SASFoundation\9.4\graph\sashelp' 'C:\Program Files\SASHome\SASFoundation\9.4\mlearning\sashelp'
'C:\Program Files\SASHome\SASFoundation\9.4\spdsclient\sashelp' 'C:\Program
```

```
Files\SASHome\SASFoundation\9.4\stat\sashelp' )' */
/* JOBSPLIT: DATASET INPUT SEQ #C00001.PRDSAL2.DATA */
/* JOBSPLIT: LIBNAME #C00001 V9 'C:\Program Files\SASHome\SASFoundation\9.4\core\sashelp' */
/* JOBSPLIT: CONCATMEM #C00001 SASHELP */
/* JOBSPLIT: LIBNAME SASHELP V9 '( 'C:\Program Files\SASHome\SASFoundation\9.4\nls\en\SASCFG' 'C:\Program
Files\SASHome\SASFoundation\9.4\core\sashelp' 'C:\Program Files\SASHome\SASFoundation\9.4\aacomp\sashelp' 'C:\Program
Files\SASHome\SASFoundation\9.4\cas\sashelp' 'C:\Program Files\SASHome\SASFoundation\9.4\cmp\sashelp' 'C:\Program
Files\SASHome\SASFoundation\9.4\graph\sashelp' 'C:\Program Files\SASHome\SASFoundation\9.4\mlearning\sashelp'
'C:\Program Files\SASHome\SASFoundation\9.4\spdsclient\sashelp' 'C:\Program
Files\SASHome\SASFoundation\9.4\stat\sashelp' )' */
/* JOBSPLIT: DATASET OUTPUT SEQ WORK.TEST.DATA */
/* JOBSPLIT: LIBNAME WORK V9 'C:\Users\pmason\AppData\Local\Temp\SAS Temporary Files\_TD9604_ARGENXLAP3120_' */
/* JOBSPLIT: FILE OUTPUT C:\Users\pmason\AppData\Local\Temp\SAS Temporary Files\_TD9604_ARGENXLAP3120_\example1.txt */
/* JOBSPLIT: ELAPSED 29  */
/* JOBSPLIT: SYSSCP WIN */
/* JOBSPLIT: PROCNAME DATASTEP */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
data test ;
   set sashelp.prdsale sashelp.prdsal2 ;
run ;
/* JOBSPLIT: JOBENDTIME 06MAR2023:07:02:51.95 */
/* JOBSPLIT: END */
```

The key bits of information that we can extract from this text is as follows:

- JOBSPLIT: DATASET INPUT - tells us what tables were read in by the step
- JOBSPLIT: DATASET OUTPUT - tells us what tables were written out by the step
- JOBSPLIT: ELAPSED - tells us the elapsed time taken for the step, in milli-seconds
- JOBSPLIT: PROCNAME - tells us the type of step, e.g., DATASTEP, FREQ, SORT, etc.

There is other info that we can get which can be useful but to make a diagram showing the flow of data in a program only requires the inputs and outputs from steps. I like to also know what kind of step it was, and how long it took.

READING IN THE OUTPUT OF PROC SCAPROC

To read the information in from the PROC SCAPROC text is quite easy. You just need to scan the lines for the information you need, and to keep it in some variables. I use the scan function to break lines up into words separated by spaces.

The following code looks at words separated by spaces from lines read in. It then works out which are input tables, output tables and what step they belong to. This would then be the bare minimum needed to draw a diagram of the data flow.

```
data scaproc_parsed(keep=in out step) ;
         retain step 1 ;
         infile "c:\users\phil\scaproc.txt" ;
         input ;
         word1=scan(_infile_,1,' ') ;
         word2=scan(_infile_,2,' ') ;
         word3=scan(_infile_,3,' ') ;
         word4=scan(_infile_,4,' ') ;
         word5=scan(_infile_,5,' ') ;
         word6=scan(_infile_,6,' ') ;
         word7=scan(_infile_,7,' ') ;
         if word2='JOBSPLIT:' & word3='DATASET' & word4='INPUT' then in=word6 ;
         if word2='JOBSPLIT:' & word3='DATASET' & word4='OUTPUT' then out=word6 ;
         if word2='JOBSPLIT:' & word3='STEP' then step+1 ;
         if in>'' or out>'' then output ;
run ;
```

This program will read the previous SCAPROC text file and result in a SAS table like this …

	step	in	out
1	1	#C00001.PRDSALE.DATA	
2	1	#C00001.PRDSAL2.DATA	
3	1		WORK.X.DATA

You will notice that instead of SASHELP, I have #C00001 as the libname recorded. There is another line in the SCAPROC text file that tells us what the actual libref is for this, so we can code around this issue although I haven't done so here for simplicity.

What we can see from this data is that step 1 has 2 input tables and 1 output table. With that information we can draw a simple diagram. Or we could use this information to sit down with VISIO and construct a

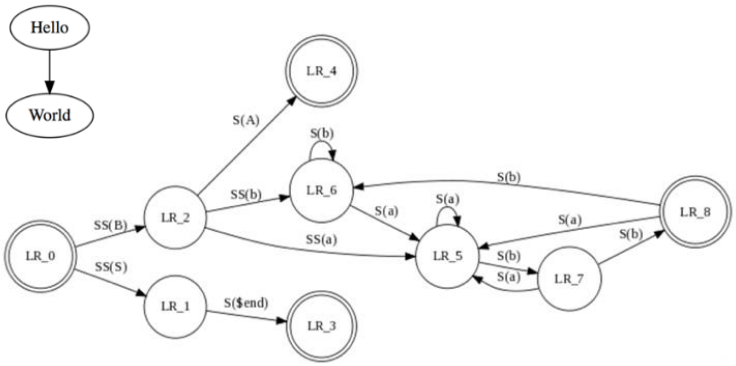
diagram from it. However, if we distill this information into a language used to make diagrams, then we can make one automatically.

GRAPHVIZ

Graphviz is open-source graph visualization software that has been around for a very long time. It uses a simple text language as input which is used to construct a diagram. This language is called the DOT language. Here is a simple world graph, followed by the DOT language for a more complex one.

```
digraph G {Hello->World}

digraph finite_state_machine {rankdir=LR;
        size="8,5"
        node [shape = doublecircle];
                LR_0 LR_3 LR_4 LR_8;
        node [shape = circle];
        LR_0 -> LR_2 [ label = "SS(B)" ];
        LR_0 -> LR_1 [ label = "SS(S)" ];
        LR_1 -> LR_3 [ label = "S($end)" ];
        LR_2 -> LR_6 [ label = "SS(b)" ];
        LR_2 -> LR_5 [ label = "SS(a)" ];
        LR_2 -> LR_4 [ label = "S(A)" ];
        LR_5 -> LR_7 [ label = "S(b)" ];
        LR_5 -> LR_5 [ label = "S(a)" ];
        LR_6 -> LR_6 [ label = "S(b)" ];
        LR_6 -> LR_5 [ label = "S(a)" ];
        LR_7 -> LR_8 [ label = "S(b)" ];
        LR_7 -> LR_5 [ label = "S(a)" ];
        LR_8 -> LR_6 [ label = "S(b)" ];
        LR_8 -> LR_5 [ label = "S(a)" ];
}
```



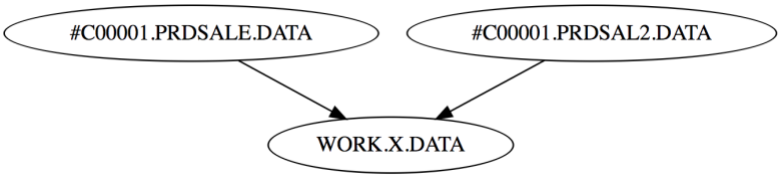
Defining a diagram

We can use the data from our table which was produced after analyzing the PROC SCAPROC text file. That data shows us that step 1 has 2 inputs and 1 output. In DOT language this can be represented like this…

```
    digraph {
    "#C00001.PRDSALE.DATA" -> "WORK.X.DATA"
    "#C00001.PRDSAL2.DATA" -> "WORK.X.DATA"
    }
```

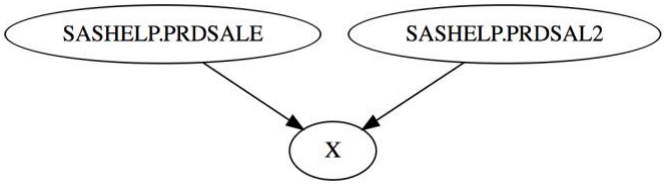
Making a diagram

Once we have the DOT commands to make a diagram, then we can use that with the Graphviz software to generate it. This can be done in several ways. You can install the software on your own machine and run it locally on that machine. Or if you don't have the software installed you can use a web version of it, such as WebGraphviz, to generate the diagram. So, if I copy and paste this DOT language into WebGraphviz, it generates the following diagram…



I could, of course clean up the text a bit manually, or enhance my SAS program to do it for me, thus making a diagram like this…

```
    digraph {
    "SASHELP.PRDSALE" -> "X"
    "SASHELP.PRDSAL2" -> "X"
    }
```



Automating the approach

I have made some macros to enable this technique to be used more easily on all kinds of SAS code. The macros are:

- **eanbegin** - turns on recording with PROC SCAPROC.
- **eanend** - turns off recording and writes output to an external file.
- **percentiles** - calculates some percentiles based on the run time of steps, so we know which are the longest ones.
- **scaproc_analyse** - analyses the output from PROC SCAPROC and produces DOT language that is used by Graphviz to make a diagram.

To use this in your own SAS program

1. You need to have the macros accessible either via an auto-call library or just run them first. Just copy them from this document and run in your SAS session.
2. Turn on Enhanced Logging:
   %let _eandebug=scaproc ;
3. Put the %EANBEGIN macro before your code.
4. Put the %EANEND macro after your code.
5. Run those macros with your code.
6. Copy the lines from the table just created (GRAPHVIZ).
   a) View the table.
   b) Click on column, to select all values in it
   c) Then control-C.
7. Go to http://webGraphviz.com
   a) Paste the lines into the Text Area.
   b) Press "Generate Graph!" button.
8. Now you will have your diagram displayed in your web browser. You can copy it into documentation, or you could save it as a SVG (small and scalable), PNG (small and not scalable) or BMP (big and not scalable). I would suggest SVG as the file size is small and you can scale it as big or small as required without reducing quality.

Example showing how we can make a diagram

The statements highlighted in yellow are the ones which are used to create the information needed to make a flow diagram of your SAS code.

```
%* set macro variable to turn on SCAPROC and set verbose logging ;
%let _eandebug=scaproc,verbose;

%* Start recording SCAPROC data ;
%eanbegin(Sample 1)

*******************************************************************************;
*** This is the sample program we will measure and then make a flow chart ;
```

```
*******************************************************************************;
data x ;
   set sashelp.class ;
run ;

data y ;
   set sashelp.class ;
run ;

proc summary data=x ;
   class sex ;
   var height ;
   output out=x2 mean= ;
run ;

proc summary data=y ;
   class sex ;
   var height ;
   output out=y2 mean= ;
run ;

proc sort data=x2 out=x3 ;
   by sex ;
run ;

proc sort data=y2 out=y3 ;
   by sex ;
run ;

data z ;
   merge x3 y3 ;
      by sex ;
run ;

proc print ;
run ;

proc sql ;
   create table sql_table as
   select *
   from x
   left join
   y
   on x.sex=y.sex ;
quit ;


*****************************;
*** finish of sample program ;
*****************************;

%* Finish recording SCAPROC data and write it out ;
%eanend


%* Generate the Graphviz dot language to be used to make diagram ;
%scaproc_analyse
```

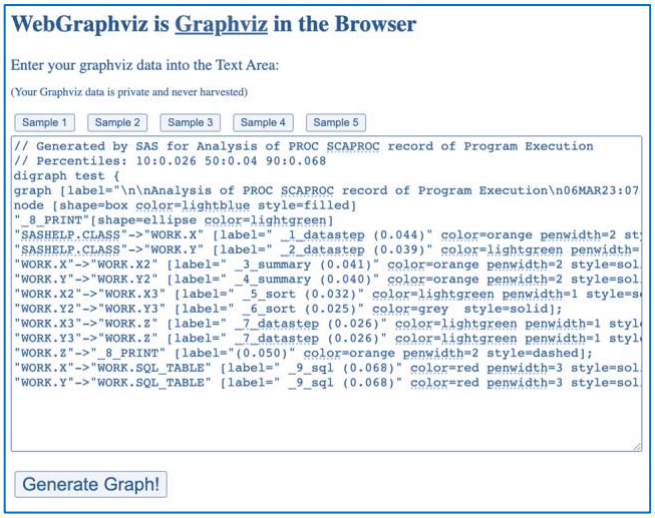
The code above produces a file containing the SCAPROC data, which can be seen here https://github.com/philipmason/Wood-Street-Consultants/blob/main/Doxygen/SAS/output/example2.txt

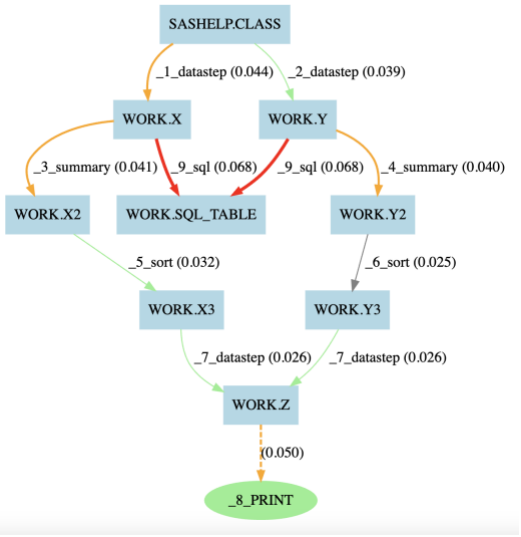
Several tables are produced by the scaproc_analyse macro, but the main one that we are interested in is called Graphviz. This contains the DOT language statements which are used to make the diagram.

```
// Generated by SAS for Analysis of PROC SCAPROC record of Program Execution
// Percentiles: 10:0.026 50:0.04 90:0.068
digraph test {
graph [label="\n\nAnalysis of PROC SCAPROC record of Program Execution\n06MAR23:07:57:46"]
node [shape=box color=lightblue style=filled]
" 8 PRINT"[shape=ellipse color=lightgreen]
"SASHELP.CLASS"->"WORK.X" [label=" _1_datastep (0.044)" color=orange penwidth=2 style=solid];
"SASHELP.CLASS"->"WORK.Y" [label=" _2_datastep (0.039)" color=lightgreen penwidth=1 style=solid];
"WORK.X"->"WORK.X2" [label="  3 summary (0.041)" color=orange penwidth=2 style=solid];
"WORK.Y"->"WORK.Y2" [label="  4 summary (0.040)" color=orange penwidth=2 style=solid];
"WORK.X2"->"WORK.X3" [label=" _5_sort (0.032)" color=lightgreen penwidth=1 style=solid];
"WORK.Y2"->"WORK.Y3" [label="  6 sort (0.025)" color=grey  style=solid];
"WORK.X3"->"WORK.Z" [label=" _7_datastep (0.026)" color=lightgreen penwidth=1 style=solid];
"WORK.Y3"->"WORK.Z" [label=" _7_datastep (0.026)" color=lightgreen penwidth=1 style=solid];
"WORK.Z"->" 8 PRINT" [label="(0.050)" color=orange penwidth=2 style=dashed];
"WORK.X"->"WORK.SQL TABLE" [label="  9 sql (0.068)" color=red penwidth=3 style=solid];
"WORK.Y"->"WORK.SQL_TABLE" [label=" _9_sql (0.068)" color=red penwidth=3 style=solid];
```

```
}
```

You can now copy the rows from this table and paste them into WebGraphviz. Then click on the "Generate Graph!" button and it will make your diagram. Data appears in blue rectangles. The lines represent the data steps or procedures that ran. The thicker the line, the longer it took. Red lines took longer than the 50th percentile. Procedure outputs appear in green ovals.





FUTURE ENHANCEMENTS

There are an almost unlimited number of enhancements that can be made using these basic techniques outlined here.

1. Install Graphviz in a place accessible from SAS server so that we can then use it directly from a SAS program. This will enable the graphics to be produced automatically thereby removing any manual steps. This has already been tested on another server and works well.
2. Integrate output from SAS log so that we can choose to display various numbers on arrows and scale and color them accordingly, such as Elapsed time, CPU time, records in/out, etc.
3. Fix "#" references from PROC SCAPROC with info from code and/or log to show what library they come from.
4. Integrate tool into a SAS 9.4 Stored Process or SAS Viya job, to allow running an entire analysis by filling in some parameters and browsing to your code.
5. Make simple version of diagram based only on SAS code.
6. Make diagram that can be generated from a SAS log, enabling us to create diagrams on logs run in the past.
7. Create framework to store graph DOT commands and then enable comparison with others previously stored, so that we could compare performance between multiple runs, highlight changes in the structure of complex code diagrammatically, etc.
8. Provide parameters to allow changing of line attributes, box attributes, etc.
9. Add support for showing diagrams in a web page, which then will enable use of tooltips and links which will enable a lot more functionality such as popping up a box showing code used in a step directly from diagram.

10. Enhance code to use the grid analysis capabilities. This will suggest how to grid enable your SAS code, and how much time could be saved.

APPENDIX

SAS Macros

The following SAS macros are the version of these at the time of creating this document. They show the technique although the code is currently under development so it might not handle the analysis of all kinds of SAS code and won't have features currently being added in development. Feel free to make your own enhancements and share the code with others.

1. **EANBEGIN** – "Enhanced Analysis" begin, which is used before some SAS code that you want to analyze. This follows the SAS naming scheme along the lines of STPBEGIN/STPEND. Parameters are:
    a. **use_label** is a positional parameter which can optionally specify a label to use in .DOT output that is produced. If you don't specify a value, then one is created for you.
    b. **scaproc_file** specified the name of the file to write the scaproc output to.
    c. **where** is a keyword parameter which can optionally specify a location for PROC SCAPROC to write its output to, will default to a text file in your own home directory.
    d. **_options** is a keyword parameter which can optionally specify options to use when using PROC SCAPROC, they are just passed through to the procedure itself.
2. **EANEND** – "Enhanced Analysis" end, which is used at the end of SAS code which is being analyzed. This causes the analysis to be written out and is then ready to be analyzed. This macro has no parameters.
    a. **PERCENTILES** – Creates a range of global macro variables in the form PCTn, where n is the percentile. E.g. PCT50 is the 50th percentile. There is a default list of them created although you can override which ones are created.
    e.g. *%percentiles(sashelp.air,air,pctlpts=25 50 75)*
    b. **dset** is a positional parameter which specifies the dataset to use
    c. **var** is a positional parameter which specifies the variable to use from dataset for generating %tiles
    d. **pctlpts** is a keyword parameter which can optionally specify a list of percentiles to generate, separated by spaces. A global macro variable will be created for each one. They are prefixed with PCT. e.g. 10 would produce &pct10
3. **SCAPROC_ANALYSE** – Reads in a text file containing the output from PROC SCAPROC and produces a table containing DOT directives. These are a set of directives used with a program called Graphviz which enable graphs & diagrams to be produced. There are some manual steps that must then be taken which are outlined in the description below.

You can download the SAS macros and other things from https://github.com/philipmason/Wood-Street-Consultants/tree/main/Doxygen/SAS.

CONCLUSION

Combining SAS with Doxygen and Graphviz gives you the tools to build professional documentation and diagrams. With very little additional effort you can take your documentation to the next level. You can also make simple diagrams with very little programming or put all kinds of extra effort into creating far more elaborate and explanatory diagrams. Finally the diagrams can even be included directly into your documentation.

RECOMMENDED READING

PROC SCAPROC documentation can be found here
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p0sf63lx4fs2m5n14qv1bn8p863v.htm

Documentation for Graphviz, which enables Graph Visualization via directives provided in text form can be found at http://www.Graphviz.org/

DOT language info & links are herre https://en.wikipedia.org/wiki/DOT_(graph_description_language)

Web Version of Graphviz, which enables us to use Graphviz without installing it is here http://webGraphviz.com/

CONTACT INFORMATION

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

Philip Mason phil@woodstreet.org.uk

Wood Street Consultants, UK (currently doing work for argenx in Belgium and BOTH analytics in Germany)