PharmaSUG 2021 – SI-196

Chasing Master Data Interoperability: Facilitating Master Data Management (MDM) Through CSV Control Tables that Contain Data Rules that Support SAS® and Python Data-Driven Software Design

Troy Martin Hughes

ABSTRACT

Control tables are the tabular data structures that contain *control data*—the data that direct software execution and which can prescribe dynamic software functionality. Control tables offer a preferred alternative to hardcoded conditional logic statements, which require code customization to modify. Thus, control tables can dramatically improve software maintainability and configurability by empowering developers and, in some cases, nontechnical end users to alter software functionality without modifying code. Moreover, when control tables are maintained within canonical data structures such as comma-separated values (CSV) files, they facilitate master data interoperability by enabling one control table to drive not only SAS software but also non-SAS applications. This text introduces a reusable method that preloads CSV control tables into SAS temporary arrays to facilitate the evaluation of business rules and other data rules within SAS data sets. To demonstrate the interoperability of canonical data structures, including CSV control tables, a functionally equivalent Python program also ingests these control tables. Master data management (MDM) objectives are facilitated because only one instance of the *master data*—the control table, and single source of the truth—is maintained, yet it can drive limitless processes across varied applications and programming languages. Finally, when data rules must be modified, the control data within the control table can be changed only once to effect corresponding changes in all derivative uses of those master data.

ARRAY-BASED CONTROL TABLE INGESTION

Control tables provide a data-driven method to direct and modify software functionality by maintaining software instructions inside tables rather than in code. This alternative to hardcoded software design can be further improved by maintaining control tables in interoperable (aka, *canonical*) file formats such as Excel spreadsheets or CSV files that can be accessed by not only SAS but also other applications and programming languages.

For example, consider the requirement to scan SAS program files for specific SAS language elements such as procedure names or macro statements. In this scenario, your boss might want to understand whether SAS practitioners are utilizing the SORT procedure or the SQL procedure to order data. One solution would be to create a control table containing the search terms, to load that control table into memory or a data set, and to interrogate one or more SAS program files for the terms.

The CSV control table (e.g., d:\sas\terms.csv) contains an abbreviated list of sample terms:

```
PROC SORT,Procedure: Base
PROC FREQ,Procedure: Stats
PROC MEANS,Procedure: Stats
%MACRO,Macro Statement
%MEND,Macro Statement
%DO,Macro Statement
%LET,Macro Statement
```

The control table includes two comma-delimited columns—the term itself and its user-specified classification. Search terms can be added, removed, or modified by altering only the control table, so this data-driven design facilitates software stability and integrity—because the underlying code can remain static while its control data are modified over time. Moreover, this data-driven design supports software configurability because different users can utilize the table to search for different terms; one user might want to scour SAS programs for built-in procedures, another for built-in functions, and a third for SAS macro statements. Each of these users can utilize the identical SAS program to perform these disparate operations—by modifying *only* control data.

The sample SAS program file (d:\sas\etl.sas) will not be run but can be interrogated for keyword search terms:

```
data etl;
   set pres;
   length fullname $50 term 8 numVPs 8;
   format term 8.2;
   fullname=catx(' ',fname,lname);
   term=round((dt2-dt1)/365.25,.01);
   numVPs=countw(vp,',');
run;

proc sort data=etl;
   by num;
run;

proc means data=etl;
   var term numVPs;
run;

proc freq data=etl;
   tables term numVPs;
run;

PROC FREQ data=etl;
   tables lname;
run;

%macro sample();
%put Today is %sysfunc(putn(%sysfunc(date()),mmddyy10.));
%mend;

%sample;
```

The CTRL_TEXT_DOMAIN_TEXT macro first ingests the control table to determine the number of observations; this step is required because the observation count is used subsequently in the declaration of temporary SAS arrays. A second DATA step first ingests the control table (when _N_=1) and subsequently ingests the program file to search for the terms:

```
* saved as d:\sas\ctrl_text_domain_text.sas;
%macro ctrl_text_domain_text(ctrl= /* ctrl tab CSV */,
   domain= /* SAS program file name to parse */,
   case= /* SENSITIVE to do case-sensitive search */);
* facilitate case-insensitive FIND;
%if "%upcase(&case)"="SENSITIVE" %then %let case=;
%else %let case=,'i';
* get observation count;
%local nobs;
data _null_;
   infile "&ctrl" truncover end=eof;
   input line $500;
   if eof then call
      symputx('nobs',strip(put(_n_,8.)),'l');
run;
data metrics (drop=i term);
   if _n_=1 then do;
      i=0;
      infile "&ctrl" truncover end=eof dsd
         delimiter=',';
      do until(eof);
         i=i+1;
         length term $32 cat $32;
         input term $ cat $;
```

```
         array arrkey[&nobs] $32 _temporary_;
         arrkey[i]=strip(term);
         array arrcat[&nobs] $32 _temporary_;
         arrcat[i]=strip(cat);
         end;
      end;
   length line 8 contents $500 keyword $32;
   infile "&domain" truncover end=eof;
   input contents $500.;
   do i=1 to dim(arrkey);
      if find(contents,strip(arrkey[i])&case) then do;
         line=_n_;
         keyword=arrkey[i];
         cat=arrcat[i];
         output;
         end;
      end;
run;
%mend;
```

The user-specified file location (&LOC) should be modified, after which the macro can be executed:

```
%let loc=D:\sas\;     * USER MUST CHANGE LOCATION *;
%include "&loc.ctrl_text_domain_text.sas";
%ctrl_text_domain_text(ctrl=&loc.terms.csv,
   domain=&loc.etl.sas, case=);
```

Table 1 demonstrates the Metrics data set that is created from the default case-insensitive search.

Line	Contents	Keyword
10	proc sort data=etl;	PROC SORT
14	proc means data=etl;	PROC MEANS
18	proc freq data=etl;	PROC FREQ
22	PROC FREQ data=etl;	PROC FREQ
26	%macro sample();	%MACRO
28	%mend;	%MEND

Table 1. Metrics Data Set with Search Results

The use of SAS arrays is instrumental to the macro because it enables reserved words and special characters to be parsed without the need to mask them. For example, both %MACRO and %MEND are identified within the etl.sas program file without issue. Too often, the contents of control tables are exported from SAS data sets into macro variables or macro “lists.” Although this methodology may be preferred in some instances, character masking is often required and can unnecessarily complicate the solution.

Note that the CTRL_TEXT_DOMAIN_TEXT macro references only the structure—not the contents—of the underlying control table. Thus, the control table must be comma-delimited and contain two columns of data (because these columns are populated into the ARRKEY and ARRCAT arrays, respectively); however, the contents of these two columns are flexible and can be modified to meet the needs of specific users (thus facilitating configurability), or to adapt over time to changes in the SAS programming language itself (thus facilitating adaptability and future-proofing).

This data-driven methodology also facilitates scalability, in that as the list of keywords grows, the macro that reads these control data remains stable. Had a functionally equivalent hardcoded solution been utilized—for example, a gaggle of IF-THEN-ELSE statements to assess whether a keyword had been discovered—additional lines of code would need to be added each time a new keyword needed to be located. The increased software flexibility, adaptability, configurability, and scalability each contributes to

increased *software maintainability*—the ease with which software can be maintained (and modified if necessary) over time.

The CTRL_TEXT_DOMAIN_TEXT macro is demonstrated and more fully described in the author's text: *SAS Data-Driven Development: From Abstract Design to Dynamic Functionality*. (Hughes, 2019) The text additionally demonstrates other control table scenarios, including: maintaining a CSV control table and interrogating a SAS data set, and maintaining a SAS control table and interrogating either a text file or a SAS data set.

CONTROL TABLE INTEROPERABILITY

As mentioned, because the control table is maintained within a CSV file, it is more readily ingested by non-SAS applications. For example, another analyst might be given the same task (to find search terms within program files) but have more familiarity with the Python language. Rather than having to learn SAS or first export a SAS control table into an interoperable data structure, he could instead ingest the same CSV control file using the following Python code:

```
import csv
with open('d:/terms.csv') as ctrl:
    reader=csv.reader(ctrl)
    terms=list(reader)

metrics=[]

with open('d:/etl.sas') as f:
    datafile=f.readlines()
    lineno=0
    for line in datafile:
        lineno+=1
        for term in terms:
            if term[0] in line.upper():
                metrics.append([lineno,line,term[0]])
```

The Python code first imports the CSV control table into the Terms list, after which the program file is parsed for search terms. When a term is discovered, the term, the program line in which it was discovered, and the line number are appended to the Metrics list. The Metrics list could be further manipulated to visualize the results, but this functionality is not demonstrated.

Master data management objectives are facilitated because the single control table is now driving both the SAS program (CTRL_TEXT_DOMAIN_TEXT macro) and the Python program. If data within the control table must be modified (e.g., to add additional search terms), these changes can be made once and yet propagate to all programs relying on the control table. This efficiency overcomes a common limitation in SAS software design in which control tables are stodgily maintained as SAS data sets and are thus of limited or no use to non-SAS applications and languages that could (or should) otherwise rely on those master control data.

CONCLUSION

The use of control tables epitomizes data-driven software design by storing software instruction within malleable data structures rather than statically within code. This text introduced an array-based method to ingest and evaluate control tables, overcoming a common weakness in which SAS control data are unnecessarily converted to macro variables and macro "lists." Maintaining control tables within interoperable data structures (such as CSV files, as opposed to proprietary SAS data sets) further facilitates the ability of applications and programming languages to leverage these control tables. Furthermore, this strategy promotes MDM objectives by maintaining a single version of the truth—the master control data driving software and providing dynamic functionality.

REFERENCES

Hughes, T. M. (2019). *SAS® Data-Driven Development: From Abstract Design to Dynamic Functionality.* San Diego, California: CreateSpace.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com