SESUG Paper 167-2023

Make You Holla' Tikka Masala: Creating User-Defined Informats Using the PROC FORMAT OTHER Option To Call User-Defined FCMP Functions That Facilitate Data Ingestion Data Quality

Troy Martin Hughes

ABSTRACT

You can't just let any old thing inside your tikka masala—you need to carefully curate the ingredients of this savory, salty, sometimes spicy delicacy! Thus, when reviewing a data set that contains potential tikka masala ingredients, an initial data quality evaluation should differentiate approved from unapproved ingredients. Cumin, yes please; chicken, the more meat the merrier; coriander, of course; turmeric, naturally; yeast, are you out of your naan-loving mind?! Too often, SAS® practitioners first ingest a data set in one DATA step, and rely on subsequent DATA steps to clean, standardize, and format those data. This text demonstrates how user-defined informats can be designed to ingest, validate, clean, and standardize data in a single DATA step. Moreover, it demonstrates how the FORMAT procedure can leverage the OTHER option to create a user-defined informat that calls a user-defined FCMP function to perform complex data evaluation and transformation while simulating _ERROR_ variable functionality. Control what you put inside your tikka masala with this straightforward solution that epitomizes data-driven software design while leveraging the flexibility of the FORMAT and FCMP procedures!

INTRODUCTION

The delicacy that is tikka masala requires no introduction. Although spice is nice, my masala had better make me holla' as drenched in sweat, I gulp water while fighting off dopaminergic-induced tremors. Although chefs will vary capsaicin content and how they concoct their mouthwatering masalas, a core set of ingredients nevertheless comprises the dish—just as core categorical constituents commonly comprise valid values of some categorical variable. In this sense, the curation of tikka masala ingredients excellently models data validation, and can be used to demonstrate advanced methods that showcase data-driven software design.

SAS informats, including user-defined informats, offer a viable approach to validating categorical elements—like recipe ingredients—as they are ingested from raw data sources, such as text files. During more advanced data ingestion, user-defined informats can programmatically set the automatic _ERROR_ variable to 1 when invalid data are encountered. And within egregiously advanced data ingestion processes, the underlying logic of a user-defined informat can be maintained within a user-defined function that is indirectly called when the user-defined informat is applied. Mind-blowingly brilliant, this approach is demonstrated with respect to the culinary creation of tikka masala.

EMBRACING A DATA-DRIVEN APPROACH

A data-driven software design approach, in which business rules and data models are codified not within code but rather within control data, benefits data ingestion (and many other software operations) because control data are maintained separately from the software that interprets and operationalizes them. Thus, data-driven software design facilitates flexible software development in which different functionally equivalent solutions can be designed, yet all rely on the same underlying control data. Similarly, where user-defined data structures are standardized, the underlying code that interprets (and acts upon) them can support separate, unrelated software projects and products. That is, data-driven design, when done right, facilitates the reuse of both user-defined data structures and user-defined operations. We call that a two-fer!

Embracing this design pattern, the following Ingredients data set represents master data—the enumeration of tikka-related ingredients—that are subsequently relied upon as a baseline for validation operations:

```
data ingredients;
   infile datalines dsd;
   length ingredient $50;
```

```
   input ingredient $;
   datalines;
cumin
chili powder
coriander
turmeric
garlic
salt
chicken
;
```

In other words, Ingredients enumerates what you want inside your tikka masala, and through evaluation, operates as a lookup table that can be used to validate potential tikka masala foodstuffs. Both informat- and function-related methods are demonstrated that leverage this lookup table.

For example, the following text file (d:\sas\troys_tasty_tikka.txt) simulates one purported tikka masala recipe, which—spoiler ahead—may suffer from some serious data quality issues:

```
turmeric, 1, speck
garlic, 5, cloves
chicken, 8, breasts
cumin, 1, bounty
coriander, 20, seeds
salt, 2, pinches
sweet cream, 2, pumps
yeast, 3, packages
flour, 8, handfuls
```

The following DATA step reads the text file to create the Troys_tasty_tikka data set:

```
%let loc=d:\sas\;                  * USER MUST CHANGE LOCATION *;
filename f "&loc.troys_tasty_tikka.txt";

data troys_tasty_tikka;
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20;
   input item $ quantity measurement $;
run;
```

In subsequent sections, various methods will evaluate Troys_tasty_tikka with respect to the Ingredients lookup table to evaluate the appropriateness of specific ingredients.

CLEANING YOUR TIKKA WITH A USER-DEFINED INFORMAT

The CNTLIN option of the FORMAT procedure facilitates a data-driven approach to creating both user-defined formats and user-defined informats, and these formats and informats can subsequently be used to validate, clean, standardize, or otherwise transform data. Within the FORMAT procedure, the CNTLIN option (i.e., “control table in”) names a specially formatted data set (that must include the Fmtname, Start, Label, and Type variables, and optionally the HLO variable) that operates as a lookup table.

For example, the following DATA step transforms the Ingredients data set into the INF_tikka_data data set, which is leveraged subsequently by the FORMAT procedure:

```
data inf_tikka_data;
   set ingredients(rename=(ingredient=start)) end=eof;
   length label $32 fmtname $32 type $2 hlo $2;
   label=start;
   fmtname='inf_tikka';
   type='j';      * J denotes a character informat;
   hlo='';
   output;
   if eof then do;
      label='';
      start='';
      hlo='o';    * O denotes other (i.e., invalid);
      output;
```

```
      end;
run;

proc format cntlin=inf_tikka_data;
run;
```

The Fmtname variable names the format or informat; the Start variable represents the value being evaluated or transformed; the Label variable represents the value after transformation; the Type variable describes whether a character or numeric format or informat is being defined; and the HLO variable can optionally describe how to handle values that are observed that do not exist in the lookup table. In this example, when a value (like sweet cream or yeast) is observed, yet not found in the data model (i.e., the Ingredients data set), the user-defined informat will return a missing value. The subsequent FORMAT procedure reads the data set specified by the CNTLIN option, and creates the informat (INF_TIKKA) specified by the Fmtname variable.

The INF_TIKKA informat is applied to the Item variable in the following DATA step to initialize the Valid_item variable, which includes only valid ingredients, and which denotes invalid ingredients as missing data:

```
data valid_recipe;
   set troys_tasty_tikka;
   length valid_item $50;
   valid_item=input(item, $inf_tikka.);
run;
```

The log demonstrates application of the INF_TIKKA informat:

```
139  data valid_recipe;
140     set troys_tasty_tikka;
141     length valid_item $50;
142     valid_item=input(item, $inf_tikka.);
143  run;

NOTE: There were 9 observations read from the data set WORK.TROYS_TASTY_TIKKA.
NOTE: The data set WORK.VALID_RECIPE has 9 observations and 4 variables.
```

Table 1 demonstrates the Valid_recipe data set, in which the first six values of Item are valid, and the last three values are invalid—because sweet cream, yeast, and flour do not appear in the Ingredients data set—the control table (of master data) against which all recipes should be validated. *Perhaps the chef confused his naan and tikka masala recipes!!*

	item	quantity	measurement	valid_item
1	turmeric	1	speck	turmeric
2	garlic	5	cloves	garlic
3	chicken	8	breasts	chicken
4	cumin	1	bounty	cumin
5	coriander	20	seeds	coriander
6	salt	2	pinches	salt
7	sweet cream	2	pumps	
8	yeast	3	packages	
9	flour	8	handfuls	

Table 1. Valid_recipe Data Set Demonstrating Application of INF_TIKKA Informat

In this example, a new variable (Valid_item) is initialized, so the raw variable (Item) and transformed variable (Valid_item) can be compared. In other instances, user-defined informats can be used to transform data *in situ*—that is, by overwriting the raw variable (Item), or by not ingesting the raw variable when it is determined to be invalid (by the informat). Data quality or data transformation objectives ultimately will drive what type of operation is performed by a user-defined informat.

For example, the user-defined informat can also be applied directly by the INPUT statement as raw data are being ingested into SAS. And because the Start and Label values are initialized to missing values in

the final observation (of the prior DATA step that creates the INF_Tikka_data data set), raw values evaluated to be invalid will be detected by INF_TIKKA, and not ingested into the Troys_tasty_tikka_valid data set:

```
data troys_tasty_tikka_valid;
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20;
   input item : $inf_tikka. quantity measurement $;
run;
```

The log demonstrates the application of INF_TIKKA to Item as the raw data are ingested:

```
195  data troys_tasty_tikka_valid;
196     infile f dsd delimiter=',';
197     length item $50 quantity 8 measurement $ 20;
198     input item : $inf_tikka. quantity measurement $;
199  run;

NOTE: The infile F is:

      Filename=D:\sas\troys_tasty_tikka.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=179,

NOTE: 9 records were read from the infile F.
      The minimum record length was 16.
      The maximum record length was 21.
NOTE: The data set WORK.TROYS_TASTY_TIKKA_VALID has 9 observations and 3 variables.
```

Table 2 demonstrates the Troys_tasty_tikka_valid data set; note that the Item variable has now been initialized to missing values for the last three observations, as specified by the INF_TIKKA informat.

	item	quantity	measurement
1	turmeric	1	speck
2	garlic	5	cloves
3	chicken	8	breasts
4	cumin	1	bounty
5	coriander	20	seeds
6	salt	2	pinches
7		2	pumps
8		3	packages
9		8	handfuls

Table 2. Troys_tasty_tikka_valid Data Set Demonstrating Values Removed by the INF_TIKKA Informat

CLEANING YOUR TIKKA WITH A USER-DEFINED INFORMAT AND _ERROR_

In some cases, initializing an invalid value to missing is insufficient, and the SAS automatic _ERROR_ variable can impart additional functionality to a user-defined informat.

For example, the user-defined informat (INF_TIKKA) can be modified so that it will throw an exception whenever an invalid value (i.e., a value observed within a recipe but not found within the Ingredients control table) is detected. The following DATA step and FORMAT procedure create the INF_TIKKA_ERR informat, in which the final Label value is replaced with "_ERROR_" and in which an invalid value will now throw an exception:

```
data inf_tikka_err_data;
   set ingredients(rename=(ingredient=start)) end=eof;
   length label $32 fmtname $32 type $2 hlo $2;
   label=start;
   fmtname='inf_tikka_err';
   type='j';      * J denotes a character informat;
```

```
   hlo='';
   output;
   if eof then do;
      label='_ERROR_';  * throws an exception;
      start='';
      hlo='o';    * O denotes other (i.e., invalid);
      output;
      end;
run;

proc format cntlin=inf_tikka_err_data;
run;
```

The INPUT statement now applies the INF_TIKKA_ERR informat to the Item variable as it is being ingested into the Troys_tasty_tikka_valid data set:

```
data troys_tasty_tikka_valid;
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20;
   input item : $inf_tikka_err. quantity measurement $;
run;
```

The log demonstrates application of the INF_TIKKA_ERR informat, in which exceptions are now thrown—that is, _ERROR_ is initialized to 1, and a corresponding note is automatically printed to the log—whenever invalid values of Item are detected:

```
NOTE: The infile F is:

      Filename=D:\sas\troys_tasty_tikka.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=179,

NOTE: Invalid data for item in line 7 1-11.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---
-8----+-
7         sweet cream, 2, pumps 21
item=  quantity=2 measurement=pumps _ERROR_=1 _N_=7
NOTE: Invalid data for item in line 8 1-5.
8         yeast, 3, packages 18
item=  quantity=3 measurement=packages _ERROR_=1 _N_=8
NOTE: Invalid data for item in line 9 1-5.
9         flour, 8, handfuls 18
item=  quantity=8 measurement=handfuls _ERROR_=1 _N_=9
NOTE: 9 records were read from the infile F.
      The minimum record length was 16.
      The maximum record length was 21.
NOTE: The data set WORK.TROYS_TASTY_TIKKA_VALID has 9 observations and 3 variables.
```

Table 2 again demonstrates the Troys_tasty_tikka_valid data set—that is, the identical data set has been created by applying the new informat. However, exceptions have now been thrown (by leveraging _ERROR_), which can facilitate additional functionality through programmatic evaluating of _ERROR_.

For example, the following DATA step uses conditional logic (based on the value of _ERROR_) to initialize the Msg variable to a value to be printed to the log:

```
data troys_tasty_tikka_valid;
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20 msg $20;
   input item : $inf_tikka_err. quantity measurement $;
   if _error_=0 then msg='Tikka approved!';
   else msg='Get outta my tikka!!!';
   put @1 _error_ @3 msg;
run;
```

The log (with emphasis added) demonstrates that the first six observations are tikka-approved, whereas the final three are not:

```
NOTE: The infile F is:
```

```
      Filename=D:\sas\troys_tasty_tikka.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=179,

0 Tikka approved!
0 Tikka approved!
0 Tikka approved!
0 Tikka approved!
0 Tikka approved!
0 Tikka approved!
NOTE: Invalid data for item in line 7 1-11.
1 Get outta my tikka!!
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---
-8----+-
7         sweet cream, 2, pumps 21
item=  quantity=2 measurement=pumps msg=Get outta my tikka!! _ERROR_=1 _N_=7
NOTE: Invalid data for item in line 8 1-5.
1 Get outta my tikka!!
8         yeast, 3, packages 18
item=  quantity=3 measurement=packages msg=Get outta my tikka!! _ERROR_=1 _N_=8
NOTE: Invalid data for item in line 9 1-5.
1 Get outta my tikka!!
9         flour, 8, handfuls 18
item=  quantity=8 measurement=handfuls msg=Get outta my tikka!! _ERROR_=1 _N_=9
NOTE: 9 records were read from the infile F.
      The minimum record length was 16.
      The maximum record length was 21.
NOTE: The data set WORK.TROYS_TASTY_TIKKA_VALID has 9 observations and 4 variables.
```

As a final example, programmatic evaluation of _ERROR_ can be utilized to retain only those observations for which no exception was detected. The following DATA step creates the Troys_tasty_tikka_only_valid data set time, only those observations that lack invalid data (i.e., _ERROR_=0) are ingested:

```
data troys_tasty_tikka_only_valid;
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20;
   input item : $inf_tikka_err. quantity measurement $;
   if _error_=0 then output;
run;
```

The log demonstrates that Troys_tasty_tikka_only_valid is created having only six observations (of the nine observations that were read):

```
NOTE: The infile F is:

      Filename=D:\sas\troys_tasty_tikka.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=179,

NOTE: Invalid data for item in line 7 1-11.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---
-8----+-
7         sweet cream, 2, pumps 21
item=  quantity=2 measurement=pumps _ERROR_=1 _N_=7
NOTE: Invalid data for item in line 8 1-5.
8         yeast, 3, packages 18
item=  quantity=3 measurement=packages _ERROR_=1 _N_=8
NOTE: Invalid data for item in line 9 1-5.
9         flour, 8, handfuls 18
item=  quantity=8 measurement=handfuls _ERROR_=1 _N_=9
NOTE: 9 records were read from the infile F.
      The minimum record length was 16.
      The maximum record length was 21.
NOTE: The data set WORK.TROYS_TASTY_TIKKA_ONLY_VALID has 6 observations and 4 variables.
```

Table 3 displays Troys_tasty_tikka_only_valid, in which only the first six observations are ingested:

	item	quantity	measurement
1	turmeric	1	speck
2	garlic	5	cloves
3	chicken	8	breasts
4	cumin	1	bounty
5	coriander	20	seeds
6	salt	2	pinches

Table 3. Troys_tasty_tikka_only_valid Data Set with Invalid Observations Not Ingested

Note that the _ERROR_ automatic variable automatically resets to 0 when a new observation is read, so the detection of an exception on a prior observation will not influence the values of _ERROR_ during later observations. However, if multiple variables are being evaluated by multiple user-defined informats within a single observation, each variable will need to be evaluated singly—that is, by a separate INPUT statement that ends with a trailing @ operator—and _ERROR_ will need to be evaluated, saved to a separate variable, and reinitialized to 0 after each separate INPUT statement. This methodology lies beyond the scope of this text, but is demonstrated in Section 12.5.2 of the author's text: *SAS® Data-Driven Development: From Abstract Design to Dynamic Functionality, Second Edition*. (Hughes, 2023)

CLEANING YOUR TIKKA WITH A DATA STEP HASH OBJECT

The hash object provides another methodology through which to validate categorical data—that is, to evaluate whether some value exists in (or *is a member of*) a lookup table. The hash CHECK method evaluates whether a value appears in a hash object, and returns 0 when the value is found.

For example, the following DATA step reads the Troys_tasty_tikka data set (in which both valid and invalid values are present), instantiates a hash object (H) by uploading the Ingredients data set, and utilizes the hash CHECK method to delete all values of Item (by initializing them to missing) not found in the hash object:

```
data troys_tasty_tikka_valid_hash (drop=ingredient rc);
   set troys_tasty_tikka;
   length ingredient $50;
   if _n_=1 then do;
      declare hash h(dataset: 'ingredients');
      rc=h.defineKey('ingredient');
      rc=h.defineDone();
      call missing(ingredient);
      end;
   if h.check(key: item)^=0 then item='';
run;
```

The Troys_tasty_tikka_valid_hash data set that is created is identical to the data set shown in Table 2, in which invalid values have been reinitialized to missing, yet all nine observations have been ingested. SAS hash objects provide an efficient, in-memory solution to data validation, cleaning, and transformation operations; however, the complexity of hash methods (such as DEFINEKEY, DEFINEDONE, and CHECK) can hinder their adoption by some SAS practitioners.

For this reason, one solution that simplifies hash code is to encapsulate hash functionality inside a user-defined function or subroutine (defined by the FCMP procedure). Thereafter, end users can call the user-defined function or subroutine using a single statement, which obviates hash complexity.

CLEANING YOUR TIKKA WITH A USER-DEFINED FUNCTION

Placing hash object functionality inside a user-defined function or subroutine simplifies the code performing the hash lookup because the hash object can be declared, instantiated, manipulated, and evaluated inside the FCMP procedure. This can greatly improve software readability as well as modularity and reusability.

The following FCMP procedure defines the MAKE_YOU_HOLLA_TIKKA_MASALA function, which evaluates whether some parameterized value appears in the Ingredients data set:

```
proc fcmp outlib=work.funcs.recipes;
   function make_you_holla_tikka_masala(ingredient $) $;
      declare hash h(dataset: 'ingredients');
      rc=h.defineKey('ingredient');
      rc=h.defineDone();
      if h.check()=0 then return(ingredient);
      else return('');
      endfunc;
quit;
```

If a value is passed to MAKE_YOU_HOLLA_TIKKA_MASALA that does not appear in the Ingredients data set, the function returns a missing value rather than the ingredient. This functionality can be used both to initialize a new variable and to overwrite an existing variable.

The following DATA step calls the MAKE_YOU_HOLLA_TIKKA_MASALA function to validate the Item variable, by transforming it *in situ* to a missing value if its value does not appear in the Ingredients data set:

```
options cmplib=work.funcs;

data troys_tasty_tikka_valid_hash;
   set troys_tasty_tikka;
   item=make_you_holla_tikka_masala(item);
run;
```

The Troys_tasty_tikka_valid_hash data set is identical to that created in the previous section (without the use of the user-defined MAKE_YOU_HOLLA_TIKKA_MASALA function), as well as the data set demonstrated in Table 2. Moreover, the complexity of the validation process has been simplified to a single invocation of the user-defined function.

However, note that the DATA step ingests the Troys_tasty_tikka *data set*, rather than the Troys_tasty_tikka *raw text file*. The next section demonstrates how a user-defined informat can be designed to call a user-defined function (such as MAKE_YOU_HOLLA_TIKKA_MASALA) to transform raw data as they are initially evaluated. This powerful combination can forego having to separate data ingestion and data validation operations into separate DATA steps.

CLEANING YOUR TIKKA WITH A USER-DEFINED INFORMAT THAT CALLS A USER-DEFINED FUNCTION

When the INF_TIKKA user-defined character informat was previously created, it required an entire DATA step that transformed and initialized required variables—Fmtname, Start, Label, Type, and HLO—to be leveraged by the subsequent FORMAT procedure CNTLIN option. Although emblematic of data-driven design, the CNTLIN method is far from straightforward.

However, informats (and formats) can also be created by utilizing the OTHER option to call a user-defined format. Thus, the following FORMAT procedure creates the INF_TIKKA_HASH informat, but indirectly relies on the MAKE_YOU_HOLLA_TIKKA_MASALA user-defined function rather than a specially formatted control table:

```
proc format;
   invalue $ inf_tikka_hash other=[make_you_holla_tikka_masala()];
run;
```

Moreover, the informat still demonstrates data-driven design because the user-defined function relies on the underlying Ingredients data set—the control table from which the function's hash object is instantiated. Thus, the same Ingredients data set can be operationalized as a control table through various methods.

And because data validation is again delivered through a user-defined informat, the INPUT statement in the following DATA step applies the INF_TIKKA_HASH informat as the raw Item values are ingested from the text file:

```
data troys_tasty_tikka_valid;
```

```
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20;
   input item : $inf_tikka_hash. quantity measurement $;
run;
```

The Troys_tasty_tikka_valid_hash data set is identical to that created in the previous section (without the use of the user-defined MAKE_YOU_HOLLA_TIKKA_MASALA function), as well as the data set demonstrated in Table 2.

Note one limitation of this methodology, however, in that because the OTHER option is utilized to call the user-defined function, OTHER cannot be used to initialize the _ERROR_ variable to 1 to throw an exception. And because the user-defined function does not have access to the program data vector (PDV) of the data set to which the user-defined informat is being applied, neither can the user-defined function access, return, or modify the _ERROR_ automatic variable.

With minimal finagling, however, the DATA step can be redesigned to again throw exceptions and exclude observations having invalid data—while still relying on the INF_TIKKA_HASH user-defined informat, created dynamically from the underlying MAKE_YOU_HOLLA_TIKKA_MASALA user-defined function. The following DATA step evaluates whether Item is missing, and because INF_TIKKA_HASH transforms invalid values into missing values, subsequently throws an exception (by initializing _ERROR_) when Item is invalid, and retains the observation when Item is valid:

```
data troys_tasty_tikka_only_valid;
   infile f dsd delimiter=',';
   length item $50 quantity 8 measurement $ 20;
   input item : $inf_tikka_hash. quantity measurement $;
   if missing(item) then _error_=1;
   else output;
run;
```

The log demonstrates that although nine observations were evaluated, only six—those having a valid Item value—were retained:

```
NOTE: The infile F is:

      Filename=D:\sas\troys_tasty_tikka.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=179,

RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---
-8----+-
7         sweet cream, 2, pumps 21
item=  quantity=2 measurement=pumps _ERROR_=1 _N_=7
8         yeast, 3, packages 18
item=  quantity=3 measurement=packages _ERROR_=1 _N_=8
9         flour, 8, handfuls 18
item=  quantity=8 measurement=handfuls _ERROR_=1 _N_=9
NOTE: 9 records were read from the infile F.
      The minimum record length was 16.
      The maximum record length was 21.
NOTE: The data set WORK.TROYS_TASTY_TIKKA_ONLY_VALID has 6 observations and 3 variables.
```

The Troys_tasty_tikka_only_valid data set is identical to the data set shown in Table 3. However, it achieves validation not through CNTLIN convolutions but rather through a user-defined informat that calls a user-defined function.

This simplicity ensures that a single control table—the Ingredients data set of master data—can be simultaneously leveraged to design a user-defined function and a user-defined informat, both of which are separately capable of validating ingredient values. And through this steadfast adherence to master data management (MDM) principles, whenever this single control table is updated, both the derivative user-defined function (MAKE_YOU_HOLLA_TIKKA_MASALA) and user-defined informat (INF_TIKKA_HASH) can be regenerated (without alteration of any SAS code), and both will be updated to reflect the modified control table. This is the simplicity and power of data-driven software design.

CONCLUSION

Tikka masala is an amazing amalgam of sweet and spicy, salty and savory, that leaves customers sweaty and breathless yet begging for more. So, too, the simplicity with which data-driven design can be leveraged to validate, standardize, and otherwise transform raw data as they are ingested into SAS data sets will savagely satisfy SAS practitioners. This text demonstrated methods in which a single control table—a data set enumerating a list of valid values—can be used simultaneously both to create a user-defined function and a user-defined informat. Moreover, use of the _ERROR_ automatic SAS variable was demonstrated, in which exceptions can be thrown during data ingestion, both directly via informats that leverage the CNTLIN option, and indirectly via informats that leverage the OTHER option to call a user-defined function.

REFERENCES

Hughes, T. M. (2023). *SAS® Data-Driven Development: From Abstract Design to Dynamic Functionality, Second Edition.* San Diego: Kindle Direct Publishing.

CONTACT INFORMATION

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

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