Using SAS ® PROC SQL to Support a Build Combinations Tool to Support Modularity

Stephen Sloan, Accenture

ABSTRACT

With SAS ® PROC SQL we can use a combination of a manufacturing Bill of Materials and a sales specification document to calculate the total number of configurations of a product that are potentially available for sale. This will allow the organization to increase modularity and reduce cost while focusing on the most profitable configurations with maximum efficiency. Since some options might require or preclude other options, the result is more complex than a straight multiplication of the numbers of available options. Through judicious use of PROC SQL, we can maintain accuracy while reducing the time, space, and complexity involved in the calculations.

INTRODUCTION

A build combination of a product is defined as a saleable unit that is designed and manufactured per the customer's selection of available options.

The total number of build combinations is a simple indication of overall product complexity. It can be used to compare the complexity of similar products offered for sale by competitors.

To calculate the number of build combinations is not as simple as multiplying the number of available options.

For example, a laptop computer available with 3 options of memory and 2 options of wireless card will result in a total of (3*2) = 6 build combinations. However, if a customer is given a choice of not to opt for the wireless, there are actually (3*3) = 9 build combinations. Furthermore, if the manufacturer adds an exclusion that one of the memory options is not available with a particular wireless card, the number of build combinations decreases to 8. Additionally, if the laptop comes with store options like an extra battery, a mouse, a webcam and a protection screen and the customer can choose any, all, or none of them, there are now $2^4 = 16$ new combinations, to be multiplied by the total of the other combinations.

THE PROBLEM WITH TOO MANY OPTIONS

As new options are added to an available product, the number of build combinations for that product increases exponentially over time. In addition, if there are more requirements or restrictions placed on combinations, the complexity of the calculation also increases. It can be cumbersome to manually calculate or keep track of the total number of build combinations. The increasing number of options and complexity of calculations can lead to a considerable amount of calculation time and difficulty keeping track of available options. From a retail point of view, this causes a dealer to make one of the following choices:

- Keep a large inventory with all available configurations, increasing the cost of inventory and the risk of unsold items.
- Keep a large inventory of all available parts, leading to a large cost of inventory and delay in assembly
- Order the finished product with the specified options, leading to delay in closing the sale.

A build combinations analyzer with the goal of modularity will improve this situation.

PROGRAM LOGIC FOR THE BUILD ANALYZER

The purpose of the analyzer is to calculate the number of possible combinations of options for a given item. The item is configured from a number of different categories of options. See Figure 1.

For example, a computer might have a monitor, memory, keyboard, mouse, wireless card, among other options.

Each of these would be either mandatory, meaning that at least one must be chosen (for example, a monitor) or optional, meaning that the customer could buy the product without the option (for example, wireless card). The customer would only choose one option for categories like monitor, while the customer could choose many options for categories like installed software.

After the above information is recorded, the program then looks at the exclusions and requirements that have been input.

For example, a printer could require a printer cable and SAS ® Graph requires SAS ® Base.

The program logic is as follows:

• Establish macros and environmental variables.

• Read an input spread sheet with the information about the options.

• Organize the information from the spread sheet into categories.

• Use the information on the spread sheet to determine the number of options in each category.

• Get the Cartesian product of the combinations of items in each category.

• Use the information on the spread sheet to process exclusions and requirements. This will reduce the number of possible combinations.

• Produce the report.

NOTE: When there are large numbers of possible categories, the program optimizes the space and time requirements by creating the Cartesian products for subsets of the total number of categories, and then multiplying the counts from the different subsets.

* A sample SAS ® program is available with this document.

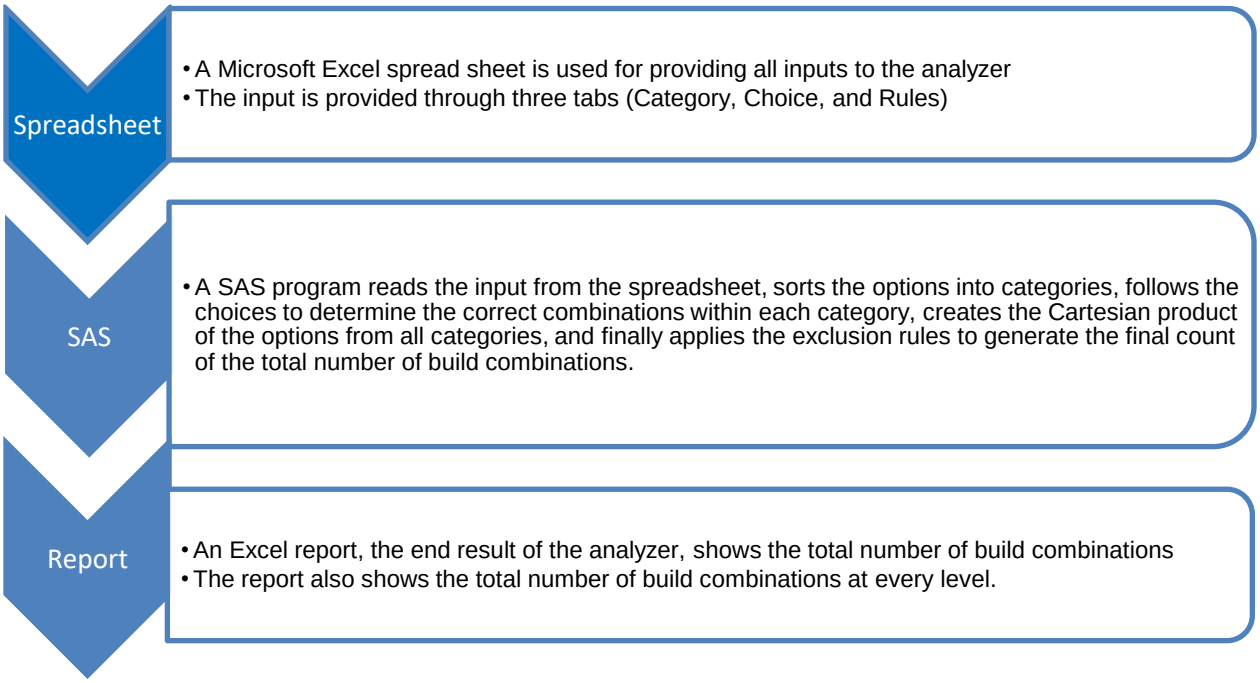

Figure 1. The process of producing the report from the build combinations analyzer.

The build analyzer contains a number of benefits when compared with doing the analysis manually. See Figure 2 for a comparison. In one test case, the analyzer saved over 90% of the CPU time compared with other methods. There were trillions of possible combinations and the analyzer calculated the amount of combinations in a few minutes.

MANUAL METHOD	BUILD COMBINATION CALCULATOR
Very cumbersome	Programmable
Inaccurate	Accurate
Complicated (Difficult to monitor progress)	Simple to use (Progress can be tracked at every option)
Incapable of handling large number of exclusion rules	Capable of including large number of rules
Any change causes the calculation to rstart	Can accommodate changes at any level

Figure 2. Using the build combinations analyzer compared with performing the analysis manually

CONCLUSION

Using a build combinations analyzer created in SAS ® can help manufacturers and retailers increase the modularity of their product offering, test the impact of new options and rules, and enable organizations to reduce inventory expenses while decreasing the amount of time necessary for order fulfilment. The analyzer can then consider the standard cost of different configurations in conjunction with the profitability and volume of sales to choose the appropriate level of modularity.

CONTACT INFORMATION

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

Name: Stephen Sloan

Enterprise: Accenture

Title: Data Science Senior Principal

Work Phone: 917-375-2937

E-mail: Stephen.b.sloan@accenture.com

SAS ® CODE

```
*** This is a sample program showing how to take a spread sheet identifying
categories, options, and exclusion rules, and produce a report showing the
combinations ***;
*** With a larger data set the counting of options will be done in a PROC SQL, so that
exclusions can be more easily processed ***;

*** Identify libraries and options ***;

libname dat 'C:\Sample_Data';

libname big 'H:\SASFILES';

options DQUOTE MSGLEVEL=I COMPRESS=YES;

*** Macro to create data set ***;

%macro ds(name);
if upcase(category)=upcase("&name") then do;
      output &name.;
      if first.category and put(category,$req.)='O' and put(category,$amt.)='1' then
do;
         option_code='  ';
         output &name.;
      end;
 end;
%mend;

*** Macro to count entries in data set ***;
%macro count(ds);
proc summary n nway data=&ds;
output out=&ds._count(rename=_freq_=&ds._count drop=_type_ pointobs=YES);
run;
%mend;

  *** Read in the tabs from the Excel sheet ***;
proc import datafile='C:\Sample_Data\Sample.xlsx' out=parts replace dbms=excel;
sheet='Choice';
run;

proc import datafile='C:\Sample_Data\Sample.xlsx'
out=category(rename=(category=start Mandatory__M____or_Optional__O_=label)
where=(start^=' ')) replace dbms=excel;
sheet='Category';
run;

proc import datafile='C:\Sample_Data\Sample.xlsx' out=rules replace dbms=excel;
sheet='Rules';
run;

*** Create formats to look up category types and option names ***;
data category;
     set category;
       start=compress(start);
       fmtname='$req';
run;

proc format cntlin=category;
run;
```

```
data category;
     set category;
       label=Number_that__can_be_chosen___or_;
       fmtname='$amt';
run;

proc format cntlin=category;
run;

data parts;
     set parts;
       category=compress(category);
run;

proc sort data=parts;
     by category option_code;
run;

data partsfmt;
     set parts(rename=(option_code=start Option_name=label));
       fmtname='$partnm';
run;

proc format cntlin=partsfmt;
run;

*** The data step below takes the category names and creates data sets ***;
data FieldAttachments Memory Wireless;
       set parts(keep=category option_code);
       by category;
       %ds(FieldAttachments);
       %ds(Memory);
       %ds(Wireless);
run;

*** The macro calls and data set names use the data sets created above to generate
macro variables with counts ***;
       %count(FieldAttachments);
       %count(Memory);
       %count(Wireless);
run;

data _null_;
     merge fieldAttachments_count
       wireless_count
       memory_count;
       call symput('fieldattachments_count',fieldattachments_count);
       call symput('memory_count',memory_count);
       call symput('wireless_count',wireless_count);
run;

*** The steps below create data sets for those categories which can have multiple
selections, not just a single selection.  ***;
%macro getcomb(num,tot,item);
        do i=1 to &tot;
           &item.b{i}=&item{i};
        end;
       do i=1 to comb(&tot,&num);
            rc=allcomb(i,&num,of &item{*});
              do j=1 to &num;
                 &item.b{j}=&item{j};
          end;
              %if &num ne &tot %then %do;
```

```
            do k=&num.+1 to &tot;
               &item.b{k}=' ';
            end;
            %end;
            output;
        end;
%mend;


data fieldattachments2;
     set fieldattachments end=eof;
       array fieldattachments{4} $ 8 fieldattachments1-fieldattachments4;
       if _n_=1 then do i=1 to 4;
          fieldattachments{i}=' ';
       end;
       fieldattachments{_n_}=option_code;
     if eof;
retain fieldattachments1-fieldattachments4;
keep fieldattachments1-fieldattachments4;
run;

data fieldattachments3;
     set fieldattachments2;
       array fieldattachments{4} $ fieldattachments1-fieldattachments4;
       array fieldattachmentsb{4} $ 8 fieldattachmentsb1-fieldattachmentsb4;
       do ii=1 to 4;
            %getcomb(ii,4,fieldattachments);
       end;
       do i=1 to 4;
          fieldattachmentsb{i}=' ';
       end;
       output;
keep fieldattachmentsb1-fieldattachmentsb4;
run;

%count(fieldattachments3);

data _null_;
     merge fieldattachments3_count;
       call symput('fieldattachments3_count',fieldattachments3_count);
run;

*** Put together the group with exclusions ***;
data final(drop=category);
            do i=1 to &fieldattachments3_count;
               set fieldattachments3 point=i;
               do j=1 to &wireless_count;
                  set wireless(rename=option_code=wireless) point=j;
                    do k=1 to &memory_count;
                         set memory(rename=option_code=memory) point=k;
                         output;
                     end;
                  end;
            end;
stop;
run;

*** Process exclusion from spread sheet ***;

data final2;
     set final;
       if memory='Sung1T' and wireless='HPWR' then delete;
run;
```

```
**** Create output data set, only if there is a small enough total to fit the
information into a spread sheet ***;
proc export data=final2 outfile='C:\Sample_Data\Report.xlsx' replace dbms='excel';
sheet='Sample report';
run;
```