%RepeatMaskoverFile (RMOF)

- This Utility Macro (RMOF) will repeatedly create and execute SAS Code by unquoting the mask (a SAS Macro Expression containing Macro variable references) after setting macro variables to values for each row in a SAS data file.
- Paul Silver, System Architect, University of Chicago

MWSUG 2024
WAUKESHA, WI
LEAD THE WAY AND PLUG INTO SAS!

%RepeatMaskoverFile: Abstract

- %RepeatMaskoverFile (referred to as %RMOF below) will generate SAS code from a SAS Data file, by repeatedly resolving a macro expression composed of macro variable references, and virtually any other text (except Macro statements) after setting macro variables to the variable values in the SAS Data file (and any other desired Macro references).
- The RMOF _mask parameter will define a quoted expression which outlines the format of the code to be generated.
- SAS macro variables will be set from each row in the SAS data file and _mask will be resolved using the %unquote function and executed automatically.

%RMOF Macro Parameters – mandatory

- **(_dset,..)**: name of SAS file containing variables to resolved into macro variables of the same name in the _mask macro expression. It is the only positional parameter in the RMOF macro call.
- **_dset** can reference a physical SAS data set or a virtual view. It can have 0 or any number of rows or variables. It can be modified by an optional where clause to limit the number of rows used.
- **_mask**= a macro expression which contains code to be generated after resolving all macro expressions, for each row in **_dset**. It must usually by quoted by %nrstr(), %nrbquote, or other macro quoting function that prevents macro variable resolution until %unquote() within RMOF execution.

%RMOF Macro Parameters – optional

- **_where** and **_keep** control which rows and columns are used from the _dset file.
- **_where**: optionally limit the incoming rows from **_dset** by defining a where expression to be applied while reading **_dset**.
- **_keep**: limit list of variables from **_dset** if desired. If left blank, all variables in **_dset** will be transferred to macro variables before _mask resolution. Variables used in **_where** must also be specified here if **_keep** is non-blank. This might be necessary if the data set names collide with other local macro variables or macro parameters. For example, if there is a variable named **_dset** in _dset, it must be excluded using the _keep parameter, in order for RMOF to work properly.

%RMOF Macro Parameters – optional 2

- This optional parameters control the layout of the code being generated, in addition to **_mask**.
- **_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**.

%RMOF Example 1 : Generate Variable list

- Example 1: Generate variable list from _**dset** 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*/)
- ,_end_text=%str(;)
-);
- run;

- Results in >
- **Data b; set a;**
- **keep var1 var2 var3; * assuming varlist contains three rows with varname = var1, var2, var3;**
- **run;**

%RMOF Example 2 : Recode variables from missing to skip

- 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;

%RMOF Example 3: Generate if/then statements for all types

- Example 3: Combine processing of numerical and character variables by adding a macro to properly type the skipcode expression, execute data set update only if some variables in varlist have skipcode conditions.

- %* Predefine %skipcode for use in %RMOF:
- **%macro skipcode(skipcode,type); %if &type=C %then “&skipcode”;%else &skipcode;%mend;**

- %RepeatMaskoverFile(work.varlist,
- ,_keep=varname expression vartype skipcode
- ,_where=skipcode>’’
- ,_beg_text=%str(data a; set a;)
- ,_mask=%nrstr(if missing(&varname) then if &expression then &varname=%skipcode(&skipcode,&type)%str(;)),
- _end_text=%str(run;)
-);

%RMOF Example 4: Generate Proc freq for 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=a;)
- ,_mask=%nrstr(table &varname/missing list;)
- ,_end_text=run%str(;)
-);

%RMOF 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 do nothing.
- Code Generation can be 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.
- %RMOF call can contain 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 can 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.

%RMOF Usability Notes: Limitations and Cautions

- 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.

Thank You for Listening!

- Paul Silver
- System Architect, NORC at University of Chicago
- Silver-paul@norc.org

MWSUG 2024
WAUKESHA, WI
LEAD THE WAY AND PLUG INTO SAS!