Paper 012-31

List Processing - Make Light Work of List Processing in SAS®

Peter Crawford, Crawford Software Consultancy Limited

ABSTRACT

In the world of extensive metadata, list processing prevails. A small utility macro has improved applications development (source and maintenance) where they handle lists. This paper demonstrates techniques. Over a period of 4 years, this macro has been refined into a robust and reliable facility that enables applications to delegate most list processing and leave cleaner, clearer code that will be cheaper to build and maintain.

INTRODUCTION

The problem: application development risks are rising.

This paper offers a solution that will help application developers using base SAS® macros, as a result, reduce risk.

APPLICATION DEVELOPMENT RISK

Most business definitions start out unique, but end up looking just like long lists. Metadata has reduced the number of programs that are being maintained just to keep up with developing business requirements. Many processes face the challenge of applying these "lists" and maintaining them.

In many situations it has been easy enough just to copy and clone existing macro code for list management. However, the repetitious nature of the cloned code, and the need to ensure safe version history meant code now inherited "in production" varies often enough to introduce a rising risk of failure.

SOLUTION SUPPORT

Macro has its place. It is not a suitable environment for high volume processing, nor likely to provide an adequate user interface, but it remains popular. That popularity is based on the convenience for generating text that can be used as syntax. This naturally fits into the deployment of metadata. Macros can traverse lists with a variety of iteration methods. This confronts the application maintainer with the nightmare (too many ways, too much code) → much better to reduce the metadata handling to events and list processing. That needs little out of the ordinary, but … a little extra discipline.

Rather than code in the style demonstrated in Appendix 1, consider supporting this:

```
%mLoopsX( execut= simple_task
        ,   with= list of values to be used as parameters for simple_task )
```

All the loop handling remains hidden within macro `%mLoopsX` .The processing for a single instance is prepared within a local macro, called here, `%simple_task`. The next sections will outline the principles and practice.

PRINCIPLES

The solution design proposed is a combination.

1. application specific processes, encapsulated within a macro, and called with a single parameter (object definition)
2. a list of values (or events) – each entry in the list is the parameter(s) to pass to the above process
3. robust standard (locked-down) macro for list-processing. (don't change it. If it doesn't do what you want, don't use it)

In combination this design might be enough.

The macro in Appendix 2 provides a well tested harness. Use it to invoke the "one-time" process for each entry in the list.

PRACTISE

A very simple use of the macro is demonstrated within the macro body. In a situation where you might want to collect the current value of several macro variables – for example, to be reinstates after a process you prefer to expose or hide, you might collect the separate values an reinstate each later. Use a small macro like

```
%MACRO syk( option ) /DES= 'collect option value with keyword' ;
 %SYSFUNC( GETOPTION( &option, KEYWORD )
%MEND  syk ;
```

In a call to mloopsx(), you establish the current settings, prior to temporarily replacing them with the requested preference.

```
%let  preserved_options = %mloopsx( execut= syk, with= &requested_overrides );
OPTION &requested_overrides ;
```

…….. next comes the section of process subjected to the overridden options …

…….. followed by …

```
 * reinstate old settings;
OPTION &preserved options ;
```

The macro has reduced list processing to one call, using an existing list as data instead of syntax

The concept is applicable in many situations: collecting performance information from many servers; executing a list of jobs; importing location-dependant data.

IS %MLOOPSX A COMPLETE SOLUTION FOR LIST PROCESSING?

OK, it isn't enough, because I now find I want more!

I supported a client who keeps monthly performance data in tables named with the month (in MONYY format). Uncomfortable at over-reliance on my `%mLoopsX()` macro for the most basic of situations, I generated a list-processing macro demonstrated in Appendix 3.

The principle that I took as justifying another macro: iterating over very similar items – they are just consecutive dates in MONYY format. That lends it self to a loop with two parameters defining the end-points.

The macro,

```
%months_set()
```

loops over a date range by months. On each iteration, it replaces a feature with the MONYY value. The feature appears as ##### in the string parameter of the macro call. The feature to be replaced is only ##### by default. Where an application cannot use that, the macro allows a different feature to be defined. The macro becomes very versatile in the environment of this monthly business model, while remaining simple.

CONCLUSION

The risk in list processing can by reduced by using simpler syntax - enabled by these robust list-processing macros.

RECOMMENDED READING

Internet search engines were not much help for this application, offering so many services for address list cleaning and marketing that the list programming language Lisp appeared well below the first page.

CONTACT INFORMATION

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

Peter Crawford
Crawford Software Consultancy Limited
31 Sefton Road,
Croydon, Surrey, CR0 7HS, UK
+44 7802732254
CrawfordSoftware@gmail.com

APPENDIX

1

OLD-STYLE LIST PROCESSING

Here is an example of what I refer to as "old-style list processing". The code fragment provides a mix of base SAS syntax list handling in variable lists and arrays, as well as macro syntax for loop and list processing .
This is the kind of risky complexity that I hope my kind of list processing will help.
What follows was intended to be supported in production. Information and parameters are moving through several looping and addressing techniques.

```
data _null_;
    x = count(compbl("&stts")," ");
    call symputx('stats',x + 1);
run;
data stats;
    length  STAT1 - STAT&stats $6.
          ;
    array words{&stats} STAT1 - STAT&stats ;
    do i = 1 to &stats;
         words{i} = scan("&stts",i," ");
    end;
    drop i
        ;
run;
proc transpose data=stats out=stats_trans(drop=_NAME_);
    var STAT1 - STAT&stats;
run;
data _null_;
    set stats_trans;
    do i = 1 to &stats;
       call symput('stats'||compress(put(_n_,8.)),COL1);
    end;
run;

 /*************************************************************/
 /* RUN PROC ON VARIABLES ONCE FOR EACH REQUEST. MERGE DATA WITH   */
 /* DATASET OF                                                    */
 /*************************************************************/

%do z = 1 %to &stats;
```

APPENDIX 2

MACRO TO SUPPORT NEW-STYLE LIST PROCESSING

The macro that follows is not the only way to achieve the objective, but is has been tested by implementation in many organizations. The code has had very few changes in the last 4 years

```
/*-------------------------------------------------------------------------*
 * Program : mloopsx.sas                                                   *
 *-------------------------------------------------------------------------*
 * Purpose : List processing: for each element of a list                   *
 *         :                  call a macro                                 *
 *         :                  without going through the statement boundary
 *-------------------------------------------------------------------------*
 * Author  : Peter Crawford - Crawford Software Consultancy Limited        *
 * Version : 2.1                                                           *
 * Created : 14.10.02                                                      *
 *-------------------------------------------------------------------------*
 *Modification History                                                     *
 * Peter Crawford 14.10.02                                                *
 *  prefix local macro vars with macro name, to avoid contention with any*
 *  macro vars inherited by the macro called                              *
 * Peter Crawford 20.02.04                                                *
 *  apply %superQ() to macro vars in tests, to avoid arithmetic in        *
 *  implicit %eval()                                                      *
 * 13/02/2005 superQ on testing for empty execut=                         *
 *-------------------------------------------------------------------------*/

%macro mLoopsX( execut   =         /* name of macro to invoke */
              ,   with   =         /* delimited list of calls */
              ,   withdlm= %str( ) /* list delimiter */
              ) /des='execute a macro for each word in a list' ;
  /* examples:
     suppose you have a macro like
               %macro  syk( op );
                  %sysfunc( getoption( &op,keyword ) )
               %mend   syk ;
      then
     %put oldops = %mloopsx( execut= syk, with=mprint mlogic symbolgen);
      loads re-usable settings for these system options
      The macro can operate on larger scale as a statement like
          %mloopsx( execut=main_process, with= London New-York Tokyo )
   *********/
  %if %superQ(execut) eq or %superQ(with) eq %then %goto mParmError ;
  %local mloopsx_i mloopsx_j ;
  %let   mloopsx_i = 1 ;
  %let   mloopsx_j = %scan( &with, &mloopsx_i, &withdlm );
  %do %until( %superQ(mloopsx_j) eq ) ;
        %&execut.(   &mloopsx_j )
      %let   mloopsx_i = %eval(        &mloopsx_i +1 );
      %let   mloopsx_j = %scan( &with, &mloopsx_i, &withdlm );
  %end ;
  %goto mExit ;

  %mParmError:
  %put mLoopsX-Error need parameters, got execut=&execut and with=&with ;
  %mExit:
%mend  mLoopsX ;
```

APPENDIX 3

SUPPORTING VARYING DATE LISTS

Macro to support generating an ordered list of dates as a series of text .

```
/*****************************************************************
*  program     : months_set.sas                                  *
*  version     : 0.1                                             *
*--------------:--------------------------------------------------*
*  originator  : Peter Crawford, Crawford Software Consultancy Ltd   *
*--------------:--------------------------------------------------*
*  description : loop over date range replacing a pattern with MONYY *
*--------------:--------------------------------------------------*
*  outputs     : text *
*--------------:--------------------------------------------------*
* change hist. : opened 22/12/2005    and tested !                 *
*--------------:--------------------------------------------------*
* dd/mm/yyyy   :                                                  *
*****************************************************************/
%MACRO months_set ( string        /* pattern to repeat over dates */
                  , seek = #####  /* string to be replaced        */
                  , from = JAN04  /* dates in valid monyy style   */
                  , to= &last_mon  /* ending date                 */
                  , looplimit= 100 /* used to avoid over-running  */ );
   %DO %WHILE( %SYSEVALF( "1&from"d LE "1&to"d )  ) ;
%SYSFUNC( TRANWRD( &string, &seek, &from ))
      %LET from = %SYSFUNC( INTNX( MONTH, "1&from"d, 1 ), MONYY5);
      %LET looplimit = %EVAL( &looplimit - 1 );
      %IF &looplimit < 1 %THEN %GOTO mexit ;
   %END ;
%mexit:
%MEND  months_set ;
/******* demos
%LET last_m_end = %SYSFUNC( INTNX( MONTH, "&SYSDATE9"D, -1, ENDING), DATE9 );
%LET last_month = %SUBSTR( &last_m_end, 3 );
%LET last_mon   = %SYSFUNC( INTNX( MONTH, "&SYSDATE9"D, -1  ), MONYY5);
%LET begin_mon  = %SYSFUNC( INTNX( MONTH, "&last_mon"D, -12 ), MONYY5);

%PUT %months_set( prim.#####( IN= _in_#####  keep= fghj jkl WHERE=( a='fghj' ))
                , from= FEB04, to = jan2004) ;
 *** handles some macro code too ! like;
%months_set( %NRSTR(%%PUT #####;), from=MAR05, to = sep05  ) ;

DATA  collected_history ;
  payments = 0 ;
  DO UNTIL( LAST.key_column );
  SET
     %months_set(
          prime.#####( IN=_in_##### KEEP= key_column &perform_cols )
                , from= &begin_mon, to= &last_mon   )
      ;
   BY key_column ;
   Where  sum( %commas(&perform_cols) ) ;
   Payments + sum( of  &perform_cols ) ;
  End;
  * processing on account key ……;
  if payments then output;
Run;
*****************************************************************/
```