Paper 104-2008

A Macro Function to Parse Any Variable Lists

Jimmy Z. Zou, Michigan State University, East Lansing, MI

ABSTRACT

How to make a user-defined macro work with multiple variables is often a difficult task. In many cases, in order to work with any variable lists just as SAS procedures do, your macro needs to 'recognize' or parse the variable lists. Unfortunately there is no macro or function available to deal with variable lists efficiently.

This paper introduces a macro function %Parse(dsn, varlist) that can parse any variable list (varlist) for a given data set (dsn). The variable list can be a conventional SAS variable list (like those used in SAS procedures and DATA steps) or a Perl regular expression. The function returns a complete list of variable names corresponding to the variable list. With additional Perl regular expressions, this function can parse variable lists or patterns far beyond the conventional ones used in SAS procedures and DATA steps. This macro function provides a useful tool for user-defined macros to handle variable lists.

INTRODUCTION

Variable lists provide a quick way to reference multiple variables. SAS allows the following variable lists to be used in DATA steps and procedures:

Table 1. SAS Variable Lists

Variable List	Example	Includes
Individual variables	cd s	cd s
Numbered range	a1-a30 (or a01-a30)	a1, a2, …, a30 (a01, a02, …, a30)
Name range	t--f	variables from t to f inclusive (based on variable positions in the data set)
Name prefix	xy:	variables with prefix xy
Special name list	_ALL_ _NUMERIC_ _CHARACTER_	All variables All numeric variables All character variables
Mixed (name range & special name)	r-numeric-z r-character-z	All numeric variables from r to z All character variables from r to z

A variable list may include one or more of the above variable lists separated by space. For the purpose of discussion, these variable lists will be referred to as conventional SAS variable lists in this paper. Variable lists are often used in procedures and DATA steps. But in many situations variable lists can not be used directly in macros without listing the variables. Therefore, it will be very useful to create a macro that can parse any variable list for a given data set. Then any macro or program that wants to parse a variable list can call this macro to do the job.

Bramely has developed a macro function to generate a unique variable name list based on user-specified variable patterns using SAS regular expression language (Bramely, 2002). But it can not parse the numbered range variable lists (e.g., a1-a30) or name range variable lists (e.g., t--f) because the range of numbers is not supported in SAS regular expressions or SAS Perl regular expressions (Note: a name range variable list is essentially based on a range of position numbers of the variables.). Another limitation of using this macro is that you have to know SAS regular expressions to use it.

MACRO FUNCTION %Parse

A macro function %Parse is created to parse any variable list for a given SAS data set. The variable list can be a conventional SAS variable list (as defined in Table 1) or a Perl regular expression (SAS Institute, 2006). You don't need to know Perl regular expressions to use this macro function to parse conventional SAS variable lists. Meanwhile, the integration of Perl regular expressions into macro %Parse, inspired by Bramely's work, gives macro function %Parse a greater capability of parsing a wide variety of variable patterns. The syntax of this macro function is as follows.

Syntax of Macro Function %Parse

Syntax
%Parse(dsn, varlist)

dsn –name of the input data set.

varlist - a variable list to be parsed, in either one of the following two forms:

(1) A conventional SAS variable list like those used in SAS procedures and DATA steps. For example, varlist = cd b03-b50 t -- f xy: _NUMERIC_ r-character-z.

(2) A Perl regular expression using / / as delimiter. For example, varlist = /^xy/.

Note: the two forms are not allowed to be mixed together in the same variable list.

Return
The function returns a complete list of variable names corresponding to the variable list (varlist). If an error occurs, the function returns a missing value (null character).

EXAMPLES

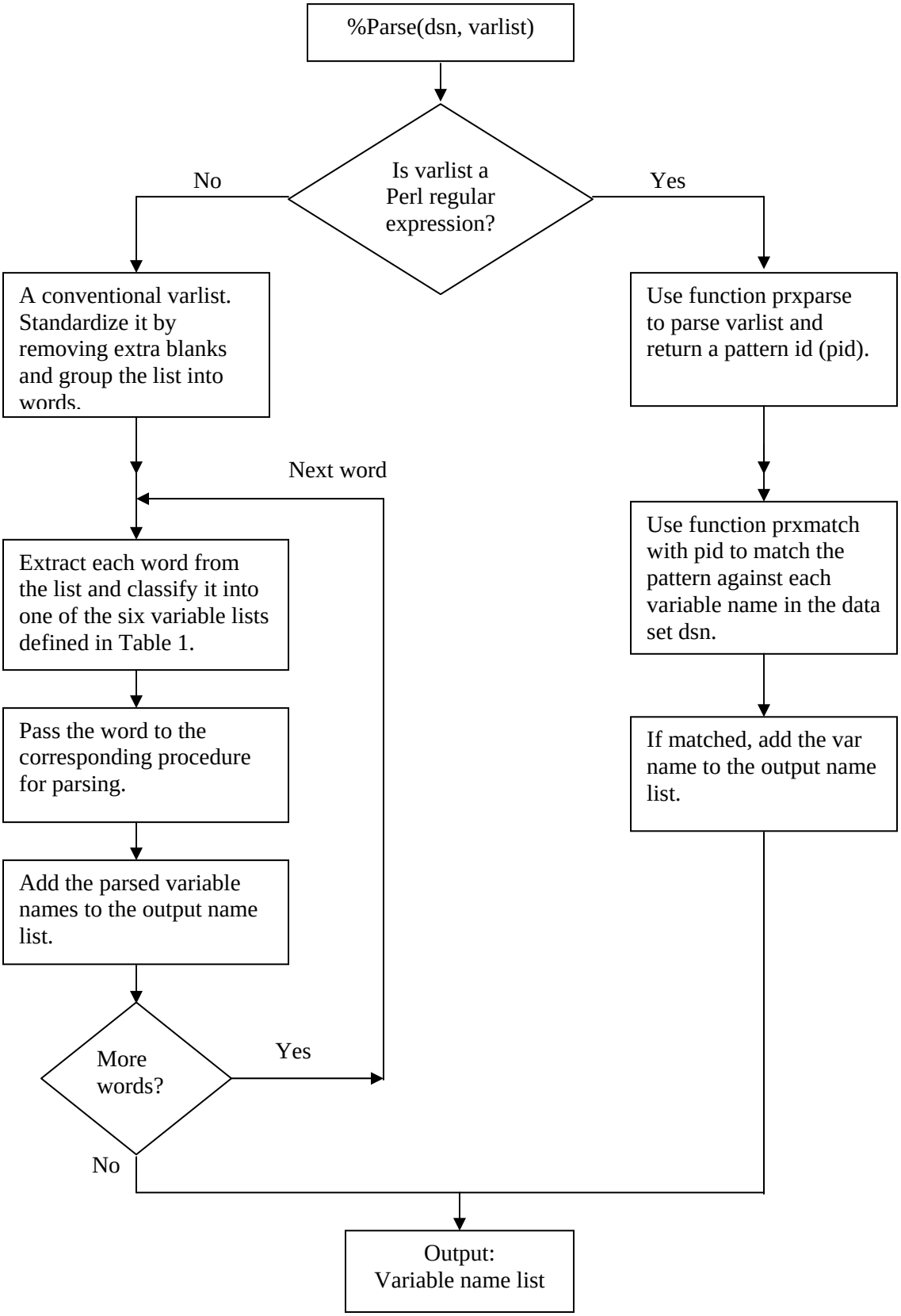
Below are some examples of using macro function %Parse.

```
data Example;
       length name $10 sex $1;
       length ID age month1-month5 b001-b020 8.;
       length State region ck1-ck5 $2;
run;

* Parse conventional variable lists;
%let varlist=month3-month5 b006-b010 name region--ck2 s:;
%put The variable names in varlist are: %Parse(Example, &varlist);
%put All variables in the data set are: %Parse(Example, _ALL_);
%put Char vars between age and region: %Parse(Example, age-character-region);

* Parse Perl regular expressions;
%put The vars starting with s (case insensitive) are: %Parse(Example, /^s/i);
%put The variables ending with e are: %Parse(Example, /e$/);
%put The variables containing a digit 2 are: %Parse(Example, /2/);
%put The variables with two consecutive digits are: %Parse(Example, /\d\d/);
%put Variable names with length of 3 are: %Parse(Example, /^.{3}$/);
```

CODING

The implementation of macro function %Parse employs SAS file I/O functions, character functions, and Perl regular expressions functions. The general approach is illustrated in the following flow chart (see Chart 1). Please see the attached source code at the end of this paper for details.

Chart 1. Coding Flow Chart for %Parse



The code for macro %Parse is a bit long and complicated. There are actually easier ways to parse conventional SAS variable lists than the methods used in function %Parse, but with limitations. In the following I will discuss these simplified versions of macro %Parse.

SIMPLIFIED VERSIONS OF %Parse

Actually these simplified macros do not parse variable lists directly. They run a PROC step or DATA step with the keep option to take care of the variable list. Because regular expressions can not be used in a procedure or DATA step, these macros can only work with the conventional variable lists. In addition, these macros cannot be coded as functions because PROC steps or DATA steps are involved. These macros have syntax similar to that of %Pase, except that a parameter “namelist” is added to specify a global variable to hold the list of names generated by the macro. The first simplified macro is defined as follows:

```
%macro Parse1(dsn, varlist, namelist);
	data _subset_; set &dsn(obs=0 keep=&varlist); run;

	%local names nn i;
	%let dsid=%sysfunc(open(_subset_));
	%let nn=%sysfunc(attrn(&dsid, nvars));
	%if &dsid %then %do i=1 %to &nn;
		%let names=&names %sysfunc(varname(&dsid, &i));
	%end;
	%let rc= %sysfunc(close(&dsid));

	%global &namelist;
	%let &namelist=&names;
%mend Parse1;
```

This macro uses the KEEP=&varlist data set option in a DATA step to create a data set _subset_ that only contains the variables specified in the varlist. The OBS=0 option is used to create an empty data set with no observations. Then file I/O functions are used to generate a list of all the variables in _subset_. Below is the second simplified macro:

```
%macro Parse2(dsn, varlist, namelist);
	proc contents data=&dsn(obs=0 keep=&varlist) out=_tmp_(keep=name)
		noprint;
	run;

	%local nn;
	%global &namelist;

	proc sql noprint;
		select name into
			:&namelist separated by ' '
		from _tmp_;
	quit;
%mend Parse2;
```

The implementation for %Parse2 is very simple. First PROC CONTENTS is called to save the variable names as character values under variable “name” in table _tmp_. The KEEP=&varlist option only keeps the variables specified by the varlist. Then PROC SQL is called to save the variable names into a global macro variable &namelist. The third simplified macro is as follows:

```
%macro Parse3(dsn, varlist, namelist);
	proc transpose data=&dsn(obs=0) out=_tmp_(keep=_name_);
		var &varlist;
	run;

	/* Get the length of dsn records */
	%local dsid rlen nn;
	%let dsid=%sysfunc(open(&dsn));
	%let rlen=%sysfunc(attrn(&dsid, LRECL));
```

```
        /* Concatenate names and save them to macro variable names */
        data _null_; set _tmp_ end=_last_;
               length _cat_ $&rlen;
               retain _cat_ '';
               _cat_ = strip(_cat_) || " " || _name_;
               if _last_ then do;
                      call symputx('names', _cat_);
               end;
        run;

        %global &namelist;
        %let &namelist=&names;
%mend Parse3;
```

This macro calls PROC TRANSPOSE with the VAR statement to select the variables and save their names as character values of variable _name_ in an output table _tmp_. Then it uses a data step to concatenate the variable names as character values of _cat_. Call SYMPUTX routine at the end of the DATA step to save the complete name list to macro variable &names.

CONCLUSION

This paper introduces a macro function %Parse(dsn, varlist) that can parse any variable list and Perl regular expression (varlist) for a given data set (dsn). In addition, three simplified versions of the macro %Parse are also introduced to parse conventional variable lists.

REFERENCES

Bramley, Michael P.D. 2002. “Combining Pattern-Matching and File I/O Functions: A SAS Macro to Generate a Unique Variable Name List”, *Proceedings of the Twenty Seventh Annual SAS Users Group International Conference, Paper 37*, Orlando , Florida, 2002.

Friedl, Jeffrey E. F. 2002. *Mastering Regular Expressions*, Second Edition, O’Reilly, 2002.

SAS Institute Inc. 2006. “Functions and Call Routines: Pattern Matching Using SAS Regular Expressions (RX) and Perl Regular Expressions (PRX)”, *SAS Language Reference: Dictionary, SAS OnlineDoc 9.1.3*.

ACKNOWLEDGEMENTS

I would like to thank my colleagues Dr. Richard T. Houang and Dr. Neelam Kher at Michigan State University for their careful reviews of this paper and valuable comments.

CONTACT INFORMATION

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

Jimmy Z. Zou
Michigan State University
240A Erickson Hall
East Lansing, MI 48824
Work Phone: 517-432-9907
E-mail: zouzhiwe@msu.edu

APPENDIX: SOURCE CODE OF MACRO FUNCTION %Parse

```
/*************************************************************************************
MACRO FUNCTION: %Parse
AUTHOR: Jimmy Z. Zou
CREATED: 7/1/2003
REVISED: 5/30/2006 to make it work with Perl regular expressions.

DESCRIPTION:
This macro function is used to parse a variable list for any given SAS data set. The
variable list is either a conventional SAS variable list (like those used in data
steps and proc steps) or a Perl regular expression. The function returns a complete
list of variable names corresponding to the variable list. If an error occurs, the
function returns a missing value (null character). Optionally, the function saves the
number of variables in the list to a global variable specified by the user.

SYNTAX:
%Parse(dsn, varlist<,nvars>)

dsn - a SAS data set name.
varlist - a variable list to be parsed, in either one of the following two forms:
      (1) A conventional SAS variable list such as
            varlist = cd  b03-b50  t -- f  xy: _NUMERIC_  r-character-z
      (2) A SAS Perl regular expression using / / as delimiter such as /^xy/.
nvars - optional parameter to specify the name of a global variable to hold the number
of variables returned.
*************************************************************************************/

%Macro Parse(dsn, varlist, nvars);
%local i j k _n_ word upword count d p p1 p2 name name1 name2 suffix1 suffix2 prefix
namelist;

%let dsid=%sysfunc(open(&dsn));

%if not &dsid %then %do;
      %put %sysfunc(sysmsg());
      %goto Exit;
%end;

/* Get the total number of variables in dsn */
%let _n_=%sysfunc(attrn(&dsid, nvars));
%let count=0;

/* If varlist is not a Perl regular expression... */
%if not %index(&varlist, /) %then %do;
      /*
      Standardize the varlist:
      Removing extra blanks in varlist and group the variables into words.
      */
      /* %let varlist=%cmpres(&varlist);*/
      %let varlist=%sysfunc(compbl(&varlist));
      %let varlist=%sysfunc(tranwrd(&varlist, %str( )-, -));
      %let varlist=%sysfunc(tranwrd(&varlist, -%str( ), -));

      /*
      Divide and Conquer:
      Set up a loop to extract the words in varlist one by one.
      Then parse each word to get the variable names.
      */
      %let i=1;
      %do %until (%qscan(&varlist, &i, %str( ))=%str());
            %let word=%qscan(&varlist, &i, %str( ));
```

```
%let upword=%upcase(&word);
%let p=%index(&word, --);

/* Parse a word like t--f (name range variable list):
1. Extract the beginning and ending variable names (name1 and name2 in
the code below).
2. Get the position numbers for name1 and name2 (p1 and p2 in the code
below).
3. Check errors.
4. Update the counter (count)
5. Get the names of the variables between name1 and name2:
            %sysfunc(varname(&dsid, j))  p1<= j <=p2
6. Add the names to the namelist using a loop.
*/
%if &p %then %do;
       %let name1=%substr(&word, 1, &p-1);
       %let name2=%substr(&word, &p+2);
       %let p1=%sysfunc(varnum(&dsid,&name1));
       %let p2=%sysfunc(varnum(&dsid,&name2));
       %if &p1=0 | &p2=0 | (&p1 > &p2) %then %do;
              %put ERROR: Invalid variable list &word;
              %goto Exit;
       %end;
       %let count=%eval(&count+&p2-&p1+1);
       %do j=&p1 %to &p2;
              %let namelist=&namelist %sysfunc(varname(&dsid, &j));
       %end;
%end;

/* Parse a word like ab: (name prefix variable list):
1. Extract the prefix.
2. Using a loop to compare the prefix with each variable name in the data
set.
3. If a name matches the prefix, add it to the namelist.
*/
%else %if %index(&word, :) %then %do;
       %let prefix=%sysfunc(compress(&word, :));
       %do j=1 %to &_n_;
              %let name=%sysfunc(varname(&dsid, &j));
                                          %if (%length(&name) >=
              %length(&prefix)) & (%sysfunc(compare(&prefix, &name,
              :i))=0) %then %do;
                     %let count=%eval(&count + 1);
                     %let namelist=&namelist &name;
              %end;
       %end;
%end;

/* Parse special SAS Name lists: _ALL_, _NUMERIC_, or _CHARACTER_ */
%else %if %sysfunc(indexw(_ALL_ _NUMERIC_ _CHARACTER_, &upword)) %then
%do;
       %do j=1 %to &_n_;
              %if &upword=_ALL_ %then %do;
                     %let namelist=&namelist %sysfunc(varname(&dsid, &j));
                     %let count=%eval(&count + 1);
              %end;
              %else %if %sysfunc(vartype(&dsid, &j))=%substr(&upword,2,1)
              %then %do;
                     %let namelist=&namelist %sysfunc(varname(&dsid, &j));
                     %let count=%eval(&count + 1);
              %end;
       %end;
%end;
```

```
/* Parse a word like x-numeric-b or x-character-b */

%else %if %index(&upword, -NUMERIC-) | %index(&upword, -CHARACTER-) %then
%do;
       %let p=%index(&upword, -NUMERIC-);
       %let q=%index(&upword, -CHARACTER-);

       %if &p %then %do;
              %let name1=%substr(&upword, 1, &p-1);
              %let name2=%substr(&upword, &p+9);
              %let type=N;
       %end;
       %else %do;
              %let name1=%substr(&upword, 1, &q-1);
              %let name2=%substr(&upword, &q+11);
              %let type=C;
       %end;

       %let p1=%sysfunc(varnum(&dsid,&name1));
       %let p2=%sysfunc(varnum(&dsid,&name2));

       %if &p1=0 | &p2=0 | (&p1 > &P2) %then %do;
              %put ERROR: Invalid variable list &word;
              %goto Exit;
       %end;

       %do j=&p1 %to &p2;
              %if %sysfunc(vartype(&dsid, &j))=&type %then %do;
                     %let namelist=&namelist %sysfunc(varname(&dsid, &j));
                     %let count=%eval(&count + 1);
              %end;
       %end;
%end;

/* Parse a word like a1-a20 or b003-b152 (numbered range variables)*/
%else %if %index(&word, -) %then %do;
       %let p=%index(&word, -);
       %let name1=%substr(&word, 1, &p-1);
       %let name2=%substr(&word, &p+1);
       %let k=%eval(%sysfunc(notdigit(&name1, -%length(&name1))) + 1);
       %let prefix=%substr(&name1,1, &k-1);
       %let suffix1=%substr(&name1,&k);
       %let suffix2=%substr(&name2,&k);
       %if %sysfunc(varnum(&dsid,&name1))=0 |
       %sysfunc(varnum(&dsid,&name2))=0 | (&suffix1>&suffix2) %then %do;
              %put ERROR: Invalid variable list &word;
              %goto Exit;
       %end;

       %let len=%length(&suffix1);
       %do j=&suffix1 %to &suffix2;
              %let d=%eval(&len-%length(&j));
              %if &d <=0 & %sysfunc(varnum(&dsid,&prefix&j)) %then %do;
                     %let namelist=&namelist &prefix&j;
                     %let count=%eval(&count + 1);
              %end;
              /* if &d>0 pad j with d leading 0s and save as jj */
              %else %do;
                     %let jj=&j;
                     %do k=1 %to &d;
                            %let jj=0&jj;
                     %end;
```

```
                                %if %sysfunc(varnum(&dsid,&prefix&jj)) %then %do;
                                        %let namelist=&namelist &prefix&jj;
                                        %let count=%eval(&count + 1);
                                %end;
                        %end;
                %end;
        %end;

        /* Parse a word like cd - just add it to the namelist */
        %else %if %sysfunc(varnum(&dsid,&word)) %then %do;
                %let count = %eval(&count + 1);
                %let namelist=&namelist &word;
        %end;

        /* An unrecognized word */
        %else %do;
                %put ERROR: Invalid variable name or list &word;
                %goto Exit;
        %end;

        %let i=%eval(&i+1);
    %end;
%end;

/* Parse a Perl regular expression */
%else %do;
    %local pid;
    %let pid =%sysfunc(prxparse(&varlist));
    %if &pid=. %then %goto Exit;
    %else %if &pid %then %do j=1 %to &_n_;
        %let name=%sysfunc(varname(&dsid, &j));
        %if %sysfunc(prxmatch(&pid, &name)) %then %do;
            %let namelist=&namelist &name;
            %let count=%eval(&count + 1);
        %end;
    %end;
    %syscall prxfree(pid);
%end;

%let rc= %sysfunc(close(&dsid));

%if &nvars ^= %then %do;
    %global &nvars;
    %let &nvars = &count;
%end;

&namelist
%Exit:
%Mend Parse;
```