Paper 096-29

Write Log (%WL) a Production Logging Utility for Data Transactional Logs

Brett C. Peppe, T-Mobile (USA), Bellevue, WA

ABSTRACT

Often SAS Data Warehousing application programmers encounter situations where a formatted log entry must be written for later review or a permanent record. For example, the record of a production job run. Successful log messages are standardized, easy to maintain and efficient to use. The ***%WL()*** macro automates many of these log writing tasks in a simple form:

```
%WL( P0, P1, P2, P3, P4, off=nn );
```


INTRODUCTION

The %WL macro automates the process of creating log records and in so doing, frees the programmer to concentrate on the particulars of the D/W task at hand. It follows the KISS principle (Keep it Simple, Simon), has a tiny code footprint and automates many common log tasks. The example presented is tailored to writing log records to flat file storage, but the technique can be adapted to any output medium. Such storage is ideal for transactional logs because it is inexpensive, flexible and easy to search via GREP or other utility. A simple end-to-end example of this powerful technique is presented.

THE PROBLEM

Anyone who has worked on enterprise grade Data Warehousing (D/W) projects knows that such projects are eventually judged on three data issue perceptions:

- Is the organization of the data suitable?
- Is the classification of the data adequate?
- Is the data clean enough?

Achieving the first two of these objectives is not a simple task, but can be done if the architects and implementers of the D/W understand the sources of the data and the requirements of the users. However cleaning the data is also critical to the long-term success of a D/W, but usually is allocated insufficient resources. Human nature being what it is, everyone one wants to be the D/W architect (designer), no one wants to be the D/W janitor (cleaner). Clean data is auditable and reconcilable against its sources.

D/W projects are notorious for requiring complex and sophisticated transactional data loaders. It common to implement some of the required data validation rules used by these data loaders as referential integrity checks, data maps or error tables. But this approach is not always suitable because of cost, performance and the lack of a crystal ball to see future requirements. Despite these limitations, the D/W still needs to efficiently track exception conditions encountered, processing statistics, status, etc. ***What to do?***

THE SOLUTION

Creating D/W tables to store all the required message conditions on a large project can be very costly. A better approach is to use cheap flat storage in the form of a run or audit log. The structure of this log can range from the very simple to the complex. This paper describes a *conceptual macro framework* that is simple to build, easy to maintain, robust and very flexible. The important concept to grasp is that %WL will work with any set of strings, be it stored in a dataset, created in a data step or a format VALUE clause. %WL allows programmers to concentrate on the programming task, rather then worrying about log formatting.

A SAS macro is nothing more then a better way to generate text using a template. The %WL macro, or its derivatives illustrate a powerful technique to create logs that are:

- Standardized
- Easy to Maintain
- Efficient to Use

Everyone has a horror story about reading very large unstructured logs created via SAS PUT statements in data steps. %WL is the end product of an evolution that started many years ago when I found myself mired in one such exercise. A log is nothing more then a sequential list of messages. Standardized messages are always much easier to understand, both for the

author and especially others who must read these logs. Each entry can be designed to suit any task, but in general a message will have one or more of following elements:

- Message encoding
- Location or context reference
- Message text
- Parameters to embed
- Offset
- Log file target

What %WL does is to automate the assembly of these message strings, one at a time, and write them to a log file. Most of these messages uniquely identify a particular condition and so have an associated code, but this is not strictly necessary, as we shall see. Establishing %WL support has five operational steps as follows

- Build the Validation Message List (V_MSG)
- Remove duplicate and invalid Messages
- Build the Validation Message Lookup ($VMSG)
- Compile $VMSG
- The %WL macro definition itself

SEEING IS BELIEVING: AN EXAMPLE

Suppose a job needs to scan an external file and (1) identify the program, (2) date-time stamp when it started and (3) identify the filename scanned. This simple example has only three messages, but the message handlers can be made as sophisticated as necessary.

Step(1)- ***Build the Validation Messages List (V_MSG)*** as follows:

```
data one;
data  vmsg_0;
      infile  datalines;
      input
        @03   Err_cls    $char3.
        @03   Err_cd     $char10.
        @07   Err_lvl    $1.
        @14   Err_msg    $char65.
        ;
      if ( Err_cd = ' ' ) then delete;
      datalines;
  SYS.S00.01 MYSCAN(pver) ptitle
  SYS.S00.02 Run Start-- On: rundate  at: runtime
  SYS.S00.03 Scanning  file: scanfile
```

Step(2) - ***Remove duplicate and invalid Messages***. Again, this step can be as sophisticated as necessary. In this example, let us only guard against duplicate messages:

```
proc sort  data=vmsg_0  nodupkeys;
      by  Err_cd;
```

Step(3) - ***Build the Validation Message Lookup ($VMSG).*** This particular implementation of %WL uses a format-based lookup because it is simple, efficient and adequate to most tasks.

```
data  vmsg_0( keep= fmtname type start label hlo );
      fmtname = '$VMSG';
      type    = 'C';
      set  vmsg_0  end=lastobs;
      start   = Err_cd;
      label   = Err_msg;
      output;
      if ( not lastobs ) then return;
      hlo   = 'O';
      start = 'OTHER';
      label = '???';
      output;
stop;
```

Step(4) - ***Compile $VMSG***. Since this example implementation of %WL uses formats, lets use Proc Format to compile the format clause:

```
proc format  cntlin=vmsg_0 ;
   *  select  $VMSG ;
```

Step(5) - ***The %WL macro definition itself***. This version implements %WL as a DO group that is used inline within a data step statement. Note that this version writes messages to a hard-coded filename of RUNLOG.

```
%macro  WL( P0, P1, P2, P3, P4, off=5 );
     do;
        format  Err_Msg  $varying130. ;
        Err_Msg = put( "&P0", $VMSG. );
        %if ( &P1 ^= ) %then Err_Msg = tranwrd( Err_Msg, "&P1", &P1 );;
        %if ( &P2 ^= ) %then Err_Msg = tranwrd( Err_Msg, "&P2", &P2 );;
        %if ( &P3 ^= ) %then Err_Msg = tranwrd( Err_Msg, "&P3", &P3 );;
        %if ( &P4 ^= ) %then Err_Msg = tranwrd( Err_Msg, "&P4", &P4 );;
        Err_Msg = compbl(Err_Msg);
        file  RUNLOG  mod notitles ;
        put   @&off  Err_msg ;
    end;
%mend   WL;
```

Where:

- P0 is the Message code
- P1…P4 are DDV ***variables*** whose ***values*** will replace ***text*** in a message string
- off=nn is the offset from the left margin of the Log

The following SAS data step accomplishes tasks (1), (2) and (3):

```
data  _null_;
   retain  pver     "&pver"   ;
   retain  ptitle   "&ptitle" ;
   format  rundate   $10.     ;
   format  runtime   $8.      ;
   format  scanfile  $char42. ;
   infile  SCANTHIS  filename=scanfile ;
   input ;
   rundate = put( today(), yymmdd10. ) ;
   runtime = put(  time(),    time8. ) ;
   %WL(SYS.S00.01,pver,ptitle,off=3)
   %WL(SYS.S00.02,rundate,runtime,off=3)
   %WL(SYS.S00.03,scanfile,off=3)
stop;
run;
```

Producing the following log entry in RUNLOG:

```
SYS.S00.01 MYSCAN(1.2.P) My Validation Scanner
SYS.S00.02 Run Start-- On: 2003-01-10  at: 22:16:36
SYS.S00.03 Scanning  file:  /var/dev/data/myfile.txt
```

CONCLUSION

Because of the complexity of transactional loaders for enterprise grade data warehouses, the demand for high quality exception and status logging continues to increase, which continue to increase in complexity. Macro utilities such as the %WL can save a great deal of time in the creation, maintenance and utilization of these logs. I hope that you will find the %WL conceptual macro as useful I have. ***Happy %WL logging.***

CONTACT INFORMATION

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

Author Name:	Brett C. Peppe
Company	T-Mobile (USA)
Address	
City state ZIP	Bellevue, WA
Work Phone:	

Fax:
Email: brett.peppe@T-Mobile.com
Brett.peppe@verizon.net