MWSUG 2011 - AD15

An EXCEL- ent Method to %INCLUDE SAS Code

Irvin Snider, Assurant Health, Milwaukee, WI

Abstract

There are plenty of times in my job when I get instructions from someone to update some SAS IF-THEN-ELSE logic and I think to myself: "If this guy can type the requirements into an Excel spreadsheet, why can't he just update the code himself?"

Well, why can't he?

In this paper, I will present a method using PROC IMPORT, a TEXT file, an %INCLUDE statement, and PROC PRINTO for the SAS-illiterate-Excel-spreadsheet-user to create SAS code without even knowing it. Functions such as COMPRESS, CATS, and TRANWRD that will assist in making the process idiot-proof will also be presented.

Introduction

This paper is a result of a problem that I had at work.

I work for an insurance company that provides health insurance for individuals. These individual medical insurance policies have their premiums recalculated on an annual basis. The factors that are used in renewal premiums are calculated by the renewal actuaries. The actuaries create the factors that are used in a rating algorithm that ultimately produces a renewal premium for our policyholders. My job, as a member of the Implementation Team, is to place these factors into the system so that our policyholders get the premium that the actuaries want. This new renewal premium comes into effect on the policies' anniversary date.

The Implementation Team monitors the renewals through the use of a Daily Renewal Report. The important data on this Report for the purposes of this paper are the Policy number, the Form number, the State where the policy was issued and the renewal date.

The actual Daily Renewal Report is an Excel workbook that contains several worksheets delivered via email. It is produced by a SAS job that the Implementation Team maintains.

There are times when the Actuarial Department wants to delay the renewal on a group of policies. For whatever reason, we do not want these policies to get the new renewal premium. This happens with enough regularity and variety that we gave these "hold" requests some attention and created a SAS macro that programmatically identifies these policies. The "Hold Policies" are identified and the policy numbers are transferred to a separate worksheet in the Daily Renewal report. These policy renewals are then delayed by another process.

The HOLD_pols macro gets updated whenever new requirements come from the actuaries.

And this is where the problem lies. Although the actuaries are technocrats, they are not SAS coders. This means that even though the changes that they request may be simple IF-THEN-ELSE statements, they are petrified to go into the HOLD_pols macro themselves and make a simple change. So, our team gets their instructions and we have to make the changes to the macro code for them.

HOLD_pols Macro

In the Beginning

Before we made some improvements to the process, the instructions from actuarial arrived via email. They looked something like this:

From:	Actuarial
To:	Irv Snider
Subject:	HOLD policies

Please hold TX forms 625 644 670 beginning 5/1/2011.

Hold WI R778 from 6/1/2011 through 12/14/2011.

Hold PA N/P100 from 5/1/2011 to 7/14/2011 and RI N/P100 from 5/1/2011 to 7/14/2011.

Thank you
The Actuarial Department

Someone on our team would then turn these instructions into SAS code. It's fairly straightforward coding: turn the sentences in the email into SAS IF-THEN-ELSE logic.

```
IF ISSTATE = 'TX' AND FORM IN ('0625' '0644' '0670') AND RNL_DT >= '01MAY2011'd THEN
OUTPUT;

ELSE IF ISSTATE EQ 'WI' AND FORM = 'R778' AND RNLDATE >= '01JUN2011'd AND RNL_DT <
'15DEC2011'd THEN OUTPUT;

ELSE IF ISSTATE IN ('PA' 'RI') AND FORM IN ('N100' 'P100') AND '01MAY2011'd <= RNL_DT
< '15JUL2011'd THEN OUTPUT;
```

The Move to Excel

We decided that it would be better to make the Hold Policies process more formal so that we could track any changes to the logic that we made. So we placed the requirements in an Excel spreadsheet on a shared directory.

At first the Excel spreadsheet was the same as the emails. The spreadsheet contained a jumble of words and numbers that looked not unlike the unstructured message reproduced in the above paragraph.

KS eff 4/1/11: 0253, H253, 0376, H376
CO eff 1/1/11: 0253, H253, 0376, H376
DE eff 1/1/11: 0770, 0880, 0225, M225, 0390, M390

Eventually we agreed that the spreadsheet would look like the one copied below.

ISSTATE	FORM	BEG_RNL_DT	END_RNL_DT
'PA' 'RI'	'N100' 'P100'	5/1/2011	7/15/2011
'PA' 'HI' 'NY'	'0192' 'M192' '0502' '0534'	5/1/2011	8/15/2011
'MN'	'0100' 'M100' 'N100' 'P100' 'T100'	6/1/2011	1/1/9999
'MN'	'V100' '0192' 'M192' '0502' '0534'	6/1/2011	1/1/9999
'TX'	'0625' '0644' '0670'	5/1/2011	1/1/9999
'WI'	'R778'	6/1/2011	12/15/2011
'ID' 'KS' 'ND' 'SD' 'WV'	'0100' 'M100' 'P100' 'V100' 'T100'	5/1/2011	6/15/2011

We wanted the information in the cells to be as close to SAS code as possible to make the coding easier.

We agreed that someone in the Actuarial Department was responsible for the upkeep of the spreadsheet.

The first row of the spreadsheet contained the variable names.

The five variables that are used to determine the Hold Status of a Form are:

ISSTATE	Issue State
FORM	The Contract or Form Number
BEG_RNL_DT	The date that Actuarial wanted the form to begin being on hold
END_RNL_DT	The date that Actuarial wanted the form to stop being on hold
RNL_DT	The Renewal Date of the policy

We would now get an email from Actuarial telling us to update the HOLD_pols macros using the information listed in the Excel spreadsheet.

This spreadsheet was an improvement to the unstructured format of the emails but we still had to update the HOLD_pols macro ourselves and had to rely on Actuarial to notify us when the macro needed updating. But things were a bit tidied up.

The Opportunity

However, it seemed that the process could be further simplified. Sure, Actuarial made it easy for us to copy and paste the variables into the code but there was room for error. If we copied and pasted from Excel into SAS, we could still make a mistake. We needed verification from Actuarial that the code was exactly what they wanted so the emails continued to fly back and forth.

This is when I thought that it would be nice if we could make the SAS code that created the Daily Report check the Excel spreadsheet each day for any changes and update the code programmatically. Then our team could go on to do bigger and better things. There would be less emails flying about and less chance for error.

So we decided to try to do this upkeep with a program.

An Observation

It was noted that the Excel spreadsheet always resulted in this logic:

IF the ISSTATE is in (‘XX’) and the FORM is in (‘XXXX”) and RENEWAL DATE is between BEG_RNL_DT and END_RNL_DT then output the policy number.

Nothing earth-shattering here. The variables might change at the whim of the Actuarial Department but the logic structure is hard code.

The General Idea

The general idea of what we wanted to do was to import the variables from the shared Excel worksheet into SAS. We would then manipulate the variables with some hard code to create a character string that resembled an IF-THEN-ELSE statement, and, afterwards, send it to a text file using PROC PRINTTO and PROC REPORT. Finally, we would bring the character string back from the text file into a DATA step with an %INCLUDE statement.

When Actuarial adds another line of variables to the Excel spreadsheet, the job will be able to update the code based on the changed version of the spreadsheet. The same thing happens when a line is removed from the spreadsheet. Actuarial could add and subtract variables at will without even having to tell our team what they were doing. The code would do the work for us.

The Complete Proof of Concept Code

The End is Near

At this point, I am going to reproduce the code that I used to prove the concept.

It is pretend data. The state abbreviations are real but the rest of the data is bogus.

The code as it is written here would not work. You would need to create an Excel spreadsheet that contains the input variables and also a place for the text file to reside.

I’ll follow up with some comments and sample output afterward.

Here’s Johnny!

```
options firstobs= 1 obs = max linesize=256;

filename text "\\...\hold_text.txt" LRECL= 32767;

proc import datafile="\\...\Renewals_on_hold.xls"

out=work.kk1
dbms=excel2000
replace;
getnames = yes;
sheet="Get_This_Sheet";
run;

data work.kk;
       set work.kk1;
```

```
keep ISSTATE FORM BEG_RNL_DT END_RNL_DT;

length isstate_fin  $24;
length Form_fin     $34;

ISSTATE       =strip(upcase(compress(ISSTATE, ,"pt")));
FORM          =strip(upcase(compress(FORM, ,"pt")));

cats_Form    = Cats( "'" , TranWrd(FORM, " " , "' '" ),"'" );
comp_Form    = compress(TranWrd(cats_Form, "''","" ),' ');
Form_fin     = strip(TranWrd(comp_Form, "''" , "' '" ));

cats_isstate = Cats( "'" , TranWrd(ISSTATE, " " , "' '" ),"'" );
comp_isstate = compress(TranWrd(cats_isstate, "''","" ),' ');
isstate_fin  = strip(TranWrd(comp_isstate, "''","' '" ));

drop
       FORM
       cats_form
       comp_form
       ISSTATE
       cats_isstate
       comp_isstate;

Format
       part_1              $19.
       part_2              $13.
       part_3              $16.
       part_4              $15.
       part_5              $13.

       beg_tick            $2.
       d_date              $1.
       beg_paren           $2.
       end_paren           $1.

       beg_RNL_DT_char     $9.
       end_RNL_DT_char     $9.;

BEG_RNL_DT_char = put(BEG_RNL_DT,date9.);
END_RNL_DT_char = put(END_RNL_DT,date9.);

drop
       BEG_RNL_DT
       END_RNL_DT;

length PART1 $ 46;
length PART2 $ 50;
length PART3 $ 28;
length PART4 $ 27;

part_1              = ' Else if ISSTATE in ';
part_2              = ' and FORM in ';
part_3              = ' <= RNL_DT ';
part_4              = ' < RNL_DT ';
part_5              = ' then output;';

end_tick            =      "'";
beg_tick            =      " '";
d_date              =      'd';
beg_paren           =      ' (';
end_paren           =      ')';


call catt(PART1, part_1, beg_paren, isstate_fin, end_paren);
call catt(PART2, part_2, beg_paren, Form_fin, end_paren);
call catt(PART3, ' and',beg_tick, beg_RNL_DT_char, end_tick, d_date, part_3);
```

```
       call catt(PART4, ' <', beg_tick, end_RNL_DT_char, end_tick, d_date);

       PART1 = compbl(PART1);
       PART2 = compbl(PART2);
       PART3 = compbl(PART3);
       PART4 = compbl(PART4);

       length final_out $ 200;

       call catt(final_out, PART1, PART2, PART3, PART4, PART_5);

       keep   final_out;
run;

options nocenter nodate nonumber;
title;

proc printto print="\\...\hold_text_igs3.txt" new;
run;

proc report data = work.kk nowd noheader;
       column
       final_out;
run;

proc printto;
run;

options firstobs= 1 obs = max;

DATA work.Daily_Renewal_Report_Sample;
       input  @1  policy   $10.
              @12 isstate  $2.
              @15 FORM     $4.
              @20 RNL_DT   mmddyy10.;

       Format RNL_DT       mmddyy10.;

datalines;
0007032430 TX N100 06/11/2011
0007103020 MN 0100 06/05/2011
0007203102 WI R778 07/20/2011
0007304292 HI 0192 07/05/2011
0007304292 RI N100 06/30/2011
;

run;

data work.Viola;

       set work.Daily_Renewal_Report_Sample;

       IF isstate = 'AA' THEN OUTPUT;

       %include text;

run;
```

Comments on Code

Wrapping of the Text

The LINESIZE=256 indicates that this is the maximum amount of character space that one has to work with in the text document. This is a limitation that can cause problems with the text output. This wrapping can make the job fail.

FILENAME TEXT

This is the location of the text file. We will place our IF-THEN-ELSE character string in this location. We will also retrieve the text in the final DATA step.

```
filename text "\\...\hold_text.txt" LRECL= 32767;
```

Import the Spreadsheet

PROC IMPORT will do the work for us. We set GETNAMES to “YES” to get the variable names from the spreadsheet. We will use these variable names in our IF-THEN-ELSE logic and the ultimate character string.

```
proc import datafile="\\...\Renewals_on_hold.xls"

out=work.kk1
dbms=excel2000
replace;
getnames = yes;
sheet="Get_This_Sheet";
run;
```

Format the Character Strings

A series of functions were used to eliminate potential keying errors in the Excel spreadsheet and to turn numbers into characters.

This series of functions will remove any extraneous punctuation and spaces. COMPRESS is a valuable tool used to remove unwanted characters. The COMPRESS function modifier ‘P’ removes all punctuation and the modifier ‘T’ removes all spaces. UPCASE capitalizes all the letters. STRIP removes leading and trailing blanks.

```
ISSTATE =strip(upcase(compress(ISSTATE, ,"pt")));
```

The next three lines of code will ensure that the variable isstate_fin will be a string of two character state names separated by commas and surrounded by single tick marks with a single space separating each state.

```
cats_isstate = Cats( "'" , TranWrd(ISSTATE, " " , "' '" ),"'" );
comp_isstate = compress(TranWrd(cats_isstate, "''","" ),' ');
isstate_fin  = strip(TranWrd(comp_isstate, "''","' '" ));
```

A PUT function will turn the dates into a character string with a DATE9 format.

```
BEG_RNL_DT_char = put(BEG_RNL_DT,date9.);
```

Hard Code Segments of the String

This may be an example of coding for job security and I cannot argue with that. I admit that this section probably adds some additional unnecessary character substitution. But it works for me.

```
part_1       = ' Else if ISSTATE in ';
part_2       = ' and FORM in ';
part_3       = ' <= RNL_DT ';
part_4       = ' < RNL_DT ';
part_5       = ' then output;';

end_tick     =      "'";
beg_tick     =      " '";
d_date       =      'd';
beg_paren    =      ' (';
end_paren    =      ')';
```

CALL CATT Call Routine

We concatenate the “parts” together using the CALL CATT call routine. CALL CATT removes only trailing blanks before concatenating character arguments. The resulting character variable is included in the CALL.

```
call catt(PART1, part_1, beg_paren, isstate_fin, end_paren);
call catt(PART2, part_2, beg_paren, Form_fin, end_paren);
call catt(PART3, ' and',beg_tick, beg_RNL_DT_char, end_tick, d_date, part_3);
call catt(PART4, ' <', beg_tick, end_RNL_DT_char, end_tick, d_date);
```

<u>COMPBL Function</u>

The COMPBL function will keep a unique blank between words. The COMPBL function removes multiple blanks in a character string by translating each occurrence of two or more consecutive blanks into a single blank.

```
PART1 = compbl(PART1);
PART2 = compbl(PART2);
PART3 = compbl(PART3);
PART4 = compbl(PART4);
```

<u>FINAL OUT</u>

Finally we create one last character string named final_out by concatenating the various “parts” using the CALL CATT call routine.

```
length final_out $ 200;

      call catt(final_out, PART1, PART2, PART3, PART4, PART_5);

      keep final_out;
```

A RUN statement ends the DATA step. WORK.KK consists of one variable named final_out.

<u>PROC PRINTTO</u>

We send the final_out character string to a text file using PROC PRINTTO and PROC REPORT.

Page numbers, dates and titles will get in the way so we set out OPTIONS to NO.

```
options nocenter nodate nonumber;
title;
```

A text file is created and is overwritten every time the job is run. We do not want the text file to be appended.

```
proc printto print="\\...\hold_text_igs3.txt" new;
run;

proc report data = work.kk nowd noheader;
      column
      final_out
      ;
run;

proc printto;
run;
```

<u>Sample Daily Renewal Report is Created</u>

The code listed here will create a sample data set that resembles the Daily Renewal Report.

```
options firstobs= 1 obs = max;

DATA work.Daily_Renewal_Report_Sample;
      input  @1  policy   $10.
             @12 isstate  $2.
             @15 FORM     $4.
             @20 RNL_DT   mmddyy10.
             ;

      Format RNL_DT       mmddyy10.;
```

```
datalines;
0007032430 TX N100 06/11/2011
0007103020 MN 0100 06/05/2011
0007203102 WI R778 07/20/2011
0007304292 HI 0192 07/05/2011
0007304292 RI N100 04/30/2011
;

run;
```

%INCLUDE

We SET the sample Daily Renewal Report into a data set named Viola.

The IF ISSTATE = ‘AA’ statement will always be ignored but it starts off a series of ELSE IF statements that will be read into the DATA step via the %INCLUDE statement. It is always best to employ ELSE IF statements for efficiency‘s sake and using this method makes it easier to code the character string.

When you submit a %INCLUDE statement, it reads an entire file into the current SAS program that you are running and submits that file to the SAS system immediately. The default record length that is used by the %INCLUDE statement is 256 characters.

```
data work.Viola;

       set work.Daily_Renewal_Report_Sample;

       IF isstate = 'AA' THEN OUTPUT;

       %include text;

run;
```

The Text File

This is what the %INCLUDE statement brings in. The text wraps in this document but it does not in the actual text.

```
Else if ISSTATE in ('PA' 'RI') and FORM in ('N100' 'P100') and '01MAY2011'd <= RNL_DT < '15JUL2011'd then output;

Else if ISSTATE in ('PA' 'HI' 'NY') and FORM in ('0192' 'M192' '0502' '0534') and '01MAY2011'd <= RNL_DT < '15AUG2011'd then
output;

Else if ISSTATE in ('MN') and FORM in ('0100' 'M100' 'N100' 'P100' 'T100') and '01JUN2011'd <= RNL_DT < '01JAN9999'd then
output;

Else if ISSTATE in ('MN') and FORM in ('V100' '0192' 'M192' '0502' '0534') and '01JUN2011'd <= RNL_DT < '01JAN9999'd then
output;

Else if ISSTATE in ('TX') and FORM in ('0625' '0644' '0670') and '01MAY2011'd <= RNL_DT < '01JAN9999'd then output;

Else if ISSTATE in ('WI') and FORM in ('R778') and '01JUN2011'd <= RNL_DT < '15DEC2011'd then output;

Else if ISSTATE in ('ID' 'KS' 'ND' 'SD' 'WV') and FORM in ('0100' 'M100' 'P100' 'V100' 'T100') and '01MAY2011'd <= RNL_DT <
'15JUN2011'd then output;
```

Viola Output

The results of the job for this sample data indicate that the code is working fine.

```
0007103020    MN    0100    06/05/2011
0007203102    WI    R778    07/20/2011
0007304292    HI    0192    07/05/2011
0007304292    RI    N100    06/30/2011
```

Full Disclosure

It seems that the general idea is doable and the code will work. However, the process does have its quirks and foibles.

As mentioned before, text wrapping is a problem. If the character string is too long, the text wraps onto the next line. This makes the job fail when the text is brought into the DATA step via the %INCLUDE statement.

On a pragmatic level, the problem was solved by forcing actuarial to abide by strict formatting rules regarding the Excel spreadsheet. We allow a maximum of five forms and five states. For the most part, this is a workable solution to the problem. Since we are limiting the number of variables that are placed in an Excel cell, we can calculate the length of the variable in the SAS code. For example, if we limit the variable isstate_fin to five states in single ticks with each having a separating space, we have a total length of 24.

I tried to program around this problem by employing macro substitution in order to save space in the text file. The text file would resemble a series of macro calls and the text substitution would take place in the final DATA step using a series of %LOCAL %LET statements but it was too much work for what it was worth. It was easier to just tell them no more than five to a cell.

Another foible is that there must be a space between the states and the forms in the Excel cells. CALL CATT, COMPRESS, CATS and STRIP all do a fine job in removing spaces. However, if the state variables are separated by asterisks or some other character and not spaces, there will be an error. There is probably a solution to this problem and it would be nice to have given the propensity for keying errors but again it is easier to just tell them to include spaces.

Finally, make sure that your SAS EG Options Results > General include text output.

Conclusion

It is possible for SAS code to be written more or less automatically with information residing in an Excel spreadsheet given certain circumstances. Ideally, the SAS code is updated and modified via an Excel spreadsheet without any intervention of the SAS programmer.

This method works best when the variables change but the logical structure remains the same.

I have used a similar method for creating INPUT statements and INFORMATS in other SAS jobs. I have also used this method to modify and update region variables based on zip codes. The variables in the spreadsheets have changed but the code has not. Best of all, I do not even know when changes are made.

References

Cody, Ron. 2004. *SAS® Functions by Example*, Cary, NC: SAS Institute Inc.

Hadden, Louise S. 2009, “Purrfectly Fabulous Feline Functions”, NESUG, 2009.

SAS Institute Inc., *SAS® Language Reference: Dictionary, Version 8,* Cary, NC: SAS Institute Inc., 1999.

SAS Institute Inc., *SAS® Procedures Guide, Version 8,* Cary, NC: SAS Institute Inc., 1999.

SAS Institute Inc., “Reading Delimited Text Files into SAS®9, Cary, NC: SAS Institute Inc., TS DOCS, TS-673.

Worden, Jeanina. 2007, “Hanging On By a String? Using Functions To Untie Text Strings”, SAS Global Forum 2007, Paper 035-2007.

Acknowledgements

Kari Koldeway is the inspiration for this problem and its solution. This paper would never have been completed without her refusing to learn SAS. Thank you for making my professional life interesting.

Thanks to Emma Snider for proofreading, spell checking, love and suggestions.

Thanks to Joe Roma for his suggestions and refreshments.

Thanks to Deanna Nguyen for her encouragement and assistance in my ever continuing pursuit to improve my communication skills.

Contact Information

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

Irvin Snider
Assurant Health
501 West Michigan
Milwaukee, WI 53201-3050
Phone: 414-299-6979
Fax: 414-299-8043
E-mail: irv.snider@assurant.com
Web: http://www.linkedin.com/in/irvinsnider