Paper AD09

Parsing Variable Lists: A Macro Function for Macros

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

ABSTRACT

Variable lists are often used in SAS procedures and DATA steps for referencing groups of variables. However, in many situations variable lists cannot be used directly in macros without enumerating the variables, which limits the macros' capability of working with multiple variables. To make macros work with any variable lists just as SAS procedures do, it is essential for the macros to be able to parse the variable lists.

This paper presents a macro function %Parse(dsn, varlist <, nvars>) 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 PROC steps and DATA steps) or a Perl regular expression. The function returns a complete list of variable names corresponding to the variable list. Optionally, the function saves the number of variables in the list to a global variable (nvars) specified by the user. With Perl regular expressions, this function can parse variable lists far beyond the conventional ones used in SAS procedures and DATA steps.

It is found that the run time for macro function %Parse(dsn, varlist <, nvars>) is not affected by the data set (dsn) size but proportional to the number of the variables in the data set. In addition, three simplified versions of the macro are also discussed.

INTRODUCTION

A variable list is a shortcut for referring to a list of variable names. SAS allows the following variable lists to be used in DATA steps and procedures (see SAS OnlineDoc 9.1.3 on SAS Variable Lists):

Table 1. SAS Variable Lists

Variable List	Example	Includes
Individual variables	cd es	cd es
Numbered range	a1-a30 (or b03-b50)	a1, a2, …, 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 very useful because they provide a quick way to reference groups of variables. They are often used in procedures and DATA steps. We can pass any conventional variable lists to the procedures (usually in the VAR statement) and the procedures "know" what variables are in the lists. But in many situations variable lists can not be used directly in macros without listing the variables. To make a user–defined macro work with any variable list just as SAS procedures do, it is essential for the macro to be able to parse the variable list. Here "parse" means to generate a list of all the variable names corresponding to the abbreviated variable list. Then the macro can extract any variables from the name list for processing. 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). This macro can parse patterns that are beyond the conventional SAS variable lists. However, this macro does not cover the "basics" – 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 the current 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 %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<,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 like those used in SAS procedures and DATA steps such as
varlist = cd b03-b50 t -- f xy: _NUMERIC_ r-character-z

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

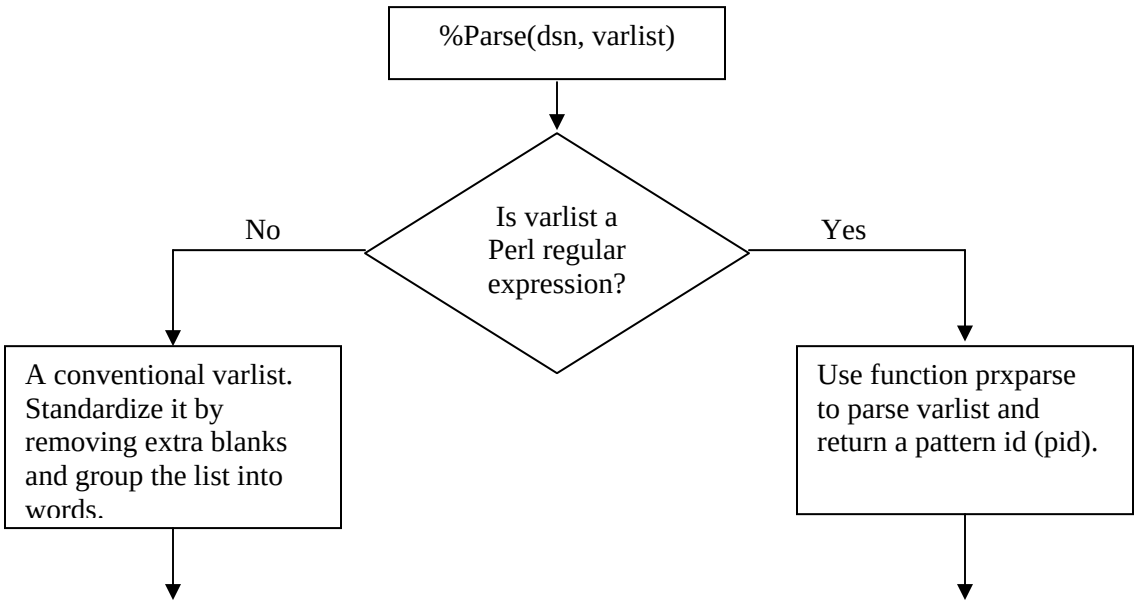
Note: the two forms are not allowed to be mixed together in the same variable list.

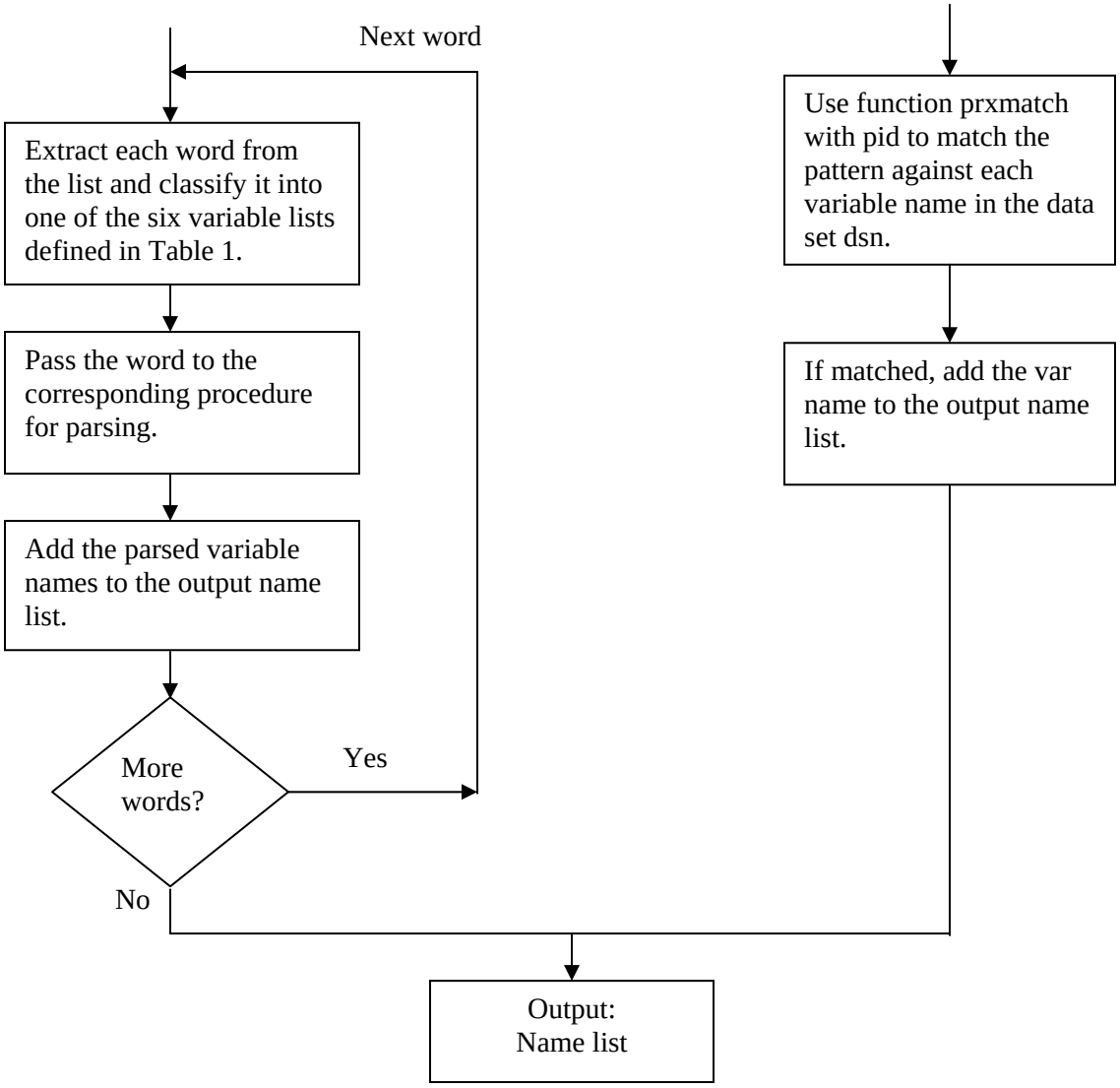
nvars - optional parameter to specify the name of a global variable to hold the number of variables returned by the function.

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).

The implementation of this macro 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. Implementation Flow Chart for %Parse





Because the implementation of this macro includes some relatively new character macro functions from the SAS default AUTOCALL macro library, you need to run this macro in SAS version 9 and make sure the default AUTOCALL library is accessible to your program. 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 severe limitations. In the following I will discuss these simplified versions of macro %Parse.

THREE 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 identical syntax that is 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, nvars);
       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;

        %if &nvars ^= %then %do;
            %global &nvars;
            %let &nvars=&nn;
        %end;
    %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_. If the optional parameter nvars is specified, it is assigned the value of &nn (number of names in &namelist). Below is the second simplified macro:

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

        %local nn;
        %global &namelist;

        proc sql noprint;
            select count(*), name into
                :nn,
                :&namelist separated by ' '
            from _tmp_;
        quit;

        %if &nvars ^= %then %do;
            %global &nvars;
            %let &nvars=&nn;
        %end;
    %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, and the number of names into nn. The third simplified macro is as follows:

```
    %macro Parse3(dsn, varlist, namelist, nvars);
        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_ '' _count_ 0;
            _cat_ = strip(_cat_) || " " || _name_;
            _count_ + 1;
            if _last_ then do;
                call symputx('names', _cat_);
                call symputx('nn', _count_);
            end;
```

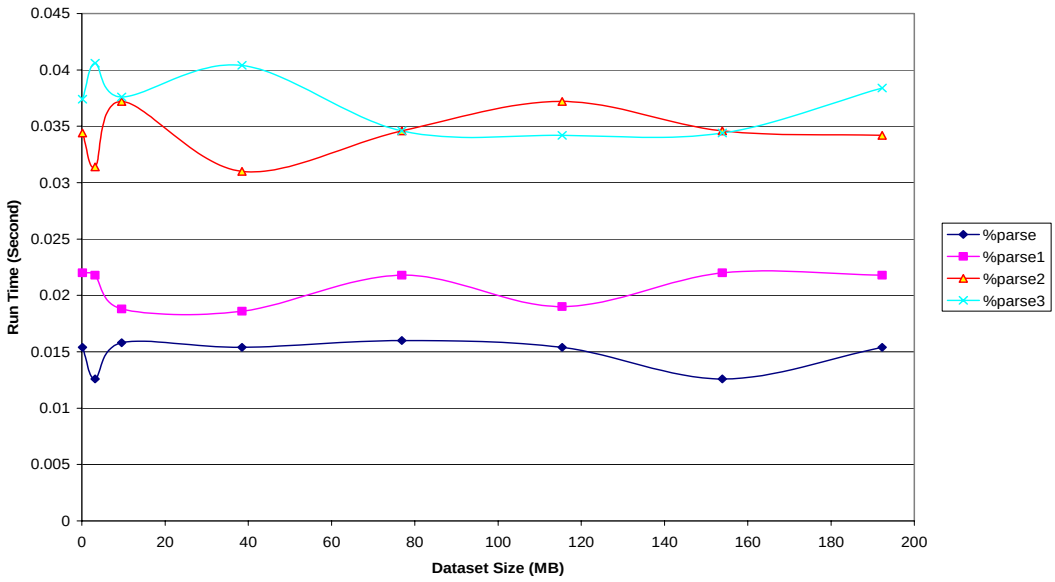
```
        run;

        %global &namelist;
        %let &namelist=&names;

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

MACRO RUN TIME

The code for %Parse is much longer than those for macros %Parse1, %Parse2, and %Parse3, but which one runs fastest? What affects the run time for these macros? To investigate these questions, two major factors related to macro run time are considered: data set size and the number of variables in the data set. Two tests were conducted to explore their relationships with the macro run time. The first test was designed to explore the relationship between the data set size and the macros' run time by controlling the number of variables. The first test included eight data sets of different sizes (ranging from 17K to 192M) but each had 30 variables. The variable list passed to the macros was the same for all data sets. Each macro ran five times with each data set. The macro run time for each data set was computed as the mean time from the five runs. The following chart shows the results from the first test (Chart 2):

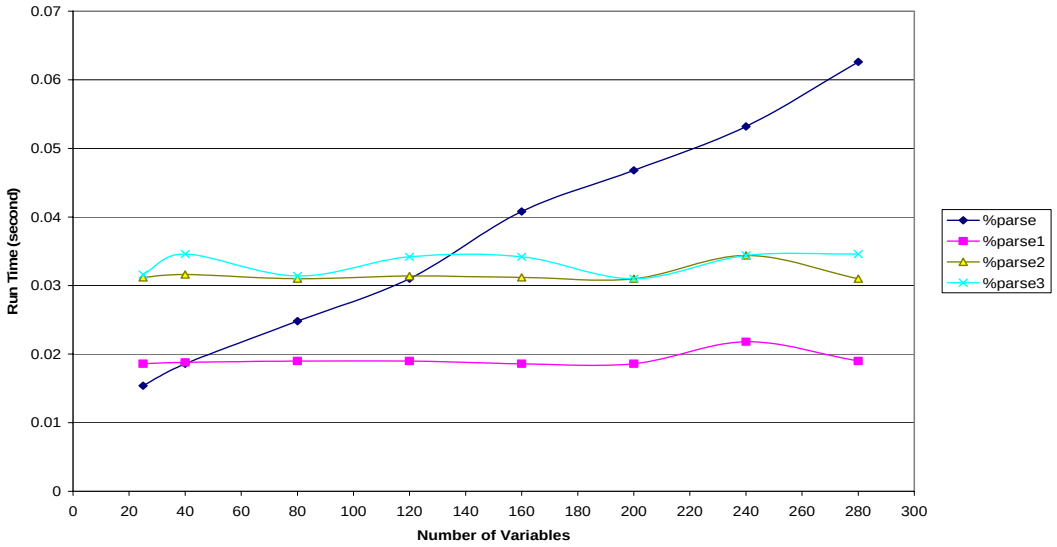
Chart 2. Macro Run Time Vs Dataset Size
(Each dataset has 30 variables)



The run time curves for macros %Parse and %Parse1 - %Parse3, though fluctuate a little bit, are pretty flat as the data set size increases. This indicates that these macros' run time is not affected by the size of the data set. The reason for this is probably that these macros only access the descriptor portions of the data sets regardless of their sizes. For small number of variables (30 in this test), %Parse runs fastest although its code is the longest.

The second test was designed to investigate the relationship between the macros' run time and the number of variables in the data set by controlling the size of the data sets. This test used eight data sets of size 33K with the numbers of variables ranging from 25 to 280. The variable list passed to the macros was the same for all data sets. Like the first test, each macro ran five times with each data set. The mean of the run time from the five runs for each data set is shown in the following chart (Chart 3):

Chart 3. Macro Run Time Vs Number of Variables
(Dataset Size = 33K)



The chart indicates that the run time for %Parse increases approximately linearly as the number of variables in the data set increases. This is probably due to the fact that, depending on the variable list to be parsed, macro %Parse may loop through the variables in the data set to search for the variables specified in the variable list. In contrast, the run time for macros %Parse1, %Parse2, and %Parse3 stays almost the same as the number of variables increases, which indicating the efficiency of the built-in keep option in selecting variables. Chart 3 suggests that for data sets with small number of variables (<40 in this test), macro %Parse runs faster than any other macros. For data sets with large number of variables (>160 in this test), %Parse is slower. The results may vary significantly with different system running environments.

In summary, the run time for macro function %Parse is not affected by the data set size but proportional to the number of the variables in the data set. The run time for macros %Parse1 - %Parse3 is not affected by data set size or the number of variables in the data set. However, keep in mind that %Parse can parse both conventional and Perl regular expression variable lists, but macros %Parse1 - %Parse3 can only parse conventional variable lists. As a macro function, %Parse can be called anywhere in a program, whereas %Parse1 - %Parse3 can only be called outside any DATA steps and PROC steps because these macros have their own DATA steps or PROC steps.

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--check2 s:;
%put The variable names are: %Parse(Example, &varlist, n);
%put The number of variables in the list is: &n;
%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$/);
%put The variables containing a digit 2 are: %Parse(Example, /2/);
%put The variables with two consecutive digits are: %Parse(Example, /\d\d/);
%put Variable names with a length of 3 are: %Parse(Example, /^.{3}$/);
```

CONCLUSION

This paper introduces a macro function %Parse(dsn, varlist <, nvars>) that can parse any variable list (varlist) for a given data set (dsn). The variable list can be a conventional SAS variable list or a SAS Perl regular expression. 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*.

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
CREATED: 7/1/2003, REVISED: 5/30/2006, by Jimmy Z. Zou

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;

/* Open data set dsn */
%let dsid=%sysfunc(open(&dsn));
%if not &dsid %then %do;
```

```
      %put %sysfunc(sysmsg());
      %goto Error;
%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 the varlist if any and group the variables into words
      separated by "*". */
      %let varlist=%cmpres(&varlist);
      %let varlist=%sysfunc(translate(&varlist, '*', ' '));
      %let varlist=%sysfunc(tranwrd(&varlist, *-, -));
      %let varlist=%sysfunc(tranwrd(&varlist, -*, -));

      /* 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());
            %let word=%qscan(&varlist, &i, *);
            %let upword=%upcase(&word);
            %let p=%index(&word, --);

            /* Parse a word like t--f (name range variable list) */
            %if &p %then %do;
                  %let name1=%substr(&word, 1, %eval(&p-1));
                  %let name2=%substr(&word, %eval(&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 Error;
                  %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 xy: (name prefix variable list) */
            %else %if %index(&word, :) %then %do;
                %let prefix=%substr(&word, 1, %eval(%length(&word)-1));
                %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, %eval(&p-1));
            %let name2=%substr(&upword, %eval(&p+9));
            %let type=N;
        %end;
        %else %do;
            %let name1=%substr(&upword, 1, %eval(&q-1));
            %let name2=%substr(&upword, %eval(&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 Error;
        %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, %eval(&p-1));
        %let name2=%substr(&word, %eval(&p+1));
        %let k=%sysfunc(anydigit(&name1));
        %let prefix=%substr(&name1,1, %eval(&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 Error;
        %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 Error;
            %end;

            %let i=%eval(&i+1);
        %end;
        %let rc= %sysfunc(close(&dsid));
%end;

/* Parse a Perl regular expression:
1. Function prxparse parses &varlist and returns a pattern id (pid) if successful.
2. If successful (pid>0), set up a loop to match the pattern against each variable
name in data set dsn using prxmatch function.
3. If a match is found, add the name to the namelist and increase the counter.*/
%else %do;
        %local pid;
        %let pid =%sysfunc(prxparse(&varlist));
        %if &pid=. %then %goto Error;
        %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;

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

&namelist
%Error:
%mend Parse;
```