Paper 10662-2016

Avoid Change Control by Using Control Tables

Frank Ferriola, Financial Risk Group

ABSTRACT

Developers working on a production process need to think carefully about ways to avoid future changes that require change control, so it's always important to make the code dynamic rather than hardcoding items into the code. Even if you are a seasoned programmer, the hardcoded items might not always be apparent. This paper assists in identifying the harder-to-reach hardcoded items and addresses ways to effectively use control tables within the SAS® software tools to deal with sticky areas of coding such as formats, parameters, grouping/hierarchies, and standardization. The paper presents examples of several ways to use the control tables and demonstrates why this usage prevents the need for coding changes. Practical applications are used to illustrate these examples.

INTRODUCTION

When I first started out using Base SAS in 1984, it was considered an “end-user” oriented language and was unstructured in how you used it. It was also scoffed at by IT (then called Systems Programming and Development where I worked) which was primarily COBOL programmers who had to follow structured process rules in order to implement a Production Process.

It was because of this structure that I got started in SAS in the first place. An implementation was going into Production by a certain date and 3 months out, my boss realized that the Financial Reporting Module was not going to be on time. Since I was an Accountant at the time, and Financial Reports were important to our department, he told me to “go learn SAS.” I did and with the help of a couple of the COBOL programmers, I was able to pull data and create reports that we could use to book the information from the system.

Over the ensuing years, SAS has become more robust and structured in many areas and is now implemented as production processes in many companies. This usually means the code goes through a rigorous testing process before implementation and any changes to that code, no matter how simple needs to go through a change control process when things either break or change.

This presents challenges to the process because simple changes that would take seconds to insert into the code and run, now takes more than a few minutes to resolve and in my experience could take months to implement.

In previous papers, I have stressed the importance of design in a production process, and it’s important to consider how things may change even if they never do, and to accomplish this I have found the use of control tables as a solution in many cases.

On the other hand, change control is necessary, however the methods employed here are intended to mitigate some of the changes that can be remedied easily, leaving time available to tackle the more complex changes.

WHAT ARE CONTROL TABLES AND WHY SHOULD WE USE THEM?

A Control Table as used in this paper is any table that helps to control a process and make it dynamic. The table must be updateable either from within the process or outside the process depending on how your production environment is set up. It can have multiple uses across several processes or just be used in one process. It can also be as small and simple as one cell or complex as a large matrix.

The control table primarily is used to avoid hardcoding inside a program. We can use them to make the process more flexible. Some examples include setting parameters, lookups of data, format, map or group the data and providing directions to the code.

CONTROL TABLES WITH PARAMETERS

Using data from a control table to set parameters for a process is a very typical way to use a control table. Many times the table has two columns with one column containing a qualifier of some kind and the second column being a value for that qualifier. Additional qualifying columns could be added.

Here is one example:

Parameter	Value
Start_DT	01/01/2015
End_DT	12/31/2015
Product	Mortgage
Cycles	12

The control Table above contains four separate parameters that can be read into a process and set macro variables based on the parameters.

```
Data _NULL_;
   Set cntl.PARM_TABLE;
   call symputx (parameter, value, 'g');
run;
```

The macrovariables created can then be used within the processing in the application. The parameters can be changed by submitting code to change the cntl.PARM_TABLE before running it. The process can also be modified to add other parameters that can be used to run certain parts of the process and bypass others. This can be done utilizing macros that can direct the process.

The control table can be updated through SAS Enterprise Guide or a script.

CONTROL TABLES FOR FORMATS

A common need to change code happens when you create formats based on information at the time the code is developed and there is a change or addition to the format. By using a control table as a way to update the format will allow you to avoid making changes directly to the code every time the data changes.

For example, let's say you have a format created for Products in your code:

```
proc format;
   value product 1 = 'Soy'
                 2 = 'Wheat'
                 3 = 'Corn'
                 4 = 'Oats'
         ;
run;
```

This month a new product is coming through the data—Canola. It would be simple to add a line to the format code to add it as product 5, but that would involve a code change and would require it to go through change control.

A control table named cntl.product would remedy this.

Product_CD	Product_Desc
1	Soy

2	Wheat
3	Corn
4	Oats
5	Canola

There are several ways to use this table, but for presentation purposes, I will show one the cntlin option in proc format.

First create a work table from the control table for formatting purposes:

```
data product_fmt;
   set cntl.product(rename=(product_cd=start product_desc=label));
   fmtname='product';
run;
```

Then use cntlin to recreate the format:

```
proc format cntlin = product_fmt;
run;
```

Opening the format will display the following:

FORMAT NAME: PRODUCT LENGTH: 6 NUMBER OF VALUES: 5
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 6 FUZZ: STD

START	END	LABEL (VER. V7|V8 06MAR2016:10:29:08)
1	1	Soy
2	2	Wheat
3	3	Corn
4	4	Oats
5	5	Canola

CONTROL TABLES FOR LOOKUPS

A control table can also be used to lookup data. The same table used above to format could be used as a lookup to data. The lookup table primarily adds additional information to the data.

The method we will use here is a join, but many other methods can be employed for this step.

Let's take the following table work.purchases:

CUSTOMER_ID	Product_Cd	Purchase_Date	Units
103	1	10/15/2015	15
103	2	10/18/2015	40
103	3	10/10/2015	25
103	4	10/15/2015	35
104	1	10/15/2015	30
104	5	10/15/2015	45

We can join with cntl.product to find the names of the products.

```
proc sql;
   create table product_map as
   select purch.*,
          prdct.product_desc
   from purchases as purch
   left join cntl.product as prdct
   on purch.product_cd=prdct.product_Cd
   order by purch.customer_id,
            purch.product_cd
   ;
quit;
```

This will result in the following table:

CUSTOMER_ID	Product_Cd	Purchase_Date	Units	Product_Desc
103	1	10/15/2015	15	Soy
103	2	10/18/2015	40	Wheat
103	3	10/10/2015	25	Corn
103	4	10/16/2015	35	Oats
104	1	10/12/2015	30	Soy
104	5	10/20/2015	45	Canola

In this manner we can use Product_Desc as the format for Product_Cd.

Once again any changes or additions to the product code list would automatically be incorporated into the lookup. This may be a preferred method in your development (lookup instead of format) but either or both could be accommodated just by updating the control table rather than changing code.

CONTROL TABLES FOR MAPPING

The lookup table can do broader mapping than just using it as a substitute for formatting. We can do broader mapping from lookup tables. In order to do this you will need a common key to be present in the tables you are joining.

The products we have been using are commodities on commodity exchanges. We'll start with a table that we need to get mappings for:

Source.commodity_data

COMMODITY_ID	EXCHANGE_CD	UOM_ID
1	CBOT	1
2	CBOT	1
3	CME	1
4	CBOT	2
5	CBOT	3
6	NYMEX	3
7	CBOT	3
8	CME	5
9	CME	5

10	NYMEX	6
11	NYMEX	6
12	NYMEX	4

Each column will require additional information from a control table, so we have created three reference tables.

Cntl.Commodity_ref

COMMODITY_ID	COMMODITY_DESC
1	Soybean
2	Soymeal
3	Soyoil
4	Wheat
5	HRW Wheat
6	Corn
7	Oats
8	Rapeseed
9	Rapeoil
10	Canola/Rape
11	Canola Meal
12	Canola Oil

Cntl.uom_ref

UOM_ID	UOM_CD
1	BU
2	CWT
3	LB
4	MMBtu
5	MT
6	T

Cntl.exchange_ref

EXCHANGE_ID	EXCHANGE_CD	EXCHANGE_DESC
1	CBOT	Chicago Board of Trade
2	CME	Chicago Mercantile Exchange
3	NYMEX	New York Mercantile Exchange

Using code to join the four tables together and pull the mapped data in.

```
proc sql;
   create table cntl.commodity_mapped
   as select
      comm.*,
      cref.commodity_desc,
      uref.uom_cd,
      eref.exchange_desc
   from cntl.commodity_data as comm
   left join cntl.commodity_ref as cref
     on comm.commodity_id=cref.commodity_id
   left join cntl.uom_ref as uref
     on comm.uom_id=uref.uom_id
   left join cntl.exchange_ref as eref
     on comm.exchange_cd=eref.exchange_cd
        ;

quit;
```

This results in the following table being created:

COMMODITY_ID	UOM_ID	EXCHANGE_ID	COMMODITY_DESC	UOM_CD	EXCHANGE_DESC
7	3	CBOT	Oats	LB	Chicago Board of Trade
1	1	CBOT	Soybean	BU	Chicago Board of Trade
5	3	CBOT	HRW	LB	Chicago Board of Trade
2	1	CBOT	Soymeal	BU	Chicago Board of Trade
4	2	CBOT	Wheat	CWT	Chicago Board of Trade
3	1	CME	Soyoil	BU	Chicago Mercantile Exchange
8	5	CME	Rapeseed	MT	Chicago Mercantile Exchange
9	5	CME	Rapeoil	MT	Chicago Mercantile Exchange
6	3	NYMEX	Corn	LB	New York Mercantile Exchange
10	6	NYMEX	Canola	T	New York Mercantile Exchange
11	6	NYMEX	Canola	T	New York Mercantile Exchange
12	4	NYMEX	Canola	MMBtu	New York Mercantile Exchange

CONCLUSION

Change control is a necessary part of the Production Process. The effective use of Control Tables can help minimize the need for change control in many cases. As you design, develop and modify your process, look for ways to implement control tables. As you get more familiar with the use of control tables you will additionally find ways for the control tables to interact with each other to solve additional coding problems.

CONTACT INFORMATION

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

Frank Ferriola
Financial Risk Group
Email: Frank.Ferriola@frgrisk.com
Work Phone: 303-548-0278