

Paper 4702-2020

**SAS® Data-Driven Design: How to Develop More Flexible, Configurable, Reusable Software through Data Independence of Control Data**

Troy Martin Hughes

**ABSTRACT**

Data-driven design describes software design in which the control logic, program flow, business rules, data models, data mappings, and other dynamic and configurable elements are abstracted to control data that are interpreted by (rather than contained within) code. Thus, data-driven design leverages parameterization and external data structures (including configuration files, control tables, decision tables, data dictionaries, business rules repositories, and other control files) to produce dynamic software functionality. This hands-on workshop introduces real-world scenarios in which the flexibility, configurability, reusability, and maintainability of SAS® software are improved through data-driven design methods, as introduced in the author’s 2019 book: *SAS® Data-Driven Development: From Abstract Design to Dynamic Functionality*. This white paper highlights one of these scenarios, in which malleable comma-separated values (CSV) files—where the variables and their order can vary—are ingested with the aid of a data dictionary control table.

**EXPENSE REPORTS**

Consider the use case in which an infinitely unmanageable paper supply company maintains expense reports as Excel spreadsheets. As companies tend to do, over time, the expense report format—including both columns and column order—might change. For example, Tables 1 and 2 demonstrate expense reports having dissimilar formats, as submitted by two salesmen, Jim and Dwight

	A	B	C	D
1	Employee	Cost	Expense	Vendor
2	Jim	4.50	parking	City of Scranton
3	Jim	65.19	client dinner	Poor Richard's
4	Jim	4.00	glasses	Rite Aid
5	Jim	5.00	mustard shirt and tie	Goodwill
6	Jim	2.00	calculator watch	
7				

**Table 1. Jim’s Expense Report**

	A	B	C	D
1	Cost	Expense	Vendor	Justification
2	7.99	bear spray	Field & Stream	I sometimes encounter bears on my way to work and need to defend myself when not carrying my nunchucks.
3	14.99	red Solo cups	Rite Aid	Michael wanted cups for a urinalysis on Toby and other employees, but mostly Toby.
4	47.53	magic supplies	Mel's Magic City	Not for Michael.
5	57.27	woman's wig	Sally Beauty	You never know when you're going to need to bear a passing resemblance to someone.
6				

**Table 2. Dwight’s Expense Report**

Note that Jim’s expense report includes an Employee column, whereas Dwight’s does not, and that Dwight’s expense report includes a Justification column, whereas Jim’s does not. Jim’s report can be saved as Jim\_APR.csv so that it can be more readily ingested by SAS software:

```
Employee, Cost, Expense, Vendor
Jim, 4.50, parking, City of Scranton
Jim, 65.19, client dinner, Poor Richard's
Jim, 4.00, glasses, Rite Aid
Jim, 5.00, mustard shirt and tie, Goodwill
Jim, 2.00, calculator watch,
```

Dwight’s report similarly can be saved as Dwight\_OCT.csv:

```
Cost, Expense, Vendor, Justification
7.99, bear spray, Field & Stream, I sometimes encounter bears on my way to work and
need to defend against them when I'm not carrying my nunchucks.
14.99, red Solo cups, Rite Aid, "Michael asked me to buy cups for a urinalysis on Toby
and other employees, but mostly Toby."
47.53, magic supplies, Mel's Magic City, Not for Michael.
57.27, woman's wig, Sally Beauty, You never know when you're going to need to bear a
passing resemblance to someone.
```

In a traditional, code-driven (i.e., hardcoded) development paradigm, these dissimilar spreadsheets would need to be ingested with separate SAS programs—because the programs lack flexibility. For example, the following hardcoded program ingests Jim’s (but not Dwight’s) expense report:

```
%let exp=/folders/myfolders/datadriven/;

filename f "&exp.jim_apr.csv";
data expenses;
  infile f dsd truncover firstobs=2;
  input employee : $10. cost : dollar11.2
        cost_description : $40. vendor : $30.;
  format employee $10. cost dollar11.2
        cost_description $40. vendor $30.;
run;
```

The INPUT and FORMAT statements would need to be overhauled to ingest Dwight’s report instead. Data-driven design, however, can leverage a data dictionary that describes all possible columns (i.e., variables) that could potentially be ingested. Each expense report can subsequently be queried to detect which variables appear in the header column, after which those columns can be dynamically ingested. Table 3 demonstrates the data dictionary that describes all anticipated columns, thus including columns found only in Jim’s report, columns found only in Dwight’s report, and columns found in both reports.

	A	B	C	D	E	F	G	H	I	J	K
1	Header	Name	Type	Inlength	Informat	Length	Format	Label	VarReq	In1	Out1
2	employee	fname	char	20		20		First Name			
3	cost	cost	num	8	dollar11.2	8	dollar11.2	Cost	Y	1	1
4	expense	cost_descri ption	char	40		40		Good or Service	Y	2	2
5	vendor	vendor	char	30		30		Vendor	Y	3	3
6	justification	justification	char	200		200		Justification	Y	4	4
7											

**Table 3. Expense Report Data Dictionary**

Once the data dictionary has been saved as a CSV file, it can be subsequently ingested into a SAS data set utilizing the INPUT statement within a DATA step. The Excel spreadsheet can be saved as Expense\_report\_data\_dict.csv:

```
Header,Name,Type,Inlength,Informat,Length,Format,Label,VarReq,In1,Out1
employee,fname,char,20,,20,,First Name,,,,
cost,cost,num,8,dollar11.2,8,dollar11.2,Cost,Y,1,1
expense,cost_description,char,40,,40,,Good or Service,Y,2,2
vendor,vendor,char,30,,30,,Vendor,Y,3,3
justification,justification,char,200,,200,,Justification,Y,4,4
```

The following data-driven solution (READ\_SCHEMA macro) interrogates the data dictionary and uses it to ingest expense reports:

```
* read_schema_5.sas;
%macro read_schema_5(schema= /* CSV data dict */,
  csvfile= /* [OPT] CSV file being ingested */,
  missvar= /* [OPT] WARN or ERR to throw except */,
  xtravar= /* [OPT] WARN or ERR to throw except */);
%let syscc=0;
%global ZZinfilePos ZZinfileCSV ZZinputPos ZZinputCSV
  ZZformat ZZlabel ZZputPos zzputCSV ZZlen ZZkeep
  ZZmissVarList ZZxtraVarList;
%local found;
%let ZZinfilePos=truncover;
%let ZZinfileCSV=truncover dsd firstobs=2;
%let ZZinputPos=;
%let ZZinputCSV=;
%let ZZformat=;
%let ZZlabel=;
%let ZZputPos=;
%let ZZputCSV=;
%let ZZlen=;
%let ZZkeep=;
%let ZZmissVarList=; * list of missing variables;
%let ZZxtraVarList=; * list of extra variables;
* read header line if CSVFILE is provided;
%if %length(&csvfile)>0 %then %do;
  filename c "&CSVfile";
  data tempheader;
    length header $32 varnum 3;
    header='err';
    varnum=0;
    infile c dsd truncover firstobs=1 obs=1;
    do while(^missing(header));
      input header : $32. @;
      header=lowcase(header);
      varnum=varnum+1;
      if ^missing(header) then output;
    end;
  run;
%end;
* read data dictionary schema;
filename fschema "&schema";
data tempschema;
  infile fschema dsd truncover firstobs=2;
  input header : $32. name : $32. type : $5.
    inlength : 8. informat : $32. length : 8.
    format : $32. label : $32. varreq : $3.
    in1 : 3. out1 : 3.;
run;
%if %length(&csvfile)>0 %then %do;
  * join CSV header with data dictionary;
```

```

proc sql noprint;
  create table tempheader2 as
  select case
    when missing(b.header) then a.header
    when ^missing(b.header) then b.header
    else ''
  end as header, a.varnum, b.name, b.type,
    b.inlength, b.informat, b.length,
    b.format, b.varreq, b.label, b.inl, b.outl
  from tempheader a
  left join tempschema b
  on lowercase(a.header)=lowercase(b.header)
  order by a.varnum;
quit;
data tempheader2 (drop=varnum);
  set tempheader2;
  retain i 0;
  inl=varnum;
  outl=varnum;
  if missing(type) then do;
    i=i+1;
    name='_var_'||strip(put(i,3.));
    type='char';
    inlength=32;
    informat='';
    length=32;
    format='';
    label=header;
  end;
* enumerate extra variables;
proc sql noprint;
  select a.header
  into : ZZxtraVarList separated by ','
  from tempheader a
  left join tempschema b
  on lowercase(a.header)=lowercase(b.header)
  where upcase(b.varreq)!='Y';
quit;
%let found=False;
%if %length(&ZZxtraVarList)>0 %then %do;
  %if %length(&xtravar)>=3 %then %do;
    %if "%upcase(%substr(&xtravar,1,3))"="WAR"
      %then %do;
      %put WARNING: Unexpected Column(s) in CSV file: &ZZxtraVarList;
      %if &syscc<4 %then %let syscc=4;
      %let found=True;
      %end;
    %else %if "%upcase(%substr(&xtravar,1,3))"
      ="ERR" %then %do;
      %put ERROR: Unexpected Column(s) in CSV file: &ZZxtraVarList;
      %if &syscc<5 %then %let syscc=5;
      %let found=True;
      %end;
  %end;
  %if &found=False %then %do;
    %put NOTE: Unexpected Column(s) in CSV file: &ZZxtraVarList;
  %end;
%end;
* enumerate missing variables;
proc sql noprint;
  select a.header
  into : ZZmissVarList separated by ','
  from tempschema a

```

```

left join tempheader b
on lowercase(a.header)=lowercase(b.header)
where upcase(a.varreq)='Y' and b.header is null;
quit;
%let found=False;
%if %length(&ZZmissVarList)>0 %then %do;
  %if %length(&missvar)>=3 %then %do;
    %if "%upcase(%substr(&missvar,1,3))"
      ="WAR" %then %do;
      %put WARNING: Missing Column(s) in CSV file: &ZZmissVarList;
      %if &syscc<4 %then %let syscc=4;
      %let found=True;
      %end;
    %else %if "%upcase(%substr(&missvar,1,3))"
      ="ERR" %then %do;
      %put ERROR: Missing Column(s) in CSV file: &ZZmissVarList;
      %if &syscc<5 %then %let syscc=5;
      %let found=True;
      %end;
    %end;
  %if &found=False %then %do;
    %put NOTE: Missing Column(s) in CSV file: &ZZmissVarList;
    %end;
  %end;
%end;
%else %do;
proc sort data=tempschema out=tempheader2;
  by in1;
  where ^missing(in1);
run;
%end;
* create informat, infile;
data _null_;
  set tempheader2 end=eof;
  length inpPos $10000 inpCSV $10000 posin 8;
  retain inpPos ' ' inpCSV ' ' posin 1;
  inpPos=catx(' ',inpPos,'@' || strip(put(posin,8.))
  || ' ' || strip(name) || ' ' ||
  ifc(upcase(type)='CHAR','$','')
  || ifc(missing(informat),
  strip(put(inlength,8.1)),
  ifc(find(informat, '.'),strip(informat),
  strip(informat)||'.')));
  inpCSV=catx(' ',inpCSV,strip(name) || ' : ' ||
  ifc(upcase(type)='CHAR','$','')
  || ifc(missing(informat),
  strip(put(inlength,8.1)),
  ifc(find(informat, '.'),strip(informat),
  strip(informat)||'.')));
  posin=posin+inlength;
  if eof then do;
    call symputx('ZZinputPos',inpPos);
    call symputx('ZZinputCSV',inpCSV);
  end;
run;
* create outformat, outfile;
proc sort data=tempheader2 out=tempheader3;
  by out1;
  where ^missing(out1);
run;
data _null_;
  set tempheader3 end=eof;
  length ptPos ptCSV form lab len kp $10000 posout 8;

```

```

retain ptPos ' ' ptCSV ' ' form ' ' lab ' ' len ' '
      kp ' ' posout 1;
form=catx(' ',form,strip(name) || ' '
  || ifc(upcase(type)='CHAR','$','')
  || ifc(missing(format),
    strip(put(length,8.1)),ifc(find(format, '.'),
    strip(format),strip(format)||'. '));
lab=catx(' ',lab,strip(name) || '=' || ' ' ||
  strip(label) || ' ');
ptPos=catx(' ',ptPos,'@' || strip(put(posout,8.))
  || ' ' || strip(name) || ' ' ||
  ifc(upcase(type)='CHAR','$','')
  || ifc(missing(format),
    strip(put(length,8.1)),ifc(find(format, '.'),
    strip(format),strip(format)||'. '));
ptCSV=catx(' ',ptCSV,strip(name) || ' ' ||
  ifc(upcase(type)='CHAR','$','')
  || ifc(missing(format),
    strip(put(length,8.1)),ifc(find(format, '.'),
    strip(format),strip(format)||'. '));
len=catx(' ',len,strip(name) || ' '
  || ifc(upcase(type)='CHAR','$','')
  || strip(put(ifn(upcase(type)=
  'CHAR',length,min(length,8)),8.)));
kp=catx(' ',kp,strip(name));
posout=posout+length;
if eof then do;
  call symputx("ZZputPos",ptPos);
  call symputx("ZZputCSV",ptCSV);
  call symputx("ZZformat",form);
  call symputx("ZZlabel",lab);
  call symputx("ZZlen",len);
  call symputx("ZZkeep",kp);
end;
run;
%mend;

```

The READ\_SCHEMA macro creates several global macro variables that can be subsequently used within a DATA step to dynamically create SAS statements:

- &ZZPUTPOS – dynamic PUT statement for creating a flat file output file
- &ZZPUTCSV – dynamic PUT statement for creating a CSV output file
- &ZZFORMAT – dynamic FORMAT statement
- &ZZLABEL – dynamic LABEL statement
- &ZZLEN – dynamic LENGTH statement
- &ZZKEEP – dynamic KEEP statement

Thus, the following macro invocation ingests Jim's report to create the Expenses data set:

```

%read_schema(schema=&exp.expense_report_data_dict.csv,
  csvfile=&exp.jim_apr.csv,
  missvar=warn,
  xtravar=warn);

filename fin "&exp.jim_apr.csv";
data expenses;
  length &ZZlen;
  infile fin &ZZinfileCSV;
  input &ZZinputCSV;
  format &ZZformat;

```

```
label &ZZlabel;  
run;
```

The MISSVAR=WARN parameter specifies that a warning statement will be printed to the log when a required variable (as specified in the VarReq column of the data dictionary) is missing. Similarly, the XTRAVAR=WARN parameter specifies that a warning statement will be printed to the log when an extra variable (i.e., not required) is present. Thus, the current READ\_SCHEMA invocation produces the following statements to the SAS log, indicating the presence of Employee and the absence of Justification:

```
WARNING: Unexpected Column(s) in CSV file: employee  
NOTE: PROCEDURE SQL used (Total process time):  
      real time          0.05 seconds  
      cpu time           0.05 seconds  
WARNING: Missing Column(s) in CSV file: justification
```

MISSVAR and XTRAVAR can be changed from WARN to ERR to instead generate a runtime error. If neither WARN nor ERR is specified, a SAS note is printed to the log. Notes, warnings, and runtime errors by default are color-coded in the log to aid in detection. More useful than merely printing these warnings and runtime errors to the log, however, SAS automatic macro variables are additionally modified to reflect exceptions that are encountered. Thus, a warning will modify the value of &SYSCC to 4 and a runtime error will modify the value of &SYSCC to 5, and these values can be programmatically detected to facilitate subsequent exception handling and exception reporting (not demonstrated)—a far better solution than needlessly checking the log after each run of the program.

The benefit of data-driven design is the flexibility that the software provides. For example, the same READ\_SCHEMA macro—without modification—can now be used to create dynamic statements that ingest Dwight's expense report:

```
%read_schema(schema=&exp.expense_report_data_dict.csv,  
             csvfile=&exp.dwight_oct.csv,  
             missvar=warn,  
             xtravar=warn);  
  
filename fin "&exp.dwight_oct.csv";  
data expenses;  
  length &ZZlen;  
  infile fin &ZZinfileCSV;  
  input &ZZinputCSV;  
  format &ZZformat;  
  label &ZZlabel;  
run;
```

Because the LENGTH, INPUT, FORMAT, and LABEL statements are generated dynamically (rather than hardcoded), they flexibly adapt to any number of variables that are encountered, as well as adapt to variables that occur in different orders within respective CSV files. Both Jim and Dwight are satisfied because they can focus on selling paper (or office hijinks) rather than needlessly modifying software. Finally, if expense reports are modified in the future (e.g., by adding or removing columns), these changes can be made in the data dictionary only without the need to modify the underlying code or SAS macros.

## CONCLUSION

Data-driven design facilitates more flexible, configurable, reusable software—future-proofed software that, to the extent possible, resists needless future modifications. This text demonstrated the use of data dictionary control tables that prescribe variables that can be ingested from CSV files. Required variables can be specified utilizing the VarReq column within the data dictionary, and notes, warnings, and runtime errors can be automatically generated to identify both missing variables and extra variables within the CSV files being ingested. Finally, for those cases in which CSV column order can vary, the READ\_SCHEMA macro dynamically ingests columns into the correct variables. Data-driven design benefits not only developers, who can maintain code more efficiently, but also software end users, who can modify software results or output based on dynamic input such as parameters and data dictionary control tables.

## CONTACT INFORMATION

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

Name: Troy Martin Hughes

E-mail: troymartinhughes@gmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

## REFERENCES

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