

Warranty Processing System

Tim Vorce, Trillium Teamologies, Royal Oak, MI

ABSTRACT

The Warranty Processing System (WPS) provides valuable information about warranty performance. The customer is a parts supplier for an automotive company. Warranty claims information is imperative to improve performance of parts suppliers, by identifying parts that require service under warranty. SAS was utilized to process the data, partly because the data was available in this format. A specialized Intel platform was constructed for this project, because of the magnitude of the processing involved.

INTRODUCTION

The system receives very large data files for vehicle and claim information as SAS data sets. The specialized Intel platform running Windows NT then processes the information into small files that provide cost and repair incident information. The warranty performance is summarized into customer defined classes, called items. In addition, there are six different divisions (each building one of the functional areas of the automobile) that have their own concerns about the claims information. The list of items is unique to each division, but since each division processes the entire claims file, it would be possible for items to overlap.

The group that is responsible for maintaining the warranty data is called Automobile Warranty System (AWS). The AWS group also has a front end for extracting reports on warranty performance. This is not very robust and the number of inquiries required is prohibitive for this application. The WPS system was created to exploit receiving a full download of the AWS data, processing the data in a similar fashion, and producing custom reports. However, there are calculations in AWS that are not well documented, which complicated the validation process.

In the WPS, the criteria for the item definitions are received as comma separated (csv) files. The outputs are also csv files. Several HTML files are produced for displaying on a web site.

DATA INPUT FILES

The input files are quite large, and contain descriptions of the vehicles and claims encountered during the year. The claim file contains information about claims against vehicles manufactured in a specific model year. The vehicle files contain information about all of the vehicles produced in the specific model year. Because of the way the manufacturing processes works, this data includes more than twelve months of data. Since one car may have many warranty claims, the claim files may get rather large, in fact some files for the earlier years have reached a size of 8 Gigabytes (Gb). The hardware to support this system is a server specifically designed and leased to the customer for this purpose. It sports a RAID cluster of Ultra-wide SCSI disks, and dual Intel processors, running Windows NT workstation operating system. The disk storage appears to the

system as one logical drive of 50 Gb, which actually may not be sufficient to store all of the data associated with this project at one time, and some will have to be archived.

The claim file has an indication of the Warranty Concern Code (WCC) and the part number that was to blame. The WCC is an automotive component system, while the part number would be the part that actually failed. There are two vehicle files. The first is a thorough description of the vehicle, including engine code, vehicle identification code (VIN), and other details about the vehicle. The other contains a long cryptic description of the vehicle, and its VIN. This file is used to extract a very short character string, about the body style.

The input files are received as SAS data sets (through the SAS spawner). Because the data is updated every month, a new download is required monthly. There is a staff at the AWS site that maintains the data and one person that is responsible to initiate the process to send the information to our machine. A script is run at the AWS site, which utilizes the network address of our machine. Once the files have arrived, a user on our machine invokes SAS and runs a script to begin processing the data that has been received. It has proved expedient to receive a full download of all the columns in the database, and delete extra columns in SAS, rather than to perform filtering at the source. The AWS group maintains an interface to allow querying against the data which is periodically used to verify the results that we generate.

During the validation cycle of WPS, there was concern about the numeric calculations performed in the program. This process was complicated by the fact that the data is updated on a monthly basis. Therefore, a monitoring process was instituted. For each model year, the size of the initial data files is logged, the date of last modification, and observation counts at various times in the processing.

LOGIC INPUT FILES

Warranty expenses are assigned to various subsystems or automotive component systems. This is accomplished by defining various items. Each of the six divisions has its own item definitions. The system is very flexible in that the item definitions are provided as csv files (generated in a spreadsheet). These values are read in, and assigned to macro variables, which drive the processing. An added complication is that negative logic may be used (all parts except the specified value). For this reason, a qualifying phrase is constructed which is utilized in a SAS DATA step, when every row of the data set is examined. This qualifying phrase has also proved useful for displaying on a web page, thereby facilitating user validation. That is because of the fact that there can be up to 19 part numbers, and the input spreadsheet is typically so wide as to make review tedious. Logic is used to translate the vehicle code used in this step to match the vehicle names used in the claim and vehicle files.

There are several types of items. First is the 'A' type that specifies which WCCs are included in that item. The next is a 'P' type that has some sort of qualification, and uses a partial mapping of the WCC. This sort of rule uses a WCC and optionally a vehicle or a part number. This allows the customer to map into a specific part that one particular vendor might supply. The final category is 'SS' for sub assembly. This can be either type, and the program either assigns it to a particular category, or ignores it if it is invalid (lacks a WCC). One of the divisions (Chassis) is more complicated, and includes other fields for qualification, namely Engine code, transmission code, driveshaft, assembly plant, body style code, vehicle, and part number.

The item definitions are accomplished by the macro dodivs, which loops through the divisions, and runs one macro to read the customer-supplied csv files, one to translate them into two different data sets corresponding to the two different item types. These datasets contain the phrase to specify the item. The Chassis division would also get an additional dataset that contains the rules used for constructing installed volumes. These rules are the item definitions, ignoring the vehicle and part number qualifications (for the installed volume report). These datasets are then used to construct web pages, so the users can validate the item definitions.

The code to read the csv files is rather straightforward, since the fields are delimited by commas. The construction of the qualifying phrase is done in the following code, which handles the cases of negative logic and missing fields.

The macro proc2 defines both the positive and negative logic, by constructing an IN() phrase and NOT IN() for the variable qual. It is called for several different columns, and if the column is blank, the variable is not included. The macro proc4 is used to insert the vehicle name if it exists. A separate macro is used because the vehicle name may have a '-', which is the switch for negative logic.

```
%macro proc2(val);
  IF &val NE '' THEN DO;
    pstr=' ';
    nstr=' ';
    flg=0;
    len=length(&val);
    do i=1 to len;
      c=substr(&val,i,1);
      select(c);
        when(' ') do;
          if flg=0 then pstr=trim(pstr) ||
            ", ";
        else nstr=trim(nstr) || ", ";
        flg=0;
      end;
      when('-') flg=1;
      otherwise do;
        if flg=0 then pstr=trimn(pstr) || c;
        else nstr=trimn(nstr) || c;
      end;
    end; /* end select */
  end; /* char loop */
  if trim(pstr) ^= ' ' and trim(nstr) ^= ' '
  then
```

```
    qual=trimn(qual)||" and &val in
    (''||trim(pstr)||'" and &val not in
    (''||trim(nstr)||'"");
  if trim(pstr) ^= ' ' and trim(nstr) = ' '
  then
    qual=trimn(qual)||" and &val in
    (''||trim(pstr)||'"");
  if trim(pstr) = ' ' and trim(nstr) ^= ' '
  then
    qual=trimn(qual)||" and &val not in
    (''||compress(nstr)||'"");
  end; /* nonblank val */
```

```
%mend proc2;
```

```
%macro proc4(val);
  if &val ^= '' then
    qual=trimn(qual)||" and &val in
    (''||trim(&val)||'"");
%mend proc4;
```

```
data rulesP
(KEEP=GRPITEM ITEM DESCRIPT qual
wcc_str);
  SET NAOSUP.&div;
  length pstr nstr $120.;
  length wcc_str $40.;
  length qual $200.;
  *
  vehicle=translate(trimn(vehicle),'+',' ');
  wcc_str=wcc;
  if rec_type='P' then do;
    qual='';
    %proc2(eng_cd);
    %proc2(trans_cd);
    %proc2(drvshft_);
    %proc2(plant_cd);
    %proc2(bod_cab_);
    %proc4(vehicle);
    %proc2(prt_num_);
    if qual ^= '' then do;
      qual=substr(qual,5);
      output;
    end; /* qual exists */
  end; /* rec_type P */
```

The vehicle definition is by a three-character vehicle name (SE2VEHCD). There is a SAS data set in the system that maps these vehicle codes to the vehicle names used in the rest of the system. In the past, a data set merge was used to accomplish this. If the user specified an invalid vehicle, then the vehicle field would be left blank. This would give the erroneous result of applying the item to all vehicles. The following logic has been incorporated into the program to report on invalid vehicle names, and remove those from the item definitions. The invalid vehicle codes are listed in the log, so that the user may take appropriate action.

```
PROC SORT DATA=NAOSUP.VEHMAP98;
  BY SE2VEHCD;
RUN;
```

```
%put delete any invalid vehicles;
proc sql noprint;
  create table temp1 as
  select distinct se2vehcd
  from naosup.&div
  where se2vehcd = ' ' or se2vehcd
  in (select se2vehcd from
  naosup.vehmap98);
```

```

        create table temp2 as
        select * from naosup.&div
        where se2vehcd not in (select
se2vehcd from temp1);

quit;

data _null_;
    set temp2;
    put 'invalid se2vehcd: ' se2vehcd;
run;

proc sql noprint;
    delete * from naosup.&div
        where se2vehcd in (select se2vehcd
from temp2);
quit;
%if &sqllobs ^= 0 %then %put deleted
&sqllobs invalid vehicle codes;

%put merging with vehicle file;

DATA NAOSUP.&div ;
    MERGE naosup.&div (IN=A)
NAOSUP.VEHMAP98 (IN=V);
    BY SE2VEHCD;
    IF A;

RUN;

```

DATA PROCESSING

The system is divided into macros. One macro oversees the entire process and allows the user to control program execution. It was considered to use data files to drive the processing, but because of the SAS experience of the customer charged with maintaining the system, the following macro was used. The file is laid out in a format that facilitates modification.

```

%macro runnao;
%let drv=D;
%let dr2=D;
%let srcloc=&drv:\NAOSOURCE;
%let yrnum=0;
options nomprint nomlogic nosymbolgen;
options noxwait;

OPTIONS MAUTOSOURCE;
OPTIONS SASAUTOS = (
    "!SASEXT0\SASAUTOS"
    "&srcloc")
    ;

%global yrmess1 yrmess2 yrmess3
yrmess4 yrm3ss5 yrmess6;
%let supd=&drv:\AWSNAOVisteonSupport;
%let inpd=&drv:\NAOINPUT;
libname naosup "&supd";

/*****
* set variables and run
*
*****/
%let outd=&drv:\Naoutput;
%let clmloc=&dr2:\Claims;
* input item specifications for each
division;
%dodivs;

* customize variables;
%let tisval=3;

```

```

%let cutoffdt=30Apr99;

*delete quality dataset if it exists;
%let qfilnm=&clmloc\qual&tisval.m.sd2;
%put quality file name is &qfilnm;
data _null_;
    stat=fileexist("&qfilnm");
    if stat =1 then do;
        put 'Deleting old quality report';
        call system("del &qfilnm");
    end;
run;

* customize model years;
%runmdlyr(1994);
%runmdlyr(1995);
%runmdlyr(1996);
%runmdlyr(1997);
%runmdlyr(1998);
%runmdlyr(1999);

%qualrep;

%put Run of NAO data complete;
%put Step Year Time;
%do i = 1 %to &yrnum;
    %put &i.. &yrmess&i;
%end;
%mend runnao;

```

The macro sets up the environment, including the search path. Global macro variables are defined so that a concise report will be generated in the log. This is advisable, since the amount of processing requires an extensive time to run, so the operator will typically return to the computer periodically to review progress. A program switch is used to define the time in service (TIS) flag. If the item data has changed, the user can execute the macro to read the new definitions, by uncommenting the statement that executes dodivs.

The cut off date is significant. Every month, a new data file is received, which has a cut off date of the last day of the preceding month. This is left as an input variable, as sometimes the file is not updated as anticipated. Time variables are generated, and outputted, to assist with monitoring of program progress. Each model year is controlled by the call to runmdlyr, and maintained by commenting or uncommenting the various lines with model years. This is significant, since typically the entire array of six model years cannot be transferred in one weekend, so the user needs to control these calls in a phased approach. Finally, the quality report is produced by combining all of the model years included in the present run.

The following steps are executed for each model year in the macro runmdlyr. The first task is to verify that necessary input files exist. This is done using a SAS function, and macro variables to specify file locations. If any file doesn't exist, that model year processing is skipped.

```

data _null_;

stat1=fileexist("&clmloc\&clmin..sd2")
;

stat2=fileexist("&vehloc\&vehtbl..sd2")
);

```

```

stat3=fileexist("&vehloc\voi&my..sd2")
;
if stat1=0 then put "claims file
doesn't exist!";
if stat2=0 then put "vehicle file
doesn't exist!";
if stat3=0 then put "voi file doesn't
exist!";
stat=stat1+stat2+stat3;
call symput('stat',stat);
run;
%if &stat = 3 %then %do;

```

Data is captured regarding the modification date of the input files, and file size. As mentioned previously, the input files are updated on a monthly basis, and sometimes there is concern as to the integrity of the data. The following macro proved useful in monitoring the number of observations in certain data sets. It supports logging the number of observations in a given data set, and is called with the library, and data set names.

```

%macro cnt2log(libn,dsn);
option nonotes;

proc sql noprint;
select nobs into: cnt
from dictionary.tables
where libname=upcase("&libn") and
memname=upcase("&dsn");
quit;

data _null_;
file outlog mod;
now=datetime();
f1=put(now,datetime.);
put f1 "Obs in &libn.&dsn: &cnt";
run;

option notes;
%mend cnt2log;

```

Since the completion of this project, it has come to my attention that the method above includes the observations that have been marked for deletion. A better method to obtain the correct number of observations is as follows.

```

%let id_num =
%sysfunc(open(&libn.&dsn));
%let cnt =
%sysfunc(attrn(&id_num,nlobs));
%let clz_dsn =
%sysfunc(close(&id_num));

```

The vehicle name is added to the vehicle file. This name is dependent on more than one column of the input. Since the data is subsequently joined on this field, consistency is critical, and a macro is used for this purpose on both the claims and vehicle file. Certain filtering is done on the vehicles. Basically, if the vehicle has not been sold within a certain range of the date of production, it is not used in the analysis. Currently, this is 60 days. A list of the valid VINs is constructed for claims filtering.

The claims file is loaded, and the vehicle column is added, and several fields are deleted. Several other fields are added to the claims file that are present in the vehicle files. Finally, the claims are deleted that are

against vehicles not contained in the list of valid vehicles produced from the vehicle file.

Next the six individual divisions are processed. For each division, the item definition files are processed in order to determine which items are present in the claims file. First macro variables are defined for the item definitions, and then the macro item is called in a DATA step. The macro basically steps through the claim file, and constructs a long IF statement to determine which item is responsible.

```

%MACRO ITEM;
* all items;
%IF %EVAL(&P1CT > 0) %THEN
%DO P = 1 %TO &P1CT;
%IF &P=1 %THEN %STR(IF WCC_CD =:
"&&P1WC&P" THEN DO;ITEM = "&&P1TM&P" ;
OUTPUT; END;) ;
%ELSE %STR(ELSE IF WCC_CD =:
"&&P1WC&P" THEN DO; ITEM = "&&P1TM&P"
; OUTPUT; END;) ;
%END; /* p1 loop */

* qualified items;
%IF %EVAL(&P2CT > 0) %THEN
%DO P = 1 %TO &P2CT;
%IF &P=1 %THEN %STR(IF WCC_CD =:
"&&P2WC&P" and &&p2QU&p THEN DO;ITEM =
"&&P2TM&P";OUTPUT; END;) ;
%ELSE %STR(ELSE IF WCC_CD =:
"&&P2WC&P" and &&p2QU&p THEN DO;ITEM =
"&&P2TM&P";OUTPUT; END;) ;
%END; /* p2 loop */

%MEND ITEM;

```

The macro naodiv is called which processes the claim data. It determines how many months will need to be included in the output files, adds placeholders for the missing month values, accumulates repairs and costs, and creates a dataset of vehicle counts by month of production. PROC SUMMARY is used to count the vehicles by month of production, as well as the number of repairs, and total costs.

The macro naotis is called which calculates the month in service values, and merges the repair and cost files with the vehicle counts to create a file with the repair counts, total costs, and calculate the repairs per 1000, and cost per unit values.

The macro naoout is called which produces the output files. There are five created for each division, namely vehicle counts, repair counts, total cost, repairs per 1000, and cost per unit. The dataset is transposed on the month of production. These is done for four different outputs, repairs, costs, repairs per 1000 vehicles, and cost per unit. A typical section of code follows. Macro variables are used to capture the values for the specific month of production, and then included in the report after the transpose operation. The variable NAME was added to the data set to facilitate this operation.

```

* Transpose the Claim Volume data
pivoting on Month of Production;

PROC TRANSPOSE DATA=CLAIMS.NA0CT
OUT=CLMTPR;
WHERE TIS60AWS=&tisval;
BY VEHICLE ITEM;

```

```

        /*BY VEH_LINE ITEM;*/
        ID REPMOP;
        VAR repairs;
RUN;

PROC CONTENTS DATA=CLMTPR OUT=REPCONT
NOPRINT;
RUN;
DATA REPCONT;
    SET REPCONT(KEEP=NAME);
    IF NAME =: 'R';
RUN;
DATA _NULL_;
    SET REPCONT;
    CALL SYMPUT('UNIT' || LEFT( PUT(
    _N_,3.)),NAME);
    CALL
SYMPUT('UOUT' || TRIM(LEFT(PUT(_N_,3.)))
, substr(NAME,2));
    CALL SYMPUT('CT',LEFT( PUT(
    _N_,3.)));
RUN;
data _null_;
    SET CLMTPR          end=EFIEOD;
    MDLYR="&mdlyr";
    %let _EFIERR_ = 0; /* clear
ERROR detection macro variable */
    %let _EFIREC_ = 0; /* clear
export record count macro variable */
    FILE "&odir\REPAIRS.CSV"
    LRECL=32767;
    IF _N_=1 THEN DO;
        PUT ' _VEHICLE' ',' 'ITEM' @;
        %DO I = 1 %TO &CT;
            PUT ',' '&UOUT&I' @;
        %END;
        PUT;
    END;

    DO;
        EFIOUT + 1;

        * Process character variable
for imbedded delimiter or quotes;
        PUT VEHICLE @;
        PUT ',' 'ITEM $ +(-1) @;
/* no delimiter found */
        %DO I = 1 %TO &CT;
            IF &&UNIT&I=. OR &&UNIT&I='
then put ',' @;
            ELSE PUT ',' &&UNIT&I @;
        %END;
        PUT;
    END;

    If _ERROR_ then /* ERROR
detection */
        call symput('_EFIERR_',1);
    If EFIEOD then
        call
symput('_EFIREC_',EFIOUT);
run;

```

Reports are generated using total vehicle volumes to calculate the cost per unit and repairs per 1000 values. For one of the divisions (Chassis), the macro INSTVOL is called. Some of the chassis items are also calculated by the number of vehicles that actually have the item installed. This is accomplished by isolating those items that are defined by criteria other than vehicles and part numbers.

The macro scrub is called which facilitates disk cleanup. This macro will move all the temporary files

into a separate directory. Input files resided in two separate directories (vehicles and claims), and the temporary files go in an archive directory. These can be deleted after the run is completed.

Finally, there is a report that needs to span model years (Quality Model year report). In order to support this feature, the data is accumulated for every year in the program run. Because of time constraints, this report is not transposed, as are the regular output reports. This gives a problem with importing into a spreadsheet because of the size of the file.

Once, all model years have been executed, the qualrep macro accumulates this information into a report. While individual model years can have more than twelve months of data, the quality model year is reformulated for a calendar year. The values for each month of production are accumulated, which may include multiple model years.

OUTPUT FILES

The output files are number of vehicles, repair incidents, cost, cost per unit, and repairs per 1000 vehicles, which are presented as csv files. The beauty of the system is that while the input files are very large, the output files are very manageable. Output files are accessed by the users over the network. Most of the files are transposed in SAS, so that the data is easily accessible by the users, who would typically be interested in only one output file. These reports have columns of, vehicle, item, and whatever months of production are present of the reporting variable. This is important, as the reports will typically exceed the 16,000 rows that can be read into their spreadsheets if they were not transposed.

A final step is to compress the output files. This will allow the five output files to be transferred as one file, and saves disk space. The code to accomplish this is as follows.

```

*compress files for distribution;
%put compressing files;
%let pgm=c:\ftp\pkzip;
%let var1=&outd\data&my\&&div&J;
%let var3=&&div&j..&my..zip;
%put Base directory: &var1;
%put New file: &var3;
data _null_;
    call system("&drv:");
    call system("cd &var1");
    call system("&pgm &var3 *.csv");
run;

```

There is another version of this system that processes European data. This version is not as robust, and the data is smaller in scope. The main difference is that the program execution is driven by input csv files. The output files are not transposed, and the customer parses the output files for publishing on a web site.

CONCLUSION

The system supports the customer need to process extremely large claim files in order to evaluate the warranty experience into a user-determined level of detail. The process is flexible in that the user can change these item definitions at any time.

The output of the system is small, manageable files

that are easily imported into spreadsheets for further analysis. The output files are available through a web interface, and the item definitions that the program uses are also displayed on web pages.

The customer found the system invaluable, since running the existing system (AWS) would have required a prohibitive amount of runs.

Future directions may include variation of the item definitions and vehicle mappings for different model years. Another option would be to construct a macro to transpose the quality report and installed vehicles report

ACKNOWLEDGMENTS

Quinton Ritz, formerly of Trillium Teamologies, Royal Oak, MI for constructing much of the SAS code for duplicating the calculations in the customer system.

Paul Gorrell, of Westat, Inc., Rockville, MD for describing a better method of determining the correct number of variables in a SAS data set, as posted on the SAS-L discussion group.

CONTACT INFORMATION

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

Tim Vorce
Trillium Teamologies
219 S. Main Street, Suite 300
Royal Oak, MI 48067
Work Phone: (248) 584-2080
Fax: (248) 584-2250
Email: tvorce@trilliumteam.com