DrSearch: Recursive SAS® Macro solution for understanding complete program dependencies in large SAS® reporting system (LSRS)

Jim Fang
sanofi-aventis U.S., Bridgewater, NJ

ABSTRACT

Maintenance of a large scale SAS Reporting System (LSRS) comprised of hundreds of SAS programs, SAS datasets and thousands of reports is a challenging task. One of the key challenges is frequent change requests from the end users demanding quick turnaround time.

This paper is an extension of the work we presented at NESUG 2006, which identified inputs and output files of each SAS program and recorded in the form of a Excel spreadsheet. If we modified the program N, we can manually search the spreadsheet to identify the programs directly depended on the program N’s output. But the complete dependency is still not clear until you continuously search direct dependency of dependent programs until no dependency can be found. This paper demonstrates an automated solution to determine the complete dependencies of all the SAS programs in the LSRS.

Such a document greatly helps us meet the challenging turnaround time underlying these change requests. Implementation of this utility will help in code design, identify issues in programs and ultimately result in better project management of large-scale SAS implementations.

KEY WORD

Recursive SAS Macro, Large SAS reporting system, Program dependency, Excel spreadsheet

INTRODUCTION

The Large SAS Reporting System (LSRS) at sanofi-aventis runs hundreds of SAS programs in a sequence to produce thousands of field based reports. Every program reads external files or SAS datasets and outputs SAS datasets or external files (Fig 1). The output SAS dataset of one SAS program forms the input SAS dataset for one or more following programs.

When change or enhancement requests are received, a programmer uses reverse engineering to identify the programs which need modifications or enhancements. The process begins with the programmer examining the inputs of the program which is writing the final reports (p150.sas Fig 1.). Suppose the programmer identifies that the input dataset out_data12 needs modifications or enhancements. Next the programmer searches the input-output spreadsheet to identify the program that outputs out_data12 and this program is p02.sas (Fig 2). Based on the analysis above either p02.sas might need modification or other programs creating input datasets for p02.sas need modification. Hence this bottom-up approach finds out all the programs, external files and data sets that need to be modified or enhanced to satisfy the change request.

Fig 1. LSRS Input-Output Spreadsheet
But after the programmer fixes the problem in program P02, he or she may not have enough time to run the whole process to reproduce the results. Or sometimes he or she cannot run the whole process because certain input files have been updated and cannot be used in the rerun.

Because the program dependencies for LSRS are usually very complex like a matrix (Fig 2.), we can save a lot of time if we only run the necessary programs. But it is very difficult to manually keep the matrix information up to date if the system is frequently updated. Running from program P02 to the last one seems to be the only way but not the smartest way. Using the following recursive SAS macro, we can identify all the programs with direct or indirect dependency with the program P02. Hence, no programs will be missed during rerun and the shortest time is needed to reproduce the reports.

**ALGORITHM FOR DETERMINING THE COMPLETE DEPENDENCY OF THE LARGE SAS REPORTING SYSTEM**

We all know that you can call another SAS macro within a SAS macro, which is called nested macro calls. When the SAS macro calls itself, we refer to it as a recursive SAS macro. This technique is very useful to resolve complex recursion problems like this one.

**STEPS**

1. **Find direct dependency information (Fig 3).**
   
   After we created the dataset with inputs and outputs for each program, we can self join the dataset by inputs to outputs to get the direct dependencies.
   
   Then we should remove the following kinds of bad dependencies:
   
   - No child dependency: there is no dependent program. In other words, the program created a permanent dataset not used in any other programs.
   - Self recursive dependency: The program is dependent on itself because the permanent dataset is overwritten within the same program.
   
   These 2 dependencies are usually bad programming habits. The first one takes extra space, unless there are important QC exceptions you probably don’t want to keep them in your system. The second one can lead recursive search into infinite loops and can be avoided by restructuring the programs.

2. **Prepare for recursive search.**

   Before we can use the power for a SAS macro to do the recursive search, we have to do some preparation work. We need to rename variable input ID to output ID and output ID to original ID. Then the direct dependency relations are served as a baseline to find the next level of dependencies recursively. Input ID is matched with output ID to bring the next level of dependencies into the final relations table.

3. **Recursive search for complete program dependencies.**

   Every recursive macro needs to have an exit point. Otherwise the program may run forever until no more memory exists. The stop point for Dr Search is no more dependency can be found in the system.
macro will keep calling itself until this condition is met. Afterwards, the complete program dependency table is finalized without the duplicated information. Finally, the results can be exported into a CSV file to be displayed in the Excel workbook (Fig 4).
CONCLUSION
A Recursive SAS macro is a special type of nested SAS macro. It is a very powerful programming technique to resolve recursion problems. For example, the Fibonacci numbers are most commonly defined recursively. Our first reaction to the problem is that it will require a complex iterative algorithm to solve. After we found that SAS macro supports the recursive algorithm, we wrote a few lines of code (please see Appendix) to solve the problem.

Although the solution looks simple, the results were proven to be very useful for large SAS reporting systems. This document / workbook can be used to do the following tasks:

1. Optimized rerun strategy to ensure faster response time for change requests and confidence on reproducible results.
2. Complete understanding vs. one-dimensional understanding of the program dependencies.
3. Identify bad program dependencies in the large SAS reporting systems such as no child dependency and self recursive dependency.
4. No more manual maintenance of an important document.
5. Better project management including easier knowledge transfer among team members.

Hopefully you can creatively use the power of recursive SAS macro in your system, and enjoy the wonderful life of automation.

REFERENCES


ACKNOWLEDGEMENTS
This paper is a compilation of the work and experience of sanofi-aventis IS Reporting and Analytics group, including Patrick Dunleavy, Vinod Narayanan, Anuj Sharma, myself and others; as well as years of collaboration with the SAS institute. The authors would like to thank continuous encouragement and guidance of Ashinoff, Craig VP, IS Sales & Marketing Ops and other members of the team for their comments and support.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Name Jim Fang
Enterprise sanofi aventis
Address 55 Corporate Drive
City, State ZIP Bridgewater, NJ 08807
Work Phone: (908)981-6080
E-mail: Jim.Fang@sanofi-aventis.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.
APPENDIX

Recursive macros used to search complete program dependencies.

/*****************************************
Program Dependency Recursive Search macro
*****************************************/
%macro DrSearch (data_in=, data_out=);
data iologic;
 set &data_in;
   prog_orig=prog_out;
   prog_out=prog_in;
run;
proc sql;
 create table iologic_new as
 select a.prog_in, a.program_in, b.prog_orig as prog_out, b.program_out
 from io.iologic_level1 a, iologic b
 where a.prog_out=b.prog_out;
quit;
proc sort nodupkey data=iologic_new;
  by prog_out prog_in;
run;
/****************************************************
Utility macro presented in the SUGI 31 paper 251-31
Demystifying the SAS Macro Facility - by Example
Harry Droogendyk, Stratia Consulting Inc.
Marje Fecht, Prowerk Consulting Ltd
****************************************************/
%if %attrn(iologic_new,nobs)>0 %then %do;
data iologic_final;
 set &data_out iologic_new;
run;
%DrSearch(data_in=iologic_new, data_out=iologic_final)
%end;
%else %do;
 proc sort nodupkey data=iologic_final out=io.iologic_final;
  by prog_out prog_in;
run;
%end;
%mend DrSearch;