%RepeatMaskoverFile

Paul Silver, NORC at University of Chicago, 2022

Abstract

Introduction

Examples

Usability Notes

Conclusions

ABSTRACT

- %RepeatMaskoverFile (referred to as %RMOF below) will generate SAS code from
- quoted (as necessary) macro expressions (mask, begin, separator, end) containing macro variables and/or macro calls,
- Using macro parameters using a SAS data file as input
- RMOF relies on %symcall set() to transfer SAS data set values after the open and fetch functions calls into SAS macro variables.

Click Headings Above to View Content

%RepeatMaskoverFile

Paul Silver, NORC at University of Chicago, 2022

INTRODUCTION : %RMOF Macro Parameters

Mandatory Parameters

_dset: name of SAS file containing variables to resolved into macro variables of the same name in the _mask macro expression. It is a required positional parameter (ie, no _dset= before it). It can be a physical data set or a virtual view. It can have 0 or any number of rows or variables. If the file is unreadable, a message will be printed and a %abort will be executed. If variables in the _keep or _where parameters are not present or wrongly written so that the generated file options don't work, %abort will also be executed. Otherwise standard SAS warning or compilation messages will result from incorrect specification.

_mask: Text string contain code to be generated after resolving all macro expression, for each row in **_dset.**

Optional Parameters

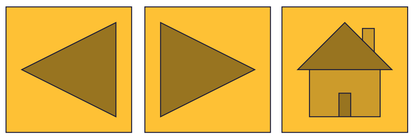
_where: Logical expression to used in a where data set option clause while reading **_dset** to limit rows used in the code generation step.

_keep: List of variable names to retain and resolve into macro variables of the same name before _mask resolution. It is not necessary, and is more useful for making clear which variables are to be used from the **_dset** file.

_begin_text: Generate code from this expression **before** the first _mask generation, but only if there are rows in **_dset**.

_sep_text: Generate code from this expression between the _mask code generation instances, but not after the last one.

_end_text: Generate code from this expression after the last _mask generation, but only if there are rows in **_dset**.



Previous Next Home

%RepeatMaskoverFile

Paul Silver, NORC at University of Chicago, 2022

Generate Variable List

Example 1: Generate variable list from data set work.varlist containing a list of sas variable names to keep in output data set B from input dataset a.

```
Data b; set a;
   %RepeatMaskoverFile(
   work.varlist /* input data set*/
   ,_beg_text=Keep
   ,_keep=varname
   ,_mask=%nrstr(
   &varname /* expression to resolve*/
   ));
run;
```

Generate if/then SAS statements by type

Example 2: Generate SAS statement to recode variable (variable) from missing to a skip code) if a logical expression (expression) is true. Note: This will fail if skipcode is given but expression is missing.

```
Data b; set a;
   %RepeatMaskoverFile(work.varlist
   ,_keep=varname expression vartype skipcode
   ,_where=vartype='C' and skipcode>' ' /* include only character variables with defined skip codes*/
   ,_beg_text=*set skipcodes as needed for character variables%str(;) /* add a comment at the beginning of the statement list, but only if some rows met the _where condition*/
   ,_mask=%nrstr(if &varname=' '  then if &expression then &varname="&skipcode";)
   ,_end_text=* end of character variable skipcodes %str(;)
   );
   %RepeatMaskoverFile(work.varlist,
   ,_keep=varname expression vartype skipcode
   ,_where=vartype='N' and skipcode>' '
   ,_mask=%nrstr(if &varname>. then if &expression then &varname=&skipcode;)
   );
   run;
```

Generate if/then SAS statements across type

Example 3: Combine processing of numerical and character variables by adding a macro to properly type the skipcode expression

```
Data b;  set a;
%* Predefine %skipcode as follows: ;
 %macro skipcode(skipcode,type); %if &type=C %then "&skipcode";%else &skipcode;%mend;
%RepeatMaskoverFile(work.varlist,
   ,_keep=varname expression vartype skipcode
   ,_where=skipcode>' '
   ,_mask=%nrstr(if missing(&varname) then if &expression then &varname=%skipcode(&skipcode,&type)%str(;))
   );
run;
```

Generate Proc Freq on selected variables

Example 4: Generate Proc freq after applying code above to variables with skipcodes. No code will be generated if all rows in varlist have skipcode missing. Note: _keep is dropped, without impact.

```
%RepeatMaskoverFile(work.varlist
   ,_where=skipcode>' ' /* include only character variables with defined skip codes*/
   ,_beg_text=%str(proc freq data=b;)
   ,_mask=%nrstr(table &varname/missing list;)
   ,_end_text=run%str(;)
);
```

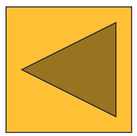
Previous Next Home

%RepeatMaskoverFile

• Paul Silver, NORC at University of Chicago, 2022

Usability Notes

- **Execution Context**: It can be used virtually anywhere in a SAS program, including open code, data steps, or proc steps.
- **Advantages**: **RMOF** generates SAS code, not macro variables, and so is **NOT** subject to the very annoying macro variable size limitation of 65,524 bytes. If the input file is empty it will automatically do nothing.
- SAS Code Generation is easier than other methods like proc sql select :into separated by) because it typically requires few or no text functions and asks you to type SAS code as you would type in plain code, just replacing specific values with macro variables.
- It contains all or most of the specifications directly in the place it is used. It can mix global and local macro variables or macro calls or other text seamlessly. It will replace many, many custom macros or data step or proc sql code macro variable generation programs.
- **&_n_** can be used in the **_mask** or **_sep_text** or **_end_text** parameters to allow a sequential number to be included in the generated code.
- **Limitations**: The macro expressions must not contain macro statements (like %if %then %do%end). However, that functionality can often be added by using wrapper macros defined before the %RMOF call (see %skipcode example above). Local macro variables that should **NOT** be used in the macro expressions include **_dsetopt _dsid _rc as well as any of the macro parameters**, since they might collide with the file generated macro variables and break the macro execution. Quoting functions like %nrstr, %nrbquote, or %superq must typically used around the input expressions to prevent resolution till macro execution time. In most cases %nrstr will suffice, though.



Previous

Home

%RepeatMaskoverFile

- **Paul Silver, NORC at University of Chicago, 2022**

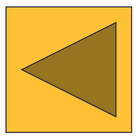
CONCLUSION

Consolidate SAS statement generation from a file and execution in one macro statement!

Avoid the macro variable limit by not using macro variables!

Avoid writing custom macros by using a general purpose utility macro

Easily write code that can handle 0 or unlimited input code generation files



Previous

Home