SESUG 2023 Paper 188

Pol-y-mor-phism in SAS®, Or, Good Programmers are Lazy

David B. Horvath, MS, CCP.

ABSTRACT

This session reviews techniques for implementing simple polymorphism in SAS programs. As part of an effort to productionalize a large number of models with multiple variables with long time series, a datamart was created in SAS datasets to minimize impact on the corporate data warehouse. Rather than coding the inputs multiple times in the build process, a set of macros were created to define the DDL for input to each model. By including different files that define those macros, different functions were implemented: Prime, shift months, define keep string.

INTRODUCTION

Polymorphism is defined as the condition of occurring in several different forms. For computing, this refers to a feature of a programming language that allows routines to use variables of different types or have different behaviors at different times based on how they are used.

This all started when we were faced with a number of time-series models that were putting too heavy a load on the data warehouse. Each month they would grab N-months from various tables to build their input datasets. The task was to build a data-mart that would retain only the data needed (storage is always a concern) and only grab the current month's data from the warehouse (minimizing load).

So we had this large pile of models, each with different data requirements – not just fields, but different time-series lengths. Some would only be current month; some would be for many months. We also expected that data requirements over time – for individual models and new ones on board.

Essentially, we had to build data-marts for each model. Standard practice for time series data was to create variables in the form base_M1, base_M2, base_M3, etc. (where 1 was the latest month). While this might be more expensive in terms of space, it limited the impact of changing history requirements or the addition of new models.

For one model, hand coding wouldn't be a big deal. But with dozens, it was a lot of work with concerns about getting it right – and then future maintenance due to the differences (and growing over time with new models).

We also needed a way to build the initial history and update it each month.

OUR SOLUTION

I did not want to write unique code for each model because I'm lazy. And, it would be a support and maintenance nightmare.

Instead, we created a form of DDL for each of the models describing base field name, data type, and months of history. Time series data would be handled using the standard naming convention.

We determined that there were four use-cases for this data:

1. Defining the initial layout of the datamart in order to build the history

(primer_macro.sas)

2. Defining the variables to retain out of prior-month history (merge_keep_macro.sas)
3. Rename the variables from the prior month to their place in the current month (M1 from last month becomes M2 this month) (shifter_macro.sas)
4. Reverse variable renaming (this will be discussed later) (shifter_macro_reverse.sas)

One approach would've been to write descriptors for those fields and a program to create the code specific to each of the use-cases. I've written and spoken about Self Modifying code (code generating code) previously. But that required writing more programs and would present additional risk of the code-files getting out of sync with future changes (DDL and model changing but not the generated code).

These processes were going to be productionalized and executed from the enterprise production scheduler. Standard scripts were written to set environment variables and execute the SAS code.

I am focusing on the portion with SAS macros. There was also standardized code to load the scores, validation, execution timing, record counts, etc. into database tables for review and usage of the results. Each of those had DDL specific to the program output.

PROCESS OVERVIEW

At the front end, we have an ETL process to grab the necessary data out of the Data Warehouse each month. There were strong political factors for use of the enterprise ETL tool.

Error! Reference source not found. shows the front-end monthly process.

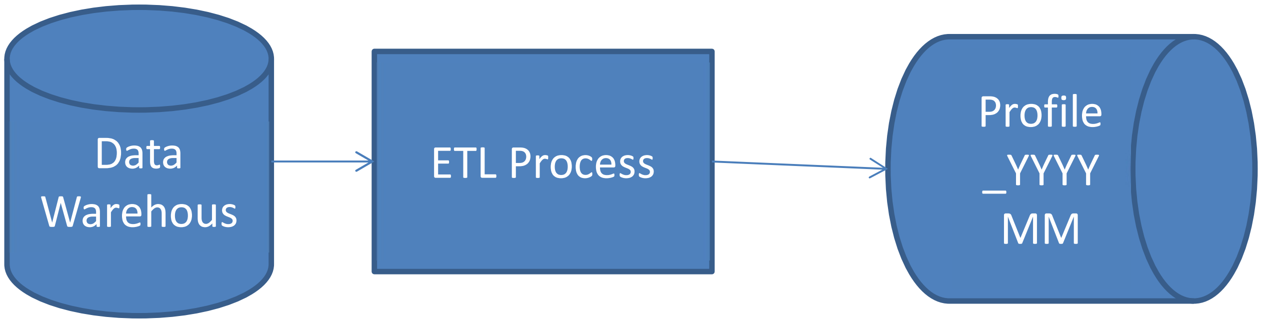

Figure 1. Front End Monthly Process

The next important part is the start of the history build – priming the datamart itself.

Error! Reference source not found.2 shows the initial datamart creation (priming) process.

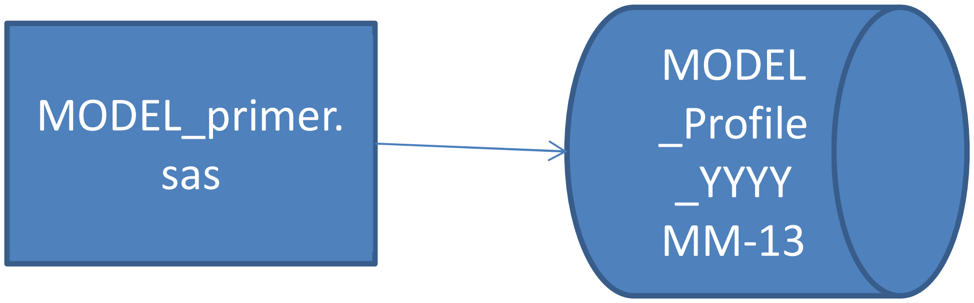

Figure 2. Initial Datamart Creation (Priming) Process

And, finally, we have the SAS processing. Put simply, This paragraph uses the PaperBody style, which uses the Verdana font, not the Arial font.

Error! Reference source not found.3 shows the main SAS processing.

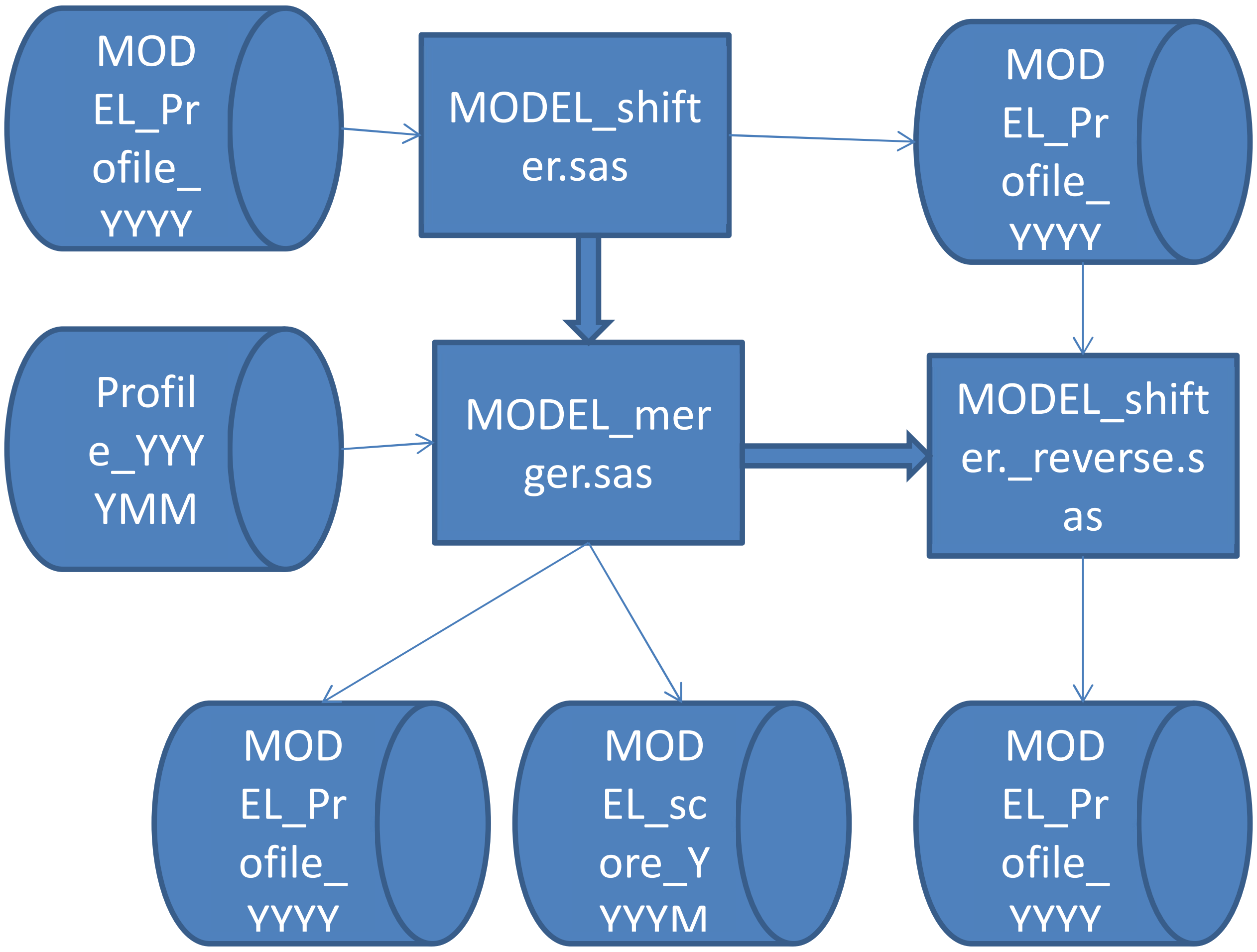

Figure 3. Primary Model/History Process

Building the Initial History

Building a historical datamart starts with the priming process (Figure 2). This needs to be run for each of the models but only one.

For each of 12 months, the ETL Process (Figure 1) is run to extract the appropriate data.

And finally, for each of the 12 months, the Primary Model/History process (Figure 3) is run. Even if a particular model did not use all 12 months of history, we ran it because we were executing these in bulk – did not want to run model A for 3, B for 6, C for 12, etc. because we figured we'd get at least one of them incorrectly.

The MODEL_profile from the prior month was run through the shifter process. Single month variables were renamed to _MX while multiple month variables were shifted to one higher month: M1 became M2, M2 became M3, M3 became M4, until Mn became Mn+1. This was performed using proc datasets: this changes the metadata in the dataset.

Another approach would have been to rename the fields on the fly in the merge statement. However, we understood that there was a processing penalty (has to occur for each record) while the dataset change occurred to the header data one time.

The MODEL_merger takes in the prior month MODEL_profile (with a keep statement that does not include _MX or Mn_1 variable names – this is how we get rid of the extra variables) and merges it with the current month Profile. This creates the current month MODEL_profile.

After the score is produced, the prior month MODEL_profile is run through another proc datasets to reverse the variable name rename process. That leaves us prepared for a rerun or if someone wants to review the data over time (auditing our process for instance).

Running This Month

For an actual processing month, the ETL Process (Figure 1) is run to extract the appropriate data.

Then the primary process (Figure 2) is run for each model. The difference between MODEL_merger and MODEL_run_score is that the run_score also executes the model code and writes out the score. This will be a bit more obvious when we look at the code.

THE DDL FILE

The following is an example DDL file for a model that includes 9 fields:

```
/* MODEL_ddl.sas edited
================================================*/
%ddl_builder_single (numeric1_m1,N,0)
%ddl_builder_single (character1_m1,C,8)
%ddl_builder_single (character2_m1,C,8)
%ddl_builder_single (character3_m1,C,8)
%ddl_builder(numericmon1_m,13,N,0)
%ddl_builder(numericmon2_m,13,N,0)
%ddl_builder(numericmon3_m,3,N,0)
%ddl_builder(charactermon1,13,C,10)
%ddl_builder(charactermon2,3,C,5)
```

There are two macro names: ddl_builder_single for one-month (updated each month) fields and ddl_builder for multiple month fields. The parameters are, in order:

1. Base variable name
2. Number of months to retain (not for ddl_builder_single)
3. Datatype: Character or numeric
4. Size of field (ignored for numeric fields).

The best way to view the behavior of this file is in the execution.

THE MODEL PRIMER

The model primer is one of the simpler uses of the DDL file:

```
/*
**********************************************************************
********
 Program Name : MODEL_primer.sas
*/

/*
```

```
   Modify the environment and get an idea how everything is set
*/
options fullstimer mprint noovp source2 compress=yes  SYMBOLGEN ;
*mlogic ;
proc options;
run;

/*
   Get some environment variables, manipulate them, and include macros
*/
%let sasfile=/home/david.horvath/Polymorphic SAS;
%let sasfilc=MODEL_profile_202208;
%let sassrc=/home/david.horvath/Polymorphic SAS;
%let sascur=profile_202209;
%let sasspr=MODEL_profile_202208;
%let sascmn=/home/david.horvath/Polymorphic SAS;
%let sasmod=/home/david.horvath/Polymorphic SAS;
%let YYYYMM=202208;

libname  u "&sasfile";

%inc "&sascmn/primer_macro.sas";


DATA  u.&sasfilc;
/* create account or cust ID also */
/*length account $ 16;
retain account ".";    */
retain primary_key .;

%inc "&sasmod/MODEL_ddl.sas";

Delete;

run;
```

Note that many of the standard inputs to this module are defined as macro variables. In my simple examples, I hard coded them; in the real process they were %getenv() calls for the appropriate value.

Two of the key lines are:

```
%inc "&sascmn/primer_macro.sas";
%inc "&sasmod/MODEL_ddl.sas";
```

The first builds the macros (ddl_builder_single and ddl_builder) as needed for the primer process and the second invokes the list of those macros that define the layout for this particular model.

The initialization/definition of the primary_key is hard-coded because that is unique to each model.

In this context, the DDL macros produce retain and length statements as appropriate:

```
 142         %inc "&sasmod/MODEL_ddl.sas";
/home/david.horvath/Polymorphic SAS
 NOTE: %INCLUDE (level 1) file
/home/david.horvath/Polymorphic SAS/MODEL_ddl.sas is file
/home/david.horvath/Polymorphic
       SAS/MODEL_ddl.sas.
 143        +/* MODEL_ddl.sas edited
 144
+==================================================*/
 145        +%ddl_builder_single (numeric1_m1,N,0)
MPRINT(DDL_BUILDER_SINGLE):   retain numeric1_m1 0 ;
 146        +%ddl_builder_single (character1_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   length character1_m1 $ 8 ;
 MPRINT(DDL_BUILDER_SINGLE):  ;
MPRINT(DDL_BUILDER_SINGLE):   retain character1_m1 ".";
 147        +%ddl_builder_single (character2_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   length character2_m1 $ 8 ;
 MPRINT(DDL_BUILDER_SINGLE):  ;
MPRINT(DDL_BUILDER_SINGLE):   retain character2_m1 ".";
MPRINT(DDL_BUILDER_SINGLE):   length character3_m1 $ 8 ;
 MPRINT(DDL_BUILDER_SINGLE):  ;
MPRINT(DDL_BUILDER_SINGLE):   retain character3_m1 ".";
 149        +%ddl_builder(numericmon1_m,13,N,0)
MPRINT(DDL_BUILDER):   retain numericmon1_m1 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m2 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m3 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m4 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m5 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m6 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m7 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m8 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m9 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m10 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m11 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m12 0 ;
MPRINT(DDL_BUILDER):   retain numericmon1_m13 0 ;
 150        +%ddl_builder(numericmon2_m,13,N,0)
MPRINT(DDL_BUILDER):   retain numericmon2_m1 0 ;
MPRINT(DDL_BUILDER):   retain numericmon2_m2 0 ;
```

```
…
MPRINT(DDL_BUILDER):   retain numericmon2_m13 0 ;
 151       +%ddl_builder(numericmon3_m,3,N,0)
…
 152       +%ddl_builder(charactermon1,13,C,10)
MPRINT(DDL_BUILDER):   length charactermon11 $ 10 ;
MPRINT(DDL_BUILDER):   retain charactermon11 ".";
MPRINT(DDL_BUILDER):   length charactermon12 $ 10 ;
MPRINT(DDL_BUILDER):   retain charactermon12 ".";
…
 MPRINT(DDL_BUILDER):   length charactermon113 $ 10 ;
 MPRINT(DDL_BUILDER):  ;
 MPRINT(DDL_BUILDER):   retain charactermon113 ".";
 153       +%ddl_builder(charactermon2,3,C,5)
…
 154
 155        Delete;
 156
```

THE MODEL MERGER/SCORE EXECUTION

The model shifter is one of the more complex uses of the DDL file:

```
/*
**********************************************************************
********
 Program Name : MODEL_merger.sas
*/

/*
   Modify the environment and get an idea how everything is set
*/
options fullstimer mprint noovp source2 compress=yes  SYMBOLGEN ;
*mlogic ;
proc options;
run;

/*
   Get some environment variables, manipulate them, and include macros
*/
%let sasfile=/home/david.horvath/Polymorphic SAS;
%let sasfilc=MODEL_profile_202209;
%let sassrc=/home/david.horvath/Polymorphic SAS;
%let sascur=profile_202209;
%let sasspr=MODEL_profile_202208;
%let sascmn=/home/david.horvath/Polymorphic SAS;
```

```
%let sasmod=/home/david.horvath/Polymorphic SAS;
%let YYYYMM=202209;

libname  u "&sasfile";

%inc "&sascmn/merge_keep_macro.sas";

%inc "&sasmod/MODEL_ddl.sas"; ;

data u.&sasfilc  (keep = primary_key
&ddl_builder_list
)
;
   merge u.&sascur   (in = a )
         u.&sasspr;

   by primary_key;
      if a=1 then do;
      /* model can be executed here as well */
     output u.&sasfilc;
    end;
run;
```

As with the Primer, many of the standard inputs to this module are defined as macro variables.

The key lines within this script are as follows:

```
%inc "&sascmn/merge_keep_macro.sas";
%inc "&sasmod/MODEL_ddl.sas"; ;
```

The first builds the macros (ddl_builder_single and ddl_builder) as needed for the shifter process and the second invokes the list of those macros that define the layout for this particular model.

In this context, the DDL macros produce a list of variables for use within a keep statement as appropriate. This macro variable is used within:

```
data u.&sasfilc  (keep = primary_key
&ddl_builder_list
)
```

We can look at the creation of the ddl_builder_list (truncated):

```
119         %inc "&sasmod/MODEL_ddl.sas"; ;
 NOTE: %INCLUDE (level 1) file
/home/david.horvath/Polymorphic SAS/MODEL_ddl.sas is file
/home/david.horvath/Polymorphic
       SAS/MODEL_ddl.sas.
 120        +/* MODEL_ddl.sas edited
 121
+===================================================*/
 122        +%ddl_builder_single (numeric1_m1,N,0)
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
 SYMBOLGEN:  Macro variable BASE resolves to numeric1_m1
 123        +%ddl_builder_single (character1_m1,C,8)
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
numeric1_m1
 SYMBOLGEN:  Macro variable BASE resolves to
character1_m1
 124        +%ddl_builder_single (character2_m1,C,8)
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
numeric1_m1 character1_m1
 SYMBOLGEN:  Macro variable BASE resolves to
character2_m1
 125        +%ddl_builder_single (character3_m1,C,8)
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
numeric1_m1 character1_m1 character2_m1
 SYMBOLGEN:  Macro variable BASE resolves to
character3_m1
 126        +%ddl_builder(numericmon1_m,13,N,0)
 SYMBOLGEN:  Macro variable MAX resolves to 13
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
numeric1_m1 character1_m1 character2_m1 character3_m1
 SYMBOLGEN:  Macro variable BASE resolves to
numericmon1_m
 SYMBOLGEN:  Macro variable I resolves to 1
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
numeric1_m1 character1_m1 character2_m1 character3_m1
numericmon1_m1
```

The resulting code (log) looks like:

```
132         data u.&sasfilc  (keep = primary_key
 133         &ddl_builder_list
 SYMBOLGEN:  Macro variable DDL_BUILDER_LIST resolves to
numeric1_m1 character1_m1 character2_m1 character3_m1
numericmon1_m1
            numericmon1_m2 numericmon1_m3 numericmon1_m4
numericmon1_m5 numericmon1_m6 numericmon1_m7
numericmon1_m8 numericmon1_m9
```

```
             numericmon1_m10 numericmon1_m11
numericmon1_m12 numericmon1_m13 numericmon2_m1
numericmon2_m2 numericmon2_m3
             numericmon2_m4 numericmon2_m5 numericmon2_m6
numericmon2_m7 numericmon2_m8 numericmon2_m9
numericmon2_m10
             numericmon2_m11 numericmon2_m12
numericmon2_m13 numericmon3_m1 numericmon3_m2
numericmon3_m3 charactermon11
             charactermon12 charactermon13 charactermon14
charactermon15 charactermon16 charactermon17
charactermon18 charactermon19
             charactermon110 charactermon111
charactermon112 charactermon113 charactermon21
charactermon22 charactermon23
 134        )
 135        ;
 SYMBOLGEN:  Macro variable SASCUR resolves to
profile_202209
 136          merge u.&sascur   (in = a )
 137                u.&sasspr;
```

Note that I put the keep statements in the data statement (rather than in the merge u.&sasspr) because I wanted to drop all other variables – including any that might be created within the score itself.

The differences between the Merge and the Score execution versions are:

1. Addition of score output dataset to the data statement.
2. Addition of the model code (often via an include) into the 'if a=1' block.
3. Addition of an output statement in that same block.

The key was to minimize the number of times we would process each file since they were rather large.

THE MODEL SHIFTER

The model shifter is one of the more complex uses of the DDL file:

```
/*
**********************************************************************
********
 Program Name : MODEL_shifter.sas
*/

/*
   Modify the environment and get an idea how everything is set
```

```
*/
options fullstimer mprint noovp source2 compress=yes  SYMBOLGEN ;
*mlogic ;
proc options;
run;

/*
   Get some environment variables, manipulate them, and include macros
*/
%let sasfile=/home/david.horvath/Polymorphic SAS;
%let sasfilc=MODEL_profile_202208;
%let sassrc=/home/david.horvath/Polymorphic SAS;
%let sascur=profile_202209;
%let sasspr=MODEL_profile_202208;
%let sascmn=/home/david.horvath/Polymorphic SAS;
%let sasmod=/home/david.horvath/Polymorphic SAS;
%let YYYYMM=202208;

libname  u "&sasfile";

%inc "&sascmn/shifter_macro.sas";

proc contents DATA = u.&sasspr;
   title "MODEL_profile before shift";
run;

proc datasets library  = u;
   modify &sasspr;
   rename
%inc "&sasmod/MODEL_ddl.sas";
;
run;

proc contents data=u.&sasspr ;
   title "MODEL_profile after shift";
run;
```

As with the other examples, many of the standard inputs to this module are defined as macro variables.

Two of the key lines are:

```
%inc "&sascmn/shifter_macro.sas";
%inc "&sasmod/MODEL_ddl.sas";
```

The first builds the macros (ddl_builder_single and ddl_builder) as needed for the shifter process and the second invokes the list of those macros that define the layout for this particular model.

In this context, the DDL macros produce rename statements for the proc datasets as appropriate:

```
124         proc datasets library  = u;
 SYMBOLGEN:  Macro variable SASSPR resolves to
MODEL_profile_202208
 125            modify &sasspr;
 SYMBOLGEN:  Macro variable SASMOD resolves to
/home/david.horvath/Polymorphic SAS
 126            rename
 127         %inc "&sasmod/MODEL_ddl.sas";
 NOTE: %INCLUDE (level 1) file
/home/david.horvath/Polymorphic SAS/MODEL_ddl.sas is file
/home/david.horvath/Polymorphic
       SAS/MODEL_ddl.sas.
 128        +/* MODEL_ddl.sas edited
 129
+===============================================*/
 130        +%ddl_builder_single (numeric1_m1,N,0)
MPRINT(DDL_BUILDER_SINGLE):   numeric1_m1 = numeric1_mX
 131        +%ddl_builder_single (character1_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   character1_m1 =
character1_mX
 132        +%ddl_builder_single (character2_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   character2_m1 =
character2_mX
 133        +%ddl_builder_single (character3_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   character3_m1 =
character3_mX
 134        +%ddl_builder(numericmon1_m,13,N,0)
MPRINT(DDL_BUILDER):   numericmon1_m13 = numericmon1_m14
numericmon1_m12 = numericmon1_m13 numericmon1_m11 =
numericmon1_m12
 numericmon1_m10 = numericmon1_m11 numericmon1_m9 =
numericmon1_m10 numericmon1_m8 = numericmon1_m9
numericmon1_m7 = numericmon1_m8
 numericmon1_m6 = numericmon1_m7 numericmon1_m5 =
numericmon1_m6 numericmon1_m4 = numericmon1_m5
numericmon1_m3 = numericmon1_m4
 numericmon1_m2 = numericmon1_m3 numericmon1_m1 =
numericmon1_m2
 135        +%ddl_builder(numericmon2_m,13,N,0)
```

```
MPRINT(DDL_BUILDER):   numericmon2_m13 = numericmon2_m14
numericmon2_m12 = numericmon2_m13 numericmon2_m11 =
numericmon2_m12
 numericmon2_m10 = numericmon2_m11 numericmon2_m9 =
numericmon2_m10 numericmon2_m8 = numericmon2_m9
numericmon2_m7 = numericmon2_m8
 numericmon2_m6 = numericmon2_m7 numericmon2_m5 =
numericmon2_m6 numericmon2_m4 = numericmon2_m5
numericmon2_m3 = numericmon2_m4
 numericmon2_m2 = numericmon2_m3 numericmon2_m1 =
numericmon2_m2
 136        +%ddl_builder(numericmon3_m,3,N,0)
MPRINT(DDL_BUILDER):   numericmon3_m3 = numericmon3_m4
numericmon3_m2 = numericmon3_m3 numericmon3_m1 =
numericmon3_m2
 137        +%ddl_builder(charactermon1,13,C,10)
MPRINT(DDL_BUILDER):   charactermon113 = charactermon114
charactermon112 = charactermon113 charactermon111 =
charactermon112
 charactermon110 = charactermon111 charactermon19 =
charactermon110 charactermon18 = charactermon19
charactermon17 = charactermon18
 charactermon16 = charactermon17 charactermon15 =
charactermon16 charactermon14 = charactermon15
charactermon13 = charactermon14
 charactermon12 = charactermon13 charactermon11 =
charactermon12
 138        +%ddl_builder(charactermon2,3,C,5)
MPRINT(DDL_BUILDER):   charactermon23 = charactermon24
charactermon22 = charactermon23 charactermon21 =
charactermon22
 NOTE: %INCLUDE (level 1) ending.
 139         ;
 NOTE: Renaming variable numeric1_m1 to numeric1_mX.
 NOTE: Renaming variable character1_m1 to character1_mX.
 NOTE: Renaming variable character2_m1 to character2_mX.
 NOTE: Renaming variable character3_m1 to character3_mX.
 NOTE: Renaming variable numericmon1_m13 to
numericmon1_m14.
NOTE: Renaming variable numericmon1_m12 to
numericmon1_m13.
…
NOTE: Renaming variable charactermon21 to charactermon22.
```

```
 140         run;

 NOTE: MODIFY was successful for
U.MODEL_PROFILE_202208.DATA.
 141

 NOTE: PROCEDURE DATASETS used (Total process time):
       real time           0.03 seconds
       user cpu time       0.04 seconds
       system cpu time     0.00 seconds
       memory              1111.00k
```

Note that the structure of the renames are important: Mn to Mn+1, then Mn-1 to Mn, etc. Any single month files are renamed to MX because the keep macro will not create those.

Also note the amount of time required.

THE MODEL REVERSE SHIFTER

Last but certainly not least is the Model profile reverse shifter:

```
/*
**********************************************************************
********
 Program Name : MODEL_shifter_reverse.sas
*/

/*
   Modify the environment and get an idea how everything is set
*/
options fullstimer mprint noovp source2 compress=yes  SYMBOLGEN ;
*mlogic ;
proc options;
run;

/*
   Get some environment variables, manipulate them, and include macros
*/
%let sasfile=/home/david.horvath/Polymorphic SAS;
%let sasfilc=MODEL_profile_202208;
%let sassrc=/home/david.horvath/Polymorphic SAS;
%let sascur=profile_202209;
%let sasspr=MODEL_profile_202208;
%let sascmn=/home/david.horvath/Polymorphic SAS;
%let sasmod=/home/david.horvath/Polymorphic SAS;
%let YYYYMM=202208;

libname  u "&sasfile";
```

```
proc contents DATA = u.&sasspr;
   title "MODEL_profile before shift";
run;

%inc "&sascmn/shifter_macro_reverse.sas";


proc datasets library=u;
   modify &sasspr;
   rename
%inc "&sasmod/MODEL_ddl.sas";
;
run;


proc contents data=u.&sasspr ;
   title "MODEL_profile after shift";
run;
```

The important lines for this process are:

```
%inc "&sascmn/shifter_macro_reverse.sas";
%inc "&sasmod/MODEL_ddl.sas";
```

The first builds the macros (ddl_builder_single and ddl_builder) as needed for the reverse shifter process and the second invokes the list of those macros that define the layout for this particular model.

In this context, the DDL macros produce rename statements for the proc datasets as appropriate:

```
125         proc datasets library=u;
 SYMBOLGEN:  Macro variable SASSPR resolves to
MODEL_profile_202208
 126            modify &sasspr;
 SYMBOLGEN:  Macro variable SASMOD resolves to
/home/david.horvath/Polymorphic SAS
 127            rename
 128         %inc "&sasmod/MODEL_ddl.sas";
 NOTE: %INCLUDE (level 1) file
/home/david.horvath/Polymorphic SAS/MODEL_ddl.sas is file
/home/david.horvath/Polymorphic
       SAS/MODEL_ddl.sas.
 129        +/* MODEL_ddl.sas edited
 130
+================================================*/
```

```
 131        +%ddl_builder_single (numeric1_m1,N,0)
MPRINT(DDL_BUILDER_SINGLE):   numeric1_mX = numeric1_m1
 132        +%ddl_builder_single (character1_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   character1_mX =
character1_m1
 133        +%ddl_builder_single (character2_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   character2_mX =
character2_m1
 134        +%ddl_builder_single (character3_m1,C,8)
MPRINT(DDL_BUILDER_SINGLE):   character3_mX =
character3_m1
 135        +%ddl_builder(numericmon1_m,13,N,0)
MPRINT(DDL_BUILDER):   numericmon1_m2 = numericmon1_m1
numericmon1_m3 = numericmon1_m2 numericmon1_m4 =
numericmon1_m3
 numericmon1_m5 = numericmon1_m4 numericmon1_m6 =
numericmon1_m5 numericmon1_m7 = numericmon1_m6
numericmon1_m8 = numericmon1_m7
 numericmon1_m9 = numericmon1_m8 numericmon1_m10 =
numericmon1_m9 numericmon1_m11 = numericmon1_m10
numericmon1_m12 =
 numericmon1_m11 numericmon1_m13 = numericmon1_m12
numericmon1_m14 = numericmon1_m13
 136        +%ddl_builder(numericmon2_m,13,N,0)
MPRINT(DDL_BUILDER):   numericmon2_m2 = numericmon2_m1
numericmon2_m3 = numericmon2_m2 numericmon2_m4 =
numericmon2_m3
 numericmon2_m5 = numericmon2_m4 numericmon2_m6 =
numericmon2_m5 numericmon2_m7 = numericmon2_m6
numericmon2_m8 = numericmon2_m7
 numericmon2_m9 = numericmon2_m8 numericmon2_m10 =
numericmon2_m9 numericmon2_m11 = numericmon2_m10
numericmon2_m12 =
 numericmon2_m11 numericmon2_m13 = numericmon2_m12
numericmon2_m14 = numericmon2_m13
 137        +%ddl_builder(numericmon3_m,3,N,0)
…
NOTE: %INCLUDE (level 1) ending.
 140          ;
 NOTE: Renaming variable numeric1_mX to numeric1_m1.
 NOTE: Renaming variable character1_mX to character1_m1.
 NOTE: Renaming variable character2_mX to character2_m1.
 NOTE: Renaming variable character3_mX to character3_m1.
 NOTE: Renaming variable numericmon1_m2 to
numericmon1_m1.
 NOTE: Renaming variable numericmon1_m3 to
numericmon1_m2.
 NOTE: Renaming variable numericmon1_m4 to
numericmon1_m3.
 NOTE: Renaming variable numericmon1_m5 to
numericmon1_m4.
 NOTE: Renaming variable numericmon1_m6 to
numericmon1_m5.
…
 NOTE: Renaming variable charactermon24 to
charactermon23.
 141         run;

 NOTE: MODIFY was successful for
U.MODEL_PROFILE_202208.DATA.
 142
 143

 NOTE: PROCEDURE DATASETS used (Total process time):
       real time           0.03 seconds
       user cpu time       0.03 seconds
       system cpu time     0.01 seconds
```

Note that, like with Model shifter, the structure of the renames are important: M2 to M1, M3 to M2, Mn+1 to Mn. Any single month files are renamed from MX back to _m1.

Also note the amount of time required.

THE ACTUAL MACRO CODE

As you've seen, the same macros (&ddl_builder and &ddl_builder_single) behaved differently depending on their context – a simple example of polymorphism. And I've been told that you can't do polymorphism in SAS. I love proving experts wrong!

The naming of the macros being defined in each of the following four files was so important that it was included in all the comments. The only way for this polymorphism to work was for them to remain with the same name.

primer_macro.sas

The primer builds retain and length statements as appropriate for each of the variables (and each of the desired months):

```
/*
   primer_macro.sas
   The following macros create the statements necessary to create
variables
   for use in creating an empty ("primed") score profile SAS dataset.
   Do not change the names of these macros. Versions with the same
name
   but designed for the shifting code (moving months) and merge also
exist.
   This allows one set of code (the include module that invokes these
   macros) to be used for two different but related purposes.
*/
%macro ddl_builder(base, max, type, length);
   %let start = %eval(&max + 1);
   %do i = 1 %to &max %by 1;
      %if &type=C %then %do;
         length &base&i $ &length %str(;);
         retain &base&i ".";
      %end;
      %else %if &type=N %then
         retain &base&i 0 %str(;);
      %else %if &type=DATE %then %do;
         retain &base&i 0 %str(;);
         format &base&i &type&length%str(;);
      %end;
   %end;
%mend ddl_builder;

%macro ddl_builder_single(base, type, length);
   %if &type=C %then %do;
      length &base $ &length %str(;);
      retain &base ".";
   %end;
   %else %if &type=N %then
      retain &base 0 %str(;);
   %else %if &type=DATE %then %do;
      retain &base 0 %str(;);
      format &base &type&length %str(;);
   %end;
%mend ddl_builder_single;
```

I'm not going to spend much space describing these – compare the inputs and outputs is the best example. One note is the type=DATE. While we all know that dates are stored as numbers, we wanted them to print out in a decent format.

merge_keep_macro.sas

The merge keep macro builds the list of variables that would be (as the name implies) kept during the merge. This drops off any of the old fields (MX and Mn+1) from the prior month model profile:

```
/*
   merge_keep_macro.sas
   The following macros create the statements necessary to KEEP the
   proper columns for the next month score profile (dropping the extra
   month of history introduced by the shifter code).

   Do not change the names of these macros. Versions with the same
name
   but designed for the shifting code (moving months) also exist.
   This allows one set of code (the include module that invokes these
   macros) to be used for two different but related purposes.
*/
%global ddl_builder_list;
%let ddl_builder_list=;
%macro ddl_builder(base, max, type, length);
   %do i = 1 %to &max %by 1;
      %let ddl_builder_list=&ddl_builder_list &base&i ;
   %end;
%mend ddl_builder;

%macro ddl_builder_single(base, type, length);
   %let ddl_builder_list=&ddl_builder_list &base ;
%mend ddl_builder_single;
```

Very simply, the macros build a list of variables based on the definition (which does not include the out-of-range variables).

shifter_macro.sas

The shifter macro builds statements for the proc datasets rename (much like the data statement rename) as described previously:

```
/*
   shifter_macro.sas
   The following macros create the statements necessary to rename
columns for month-shifting

   Do not change the names of these macros. Versions with the same
name
   but designed for the merging code (creating drop statements) also
exist.
   This allows one set of code (the include module that invokes these
   macros) to be used for two different but related purposes.
*/
%macro ddl_builder(base, max, type, length);
   %let start = %eval(&max + 1);
   %do i = &start %to 2 %by -1;
      %let from = %eval(&i - 1);
      &base&from = &base&i
   %end;
%mend ddl_builder;

%macro ddl_builder_single(base, type, length);
   %let i = X;
   %let max_l = %eval(%length(&base) - 1);
   &base = %substr(&base, 1, &max_l)&i
/*   &base = &base&i  */
%mend ddl_builder_single;
```

Note the loop is high to low.

shifter_macro_reverse.sas

The shifter macro reverse builds statements for the proc datasets rename (much like the data statement rename) as described previously:

```
/*
   shifter_macro_reverse.sas
   The following macros create the statements necessary to reverse
rename columns.

   Do not change the names of these macros. Versions with the same
name
   but designed for the merging code (creating drop statements) also
exist.
   This allows one set of code (the include module that invokes these
   macros) to be used for two different but related purposes.
*/
%macro ddl_builder(base, max, type, length);
   %let end   = &max ;
   %do i = 1 %to &end;
      %let to= %eval(&i + 1);
      &base&to   = &base&i
   %end;
%mend ddl_builder;

%macro ddl_builder_single(base, type, length);
   %let i = X;
   %let max_l = %eval(%length(&base) - 1);
   %substr(&base, 1, &max_l)&i = &base
/*   &base = &base&i  */
%mend ddl_builder_single;
```

Note that the loop is from low to high.

CONCLUSION

We had to balance some conflicting requirements including storage usage, execution time, ease of maintenance, development time, and matching corporate standards. By using the common DDL coding and use to generate the necessary code helped to standardize (and speed) the development without significant penalty in other areas. Plus it was a lot of fun to try some significant new techniques.

ACKNOWLEDGMENTS

I would like to thank the organizers of this conference. This is neither my first SESUG nor first time speaking at SESUG – I've always enjoyed working with the organizers.

I also want to thank my employer for their support in attending (and speaking at) this conference even though I'm not allowed to mention their name. Last, and certainly not least, is my spouse Mary who doesn't complain when I spend time working on conference materials.

CONTACT INFORMATION

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

David B. Horvath, MS, CCP
+1-610-859-8826
Email: dhorvath@cobs.com
Web: http://www.cobs.com
LI: http://www.linkedin.com/in/dbhorvath