Streamlining Data-Driven SAS With The %FOR Macro

From sasCommunity

Contents

Streamlining Data-Driven SAS

The %FOR Macro

The %for macro helps create simpler and more consistent data-driven SAS programs. It is patterned after the looping statements in object-oriented languages such as Python and Ruby. The %for macro is object-oriented in that it uses a common syntax to express looping through the elements of completely different kinds of objects.

The %for macro can be downloaded here (http://www.sascommunity.org/wiki/File:For.sas) . It’s best if you store the macro in a SAS macro autocall library, so that it can be used from any SAS program without the need to explicitly include the macro.

We start with simple examples demonstrating how the %for macro works with different kinds of objects. Then, we show solutions to 3 real problems posted on SAS-L. If you are a SAS-L regular, you may have seen some of the other solutions posted.

How It Works

The %for macro repeatedly generates SAS code. A call always has these 3 arguments:

```
%for(list-of-macro-variable-names, in=source-of-loop-values, do=SAS-code-to-generate)
```

The basic idea is that for each loop iteration the %for macro (1) obtains values from the data source, (2) assigns those values to the list of macro variables, (3) substitutes the macro variable values where the variables appear in the SAS code to generate, and (4) outputs (i.e., generates) the modified SAS code.

Simple Examples

The following line of SAS code will be repeated in several examples below using different data sources for the value of the macro variable &many. We'll show the log produced for each example.

```
%for(one, in=&many, do=%nrstr(%put one=&one;))
```

One thing that might look strange in this code is that the do= argument, (the %put statement), is contained in a %nrstr macro quoting function call. This is necessary to defer macro substitution until the %for macro outputs the code. This %nrstr call should be used in every %for macro call.

There are 5 kinds of data sources. We start with a **space-separated value list** data source. This kind of data is identified by enclosing the list in parentheses: ( ).

```
%let many=(hello goodbye);
%for(one, in=&many, do=%nrstr(%put one=&one;))
```

Executing this code produces the following SAS log:

```
one=hello
one=goodbye
```

A **number range** data source is identified by 2 or 3 integers separated by colons. The first number is the starting value, the second is the ending value and the third is the increment (which defaults to 1 if absent). Note that a number range is the only data source not enclosed in some kind of brackets.

```
%let many=1:5;
%for(one, in=&many, do=%nrstr(%put one=&one;))
```

Executing this code produces the following SAS log:

```
one=1
one=2
one=3
one=4
one=5
```

A **dataset** data source is identified by enclosing the dataset name in brackets: [ ]. The dataset name can be qualified with a libname and can include a where clause.

```
data example;
    one = 'interesting!'; two = 'certainly.'; output;
    one = 'useful?';      two = 'not sure.';  output;
run;
```

```
%let many=[example];
%for(one, in=&many, do=%nrstr(%put one=&one;))
```

Executing this code produces the following SAS log:

```
one=interesting!
one=useful?
```

Let’s modify the %for call above to use multiple macro variables.

```
%let many=[example];
%for(one two, in=&many, do=%nrstr(%put one=&one two=&two;))
```

Executing this code produces the following SAS log:

```
one=interesting! two=certainly.
one=useful? two=not sure.
```

Let’s return to our value list data source and try it with this new %for call.

```
%let many=(hello goodbye);
%for(one two, in=&many, do=%nrstr(%put one=&one two=&two;))
```

Executing this code produces the following SAS log:

```
one=hello two=goodbye
```

This example shows that when there are multiple variables in the macro variable list, successive entries of the value list are assigned to the successive macro variables upon each iteration, until the value list is exhausted.

We have seen examples of value list, number range and dataset data sources. For datasets, the names used in the macro variable list must match variable names in the data source dataset. For value lists and number ranges, the names used in the macro variable list are unconstrained – you can use any names.

We describe the remaining two data sources, **dataset contents** and **directory contents**, in the presentation of solutions to problems posted on SAS-L.

SAS-L Problems Solved

The 3 problems below were posted on SAS-L. There were also many solutions posted. The %for macro solutions here are consistently more concise than those posted on SAS-L. Here are the problems with %for macro solutions.

Split dataset sashelp.class into separate datasets by the age variable

This solution uses a **dataset** data source.

```
proc sort data=sashelp.class nodupkey out=ages; by age; run;

data %for(age, in=[ages], do=%nrstr(class_&age(where=(age=&age))));
    set sashelp.class;
run;
```

Thanks to Mike Zdeb for inspiring this solution with his SAS-L posting.

Rename all variables in a dataset with the same prefix (for example, "abc_")

This solution uses a **dataset contents** data source, where the dataset name is enclosed in braces: { }. This data source returns the same information as proc contents. The names which can appear in a dataset contents macro variable list are:

- name – the dataset variable name
- type – set to 1 for numeric variable, 2 for character variable
- format – the variable format
- length – the variable length
- label – the variable label

Here is the solution:

```
%let renames=%for(name, in={some_dataset}, do=%nrstr(&name=abc_&name));
proc datasets;
    modify some_dataset;
    rename &renames;
quit;
```

Note that there is always at least one blank generated between iterations of every %for loop, so it is not necessary to use a blank after the code "&name=abc_&name" in the code above.

Import all spreadsheets in all subfolders of a top folder

This solution uses **directory contents** data sources, where the directory path is enclosed in angle-brackets: < >.

The names that can appear in a directory contents macro variable list are:

- filepath – the complete path of the file, including file name with extension
- filename – the file name, including extension
- shortname – the file name, excluding extension
- extension – the file extension, including period
- isdir – 1 if the file is a directory, 0 if not

This solution also demonstrates nested %for loops!

```
%let topfolderpath= ... path to some directory ... ;
%for(filepath, in=<&topfolderpath>, do=%nrstr(
    %let subfolderpath=&filepath;
    %for(filepath shortname, in=<&subfolderpath>, do=%nrstr(
        proc import datafile="&filepath" out=&shortname
            dbms="excel" replace;
        run;
    ))
))
```

Contact Information

Jim Anderson
Philip R. Lee Institute for Health Policy Studies
UCSF

james.anderson@ucsf.edu

Retrieved from "http://www.sascommunity.org/mwiki/index.php?title=Streamlining_Data-Driven_SAS_With_The_%25FOR_Macro&oldid=18822"
Category: Macro Language

- This page was last modified on 9 October 2009, at 16:52.
- This page has been accessed 11,267 times.