PharmaSUG 2010 - AD25

# Manual to Automatic: Changing Your Program's Transmission

Arthur L. Carpenter
California Occidental Consultants

## ABSTRACT

You have successfully created a SAS® program that processes data and generates reports for a series items in the data. The program itself works fine, but the problem is that each time the program is executed, it must be manually modified specifically for that run. Perhaps the analysis variable changes or perhaps the name of the report's PDF file must reflect the name of the incoming data set. Making manual changes once in awhile is bothersome, but not onerous. But if find yourself making a great many of these manual changes, the process becomes both tedious and error prone.

Through the use of the macro language, SAS has the power make the necessary changes on the fly, automatically, without manual intervention. Learning how to convert a program from one that needs manual modifications to one that adjusts and runs automatically is not difficult. Not difficult that is, if you know the steps. This paper walks you through the steps, that you can follow, to convert your program's transmission from manual to automatic.

## KEY WORDS

macro language, automated program, dynamic process, data driven

## INTRODUCTION

The ability to creating an automated program or process is learned. You will not absorb it at night while you sleep, you can not ingest it with your Cheerios™ at breakfast, and you will not stumble on it. While the bottom line is that it is not intuitively obvious, it is not particularly difficult to master.

The key to automating a SAS program is the ability to build and process a list of activities. In the paper "List Processing Basics: Creating and Using Lists of Macro Variables" by Ron Fehd and Art Carpenter, four different techniques of list creation and management are discussed. The "List Processing Basics" paper and the one that you are currently reading, cover much of the same material, however the "List Processing Basics" paper concentrates on the process of list building, rather than explicitly discussing the overall process of program automation. After finishing this paper, I highly recommend that you read the "List Processing Basics" paper for numerous details that will not be covered here.

In this paper I will attempt a more simplified approach to the topic from a different direction.

## MAKING A LIST

Before you can successfully automate your program, you will need to be able to define what is to be done in terms of a list. The examples in this paper are for a report that is to be executed for each of a number of different countries. We therefore need to be able to build that list of countries so that the program will

automatically execute for each country on the list.

What do I mean by a 'list'? Well actually, nothing special at all. You have created numerous lists of things to do and to buy, and the lists that will be created in the automation process are only different in that they utilize the macro language.

Most of us create lists for our errands. If we are to buy groceries, we may create a list such as one of the following two lists.

*eggs*
*butter*
*milk*
*cheerios*

A list of vertical values is probably most common, however there is no reason why the list could not have been written horizontally as well. Both work equally well as a shopping list, and both have application to the automation process.

*eggs butter milk cheerios*

The lists that we use in our SAS program will be stored through the use of the macro language and macro variables. A vertical list will be stored using a series of macro variables each containing one item, and the horizontal list is stored in a single macro variable.

**%let item1 =** *eggs;*
**%let item2 =** *butter;*
**%let item3 =** *milk;*
**%let item4 =** *cheerios;*

**%let itemlist =** *eggs butter milk cheerios*;

Of course if we have to build each macro variable by hand we will not have saved ourselves much work - still too manual. So lets look at a couple of ways that SAS can build macro variable lists. Remember there are other ways to create lists, other ways than just the two shown here.

For the automated process the information that is to be used to generate the list has to come from somewhere. In the examples shown here and below, the list comes from the data, but there are many places that you can find information to feed the automation process (see the section on Information Sources later in this paper for more on this topic).

## Creating a List of Variables - The Vertical List

The 'vertical list' is made up of one macro variable for each item that is to be stored. The list can easily be created in a DATA step or in a PROC SQL step.

```
data list;
   buy='eggs    '; output list;
   buy='butter  '; output list;
   buy='milk    '; output list;
   buy='cheerios'; output list;
   run;
```

First let's build a data set which contains our list (one observation per item). The variable BUY contains the item of interest, and it is this value that we want to place into the macro variable.

The list of macro variables can be created in either a DATA step or a PROC SQL step.

```
data _null_;
   set list; ❶
   i = left(put(_n_,4.)); ❷
   call symputx('item'||i,buy); ❸
   call symputx('numitems',i); ❹
   run;
```

```
proc sql noprint;
   select buy
      into :item1 - :item999 ❸
         from list; ❶
   quit;
%let numitems = &sqlobs; ❹
```

❶ Read the items one at a time from the incoming data set containing the shopping list.
❷ Create a character variable that contains the item number.
❸ Store the individual items to buy in a macro variable of the form $ITEM_i$.
❹ Save the number of items in the list in the macro variable &NUMITEMS. &SQLOBS is an automatic counter generated during processing of the SQL step.

In the DATA step we use the SYMPUTX routine to write the macro variables to the macro symbol table, while in the SQL step the INTO phrase (along with a colon in front of the new macro variable) is used.

### Creating a Variable with a List - The Horizontal List

A horizontal list is best stored in a single macro variable. In this form of the list the individual items are stored as words within the list. This implies that there needs to be a word separator that is distinct from the values being stored (a comma will not work if an item contains a comma).

```
proc sql noprint;
  select buy
    into :itemlist separated by '' ❺
      from list;
  quit;
```

While the DATA step can be used to build this type of list, it is much easier to do it in a SQL step. The key is the SEPARATED BY ❺ phrase. This allows SQL to enter multiple values into the macro variable &ITEMLIST, with the individual words separated by blanks.

## USING A MACRO VARIABLE LIST

Now that we have built either a vertical or horizontal list of values, we need to be able to use them. In our program. For our shopping list analogy we wrote the list on a piece of paper so we could go shopping, and once at the store we read the list. In our program there are multiple ways to read our list, but usually it will involve the use of a macro %DO loop of one kind or another.

The macro language does not support arrays, there is no %ARRAY statement, however this type of processing is similar to what is done with a singly dimensioned array. The list is a vector of values, which is accessed one value at a time.

### Processing a List of Macro Variables - Vertical List

Vertical lists are generally processed with an iterative %DO loop. The loop's index also serves as the index to the list of values. Since a %DO can not appear in open code it will be written inside of a macro. In this example we are only using a %PUT to write to the LOG ❹.

```
%macro writeVertical;
%local i; ❶
%* Write the vertical list;
%do i = 1 %to &numitems; ❷
  %put item &i is &&item&i; ❸
%end;
%mend writevertical;
%writevertical
```

```
132  %writevertical ❹
item 1 is eggs
item 2 is butter
item 3 is milk
item 4 is cheerios
```

❶ Be sure to force the %DO loop index onto the local symbol table unless you have a really  really good reason for not doing so (Carpenter, 2005).
❷ The iterative %DO ranges from 1 to the number of items in the list (&numitems).
❸ The %PUT will be executed for each item on the shopping list.  Double ampersands, &&, resolve to a single ampersand.  It takes two passes for &&item&i to be resolved to the shopping list item.

| &i | &&item&i becomes | list item |
|---|---|---|
| 1 | &item1 | eggs |
| 2 | &item2 | butter |
| 3 | &item3 | milk |
| 4 | &item4 | cheerios |

❹ The LOG shows the four items from the shopping list.

## Processing a Macro Variable Containing a List - Horizontal List

When you use a horizontal list, you will need fewer ampersands than for the vertical list, but the code to extract the individual words will be more complex.

We will still be using a %DO to step through the list of values however this time we will also need to get the individual words from the overall list of words.  The %SCAN function is designed to extract the $i^{th}$ word and will be perfect for the task.  When we know the total number of items in the list an iterative %DO can be used as in the previous example.  However often when dealing with a horizontal list we often do not know how many items it contains, and a %DO %WHILE is used instead of a iterative %DO.  Just for demonstration purposes, the following example uses the slightly more complex %DO %WHILE.

```
%macro writehorizontal;
%local i item;
%* Write the horizontal list;
%let i=0; ❻
%do %while(%qscan(&itemlist,&i+1,%str( )) ❼ ne %str()); ❽
  %let i = %eval(&i+1); ❾
  %let item = %qscan(&itemlist,&i,%str( )); ❿
  %put Item &i is &item;
%end;
%mend writehorizontal;
%writehorizontal
```

The LOG shows:

```
143  %writehorizontal
Item 1 is eggs
Item 2 is butter
Item 3 is milk
Item 4 is cheerios
```

❻ The word counter is initialized to 0.
❼ %QSCAN is used to check to see if there is another word. The %STR( ) function is used to designate a blank as the word delimiter.
❽ Continue the loop for long as the extracted word is not null.
❾ The current word exists, increment the word counter by one.
❿ The current word ($i^{th}$ word) is extracted and loaded into &ITEM.

## THE 6 STEP PROGRAM

Most programs that build and process a list have certain elements. Initially, as you learn to create programs that run themselves automatically, make a conscious effort to make sure that these elements exist in your program. They may not be intuitive as you learn, however as you get proficient building this type of program the elements themselves become automatic.

1) Identify a source for the information that will drive the process.
2) Generalize or write your program to accept a macro variable list.
3) Use a %DO loop to process across all the items in the list.
4) Within the loop, access the individual elements of the list by using either:

| | |
|---|---|
| &&item&i | Vertical list (one macro variable per item) |
| %qscan(&itemlist,&i,%str( )) | Horizontal list (one macro variable - each word is an item) |

5) Build a list of macro variables (or a macro variable containing the list). Vertical or horizontal list, its your choice.
6 )For vertical lists (a list of macro variables), be sure to count the number of elements in the list.

It is common when first learning to build an automatic program to want to convert it from one that runs manually. As you make the conversion be sure that you account for each of the above steps.

## CONVERSION

Once you are comfortable with dynamic programming techniques, you will be able to build programs that execute automatically from scratch. However for most programmers it is easiest to see how to apply the six steps shown above, by starting with a manual program and converting it step by step. The following PROC REPORT step is taken from the paper by Fehd and Carpenter (2007), and it will be used here to illustrate the step by step process for the conversion from manual to automatic.

```
title1 'shoe data';
proc print
data=sashelp.shoes(obs=30);
   var region product subsidiary
sales;
   run;
```

In this example we will examine the shoe sales data in SASHELP. The data set itself is fairly simple with sales information for

various products from various regions of the world and subsidiaries within region.

shoe data

| Obs | Region | Product | Subsidiary | Sales |
|---|---|---|---|---|
| 1 | Africa | Boot | Addis Ababa | $29,761 |
| 2 | Africa | Men's Casual | Addis Ababa | $67,242 |
| 3 | Africa | Men's Dress | Addis Ababa | $76,793 |
| 4 | Africa | Sandal | Addis Ababa | $62,819 |
| 5 | Africa | Slipper | Addis Ababa | $68,641 |
| 6 | Africa | Sport Shoe | Addis Ababa | $1,690 |
| 7 | Africa | Women's Casual | Addis Ababa | $51,541 |
| 8 | Africa | Women's Dress | Addis Ababa | $108,942 |

*. . . . portions of the listing is not shown . . . .*

## 1) Determine the source of information that will drive the automation process

In this example we would like to execute a PROC REPORT step for each REGION in the data set.

```
Title "Sales in Africa";

PROC Report data = sashelp.Shoes nowd;
   where region = 'Africa';
   column product subsidiary, sales;
   define product / group;
   define subsidiary / across;
   define sales / analysis '';
   run;
```

Before we can determine the source of information that we are going to use to drive the automation process, we need to ask *what* is it that needs to be automated? In the code to the left we want to execute the same step for each value of REGION. These are the items that will become macro variables and these are the items for which we need to find a source. Since the manual operation is for each region. and since regions are in the analysis data set, the analysis data itself will become the source for our control information.

## 2) Generalize or write your program to accept a macro variable list.

We know that the incoming data will supply the information that will control the process (the various values of REGION), however we also need to generalize the program so that it can accept these values.

```
%let reg = Africa;

Title "Sales in &reg";

PROC Report data = sashelp.Shoes nowd;
   where region = "&reg";
   column product subsidiary, sales;
   define product / group;
   define subsidiary / across;
   define sales / analysis '';
   run;
```

Let's look at the generalization process in stages.

If we were only going to make the program slightly less manual, we might convert those things that change into simple macro variables. By using a %LET statement we would be able

to write the REPORT step so that it could execute for any region.

Clearly we still need to manually change the value of &REG for each region, however now we only need to make the change in one place.

```
%let reg = Africa;
%*let reg = Asia;
%*let reg = Canada;

Title "Sales in &reg";

PROC Report data = sashelp.Shoes nowd;
   where region = "&reg";
   column product subsidiary, sales;
   define product / group;
   define subsidiary / across;
   define sales / analysis '';
   run;
```

As we determine the names of the regions, that is the values of the variable REGION, we can create a %LET statement for each value. At any given time the ones that are not to be used are commented out.

Now all we have to do is uncomment the region of choice. Even this process, however is still fairly manual, and we still have to '*know*' all the values that are taken on by the variable REGION. For a large number of values this becomes awkward and unwieldily at best.

As you generalize anticipate what your program will need to look like in order for it to take advantage of the macro variable list.

3) Use a %DO loop to process across all the items in the list.
4) Within the loop, access the individual elements of the list by using &&VAR&I.

```
%do r = 1 %to &regcnt;
   Title "Sales in &&reg&r";

   PROC Report data = sashelp.Shoes nowd;
      where region = "&&reg&r";
      column product subsidiary, sales;
      define product / group;
      define subsidiary / across;
      define sales / analysis '';
      run;
%end;
```

Clearly we want to execute the REPORT step a number of times for a list of regions. A %DO loop is added to surround the repeated step and the values of the variable REGION refer to a list of macro variables of the form: &REG1, &REG2, &reg3, etc.

Because we are addressing a list of macro variables (a vertical list) the list is addressed using &&REG&I.

5) Build a list of macro variables

Because of how we built the loop and referenced the list, we need to build a list of macro variables, each with one value of REGION.

```
proc sql noprint;
   select distinct region
      into :reg1 - :reg999
         from sashelp.shoes;
   quit;
```

This PROC SQL step writes up to 999 macro variables of the form &REG1, &REG2, etc. Each will have a unique value of REGION.

6 )Count the number of elements in the list.

Our iterative %DO loop will need to have an upper bound that matches the number of regions.

```
proc sql noprint;
   select distinct region
      into :reg1 - :reg999
         from sashelp.shoes;
   quit;
%let regcnt = &sqlobs;
```

Since we have used an SQL step to build the macro variable list, we can take advantage of the automatic macro variable &SQLOBS. This macro variable is assigned a value each time an SQL statement such as SELECT is executed. In this example &SQLOBS holds the number of distinct regions.

Because &SQLOBS is reset by most SQL statements, it is always best to save the value in a macro variable of your choosing. Here we are using &REGCNT.

```
%macro SalesRpt;
%local r;
proc sql noprint;
   select distinct region
      into :reg1 - :reg999
         from sashelp.shoes;
   quit;
%let regcnt = &sqlobs;
%put &regcnt;
%do r = 1 %to &regcnt;
   Title "Sales in &&reg&r";

   PROC Report data = sashelp.Shoes nowd;
      where region = "&&reg&r";
      column product subsidiary, sales;
      define product / group;
      define subsidiary / across;
      define sales / analysis '';
      run;
%end;
%mend salesrpt;
```

The resulting program does not require the programmer to know the number of regions or their names. Indeed the number of regions can change, the region names can change, the attributes of the REGION variable can change; none of these changes will require any manual intervention on the part of the programmer. The program has been converted to automatic!

```
%macro SalesRpt;
%local r reg;
proc sql noprint;
   select distinct region
      into :reglist separated by '*'
         from sashelp.shoes;
   quit;

%let r = 0;
%do %while(%qscan(&reglist,&r+1,*) ne );
   %let r = %eval(&r+1);
   %let reg = %qscan(&reglist,&r,*);
   Title "Sales in &reg";

   PROC Report data = sashelp.Shoes nowd;
      where region = "&reg";
      column product subsidiary, sales;
      define product / group;
      define subsidiary / across;
      define sales / analysis '';
      run;
%end;
%mend salesrpt;
```

In the previous macro we built the list of regions by creating a list of macro variables (vertical list). The horizontal list could have also have been implemented by creating a single macro variable with each region stored as a word. The resulting code becomes:

We could have used &SQLOBS to count the regions as we did in the previous example, however sometimes the number of words in a list is unknown. The %DO %WHILE to the

left assumes that the number of regions is unknown.

The individual words are determined one at a time using the %QSCAN function. This simplifies the macro variable call in the TITLE and WHERE statements at the expense of the more complex %QSCAN.

## INFORMATION SOURCES

In each of the previous examples the information needed to build the list of macro variables was obtained from the data being analyzed. Fortunately, since not all of our tasks are driven by the analysis data, there are a number of other sources of information that can be used to build the macro variable lists.

- Control data sets built strictly to control macro operations
- The analysis data itself
- A table that looks like or acts like a data table, this might include something like an EXCEL spreadsheet
- OUT= option with PROC CONTENTS
- SASHELP and Dictionary views
- Results of operating system operations
- SAS system option settings
- SAS DATA step functions

Essentially any data set, function, or option that returns a value can become a source of information for an program that is to operate automatically. There are many examples of macros that take advantage of the sources just listed. You need to become familiar with each of these and you may even discover some more of your own. As you use them, notice that in each case your program will include most, if not all, of the elements of an automated program discussed earlier.

## SUMMARY

Programs that require manual intervention are more costly to maintain and are more error prone than those that automatically adjust to the task. Converting your programs from manual to automatic is not always straightforward, but the benefit can be huge. By following the steps outlined in this paper, you can successfully convert, write, and understand automated programs.

The automated aspects of your programs may be based on the analysis data or on one or more other sources of information. Learn about these sources of information and learn how to use them. Practice.

You too can move up from manual to automatic!

## ABOUT THE AUTHOR

Art Carpenter's publications list includes four books, and numerous papers and posters presented at SUGI, SAS Global Forum, and other user group conferences. Art has been using SAS® since 1977 and has served in various leadership positions in local, regional, national, and international user groups. He is a SAS Certified Advanced Professional Programmer, and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

## AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com




## REFERENCES

*Carpenter's Complete Guide to the SAS® Macro Language, 2nd Ed.*, 2004, by Arthur L. Carpenter, SAS Institute Inc., Cary NC.

Carpenter, Arthur L. and Richard O. Smith, 2000, "Clinical Data Management: Building a Dynamic Application", presented at the Pharmaceutical SAS User Group meetings, PharmaSUG, (May 2000) and published in the conference proceedings; also presented at the 8th Western Users of SAS Software, WUSS, meetings (September, 2000) and published in the conference proceedings. Also presented at the PharmaSUG (June 2009) meetings.

Carpenter, Arthur L., 2004, "Storing and Using a List of Values in a Macro Variable", Proceedings of the 12th Annual Western Users of SAS Software, Inc. Users Group Conference (WUSS), Cary, NC: SAS Institute Inc. Also in the proceedings of the Thirtieth SAS User Group International Conference (SUGI), 2005, Cary, NC: SAS Institute Inc., paper 028-30, and in the proceedings of the Pharmaceutical SAS User Group Conference (PharmaSUG), 2005, Cary, NC: SAS Institute Inc.

Carpenter, Arthur L., 2005, "Make 'em %LOCAL: Avoiding Macro Variable Collisions", proceedings of the Pharmaceutical SAS User Group Conference (PharmaSUG), 2005, Cary, NC: SAS Institute Inc., paper TT04. Also published in the proceedings of the 13th Annual Western Users of SAS Software, Inc. Users Group Conference (WUSS), Cary, NC: SAS Institute Inc., paper SOL_Make_em_local_avoiding.

Fehd, Ronald and Art Carpenter, 2007, The same data set and REPORT step is shown in the paper "List Processing Basics: Creating and Using Lists of Macro Variables" by Ronald Fehd and Art Carpenter which was presented at the 2007 SAS Global Forum (Paper 113-2007). The discussion of the paper looks at different approaches used in the automation of programs by using various kinds of macro variable lists. This paper appears in proceedings of a number of conference, including: SGF(2007), WUSS (2008), MWSUG (2009), SESUG (2009).

Sattler, Jim, 2003, "A Table-Driven Solution for Clinical Data Submission" This paper appears in the proceedings of SUGI 28 as paper 40-28 in the Applications Development section.

## ACKNOWLEDGMENTS

The use of a shopping list as a starting point for this paper was suggested by Ron Fehd.

## TRADEMARK INFORMATION