

Paper SBC-135

Using Data to Write SAS[®] Programs

Kevin McGowan and Brian Spruell, SRA International, Durham, NC

Abstract

On occasion SAS programmers need to write programs that have a lot of programming statements that can be generated from data that is already in electronic format. Examples of this include using data to write input statements, SAS macros, formats, and if -then statements. These programming techniques are particularly helpful when the data is received from an external source in electronic format and the description of the data is also in electronic format. Using data to write SAS programs cuts down on programming time as well as errors in programming. This paper will show examples of using data several different ways to generate SAS code that can be used in other SAS programs.

Introduction

When people think of writing a program in SAS or other programming language they think about writing the source code line by line. In some cases source code can be generated by using data in electronic format instead of writing the code line by line. For SAS programs data can be used to generate input statements, format statements, if -then statements, put statements and macro language statements. Generating source code with a SAS program has several advantages – it saves time, it can reduce errors, and it can also be used to perform quality control checks on data or programs. Examples of data that can be used to write SAS programs are data dictionaries /codebooks, databases, spreadsheets, and reports.

Items to consider before using data to write a program

Is there a lot of data that can be used to write the program? If the amount of data is small it may not be worth the effort to write a program to translate the data into source code. If the amount of data is small but there might be a bigger set of data in the same format in the future then it probably is a good idea to use the data to write a program.

Is the data in an easy to access format? In most cases the data should be in electronic format such as a report, text file, spreadsheet or database for it to be used to create a SAS program. If the data is only available on paper it might be worthwhile to convert it to electronic format if doing so will allow the program to be used again in the future.

Can the program be reused? If the data that is used is in a standard format and more data will be received in the same format in the future then it is probably a good candidate for writing a program to convert the data into a SAS program. If the data is a one time only example then it might not be worth the time to convert it to a program. An

exception to that rule would be if there is a lot of data that would take a long time to write a standard SAS program.

Example 1 – Using data to write a SAS input statement

Many times when data is received from an external source the file is accompanied by a codebook or some other form of data in electronic format that describes the format of the data. If the data is in text format in many cases a SAS input statement needs to be written to read the data into a SAS dataset for processing. Rather than write the input statement by hand, the codebook can be read and used to generate the input statement.

If the data is in this format :

<u>Fname</u>	<u>Lname</u>	<u>Address</u>	<u>City</u>	<u>State</u>	<u>Zip</u>
John	Smith	12 Elm St.	Cary	NC	27511
Mary	Jones	11 Oak Rd.	Raleigh	NC	27609
Avery	Johnson	123 Blur Ln.	Apex	NC	27502

The codebook for the data would be:

<u>Variable Name</u>	<u>Variable Length</u>	<u>Variable Type</u>
Fname	20	Char
Lname	20	Char
Address	20	Char
City	10	Char
State	2	Char
Zip	5	Num

Then in order to convert this codebook to a basic SAS input statement the following code can be used:

```
Filename indata 'codebook.dat';  
Filename outsas 'input_statement.sas';
```

```
Data codein
```

```
  Infile indata dsd dlm='08'x; /* assumes data is tab delimited */
```

```
  Input vname vlength vtype;
```

```
File outsas;
```

```
Put 'input ';
```

```

Put vname vlength @;

If vtype='Char' then do;
    Put '$';
End;
Else do;
    Put ;
End;

```

The above example is very simple and a good starting point for a program that can be made more complex and flexible. One modification that is possible to this program is to use it on data that is not tab delimited, that would make the program more complex to write.

One point to note for this example is that programs that use data to write SAS code can be very simple ad hoc programs that do not have a lot of flexibility and error checking or they can be more flexible and have a lot of error checking. Deciding which type of program to write depends on how often the program will be reused. If the program is only going to be used one time then it can be less structured. If there is a need to reuse the program in the future then it is a good idea to make the program more robust including error checking and similar features.

Example 2 – Using data to write code to change values

There are cases where it is necessary to change the values of variables from the current value to a new value. An example of this would be cleaning data to get it ready for analysis. If the values used to change variables are already in electronic format then the data can be used to write SAS code to change the variables instead of writing the code by hand.

The data used to change variable values might look like this:

Varname	OldValue	NewValue
Bodywt	45	78
Bodywt	56	90
Kidwt	4	6
Kidwt	7	11
Brwt	14	19
Brwt	19	27
Lwt	21	25
Lwt	43	56

The code to read in this data and convert it to if-then statements that will change the data would look like this:

```
Filename indata 'change_data.dat';
Filename outsas 'recode_statement.sas';

Data codein
  Infile indata dsd dlm='08'x; /* assumes data is tab delimited */

  Input varname $ oldvalue newvalue;

  File outsas;

  Put 'if ' varname '=' oldvalue 'then ' varname '=' newvalue ';;';
```

The output from running the code above with the data will be:

```
if Bodywt =45 then Bodywt =78 ;
if Bodywt =56 then Bodywt =90 ;
```

The above example uses numeric values so it will work fine. If the data to be recoded is character strings then more care needs to be taken to make sure the code will work. The reason for this is that when comparing character strings spaces are important while spaces are not relevant when comparing numeric values.

If x='Joe' is not the same as if x=' Joe' since there is an extra space before the J in the second example. If comparison code does not work with character data, it is a good idea to check for extra spaces with character variables since that is an error that often goes undetected.

Example 3 – Using data to create SAS formats

SAS formats can be a very valuable tool when used with data because formats can make data much more readable. Just as in the above examples with input statements and recode statements, SAS formats can be generated from data to save time as opposed to entering the format statements by hand. This is probably the most well known example of generating code from data in SAS because it actually uses a built in feature of proc format to create formats without actually generating SAS programming code.

The first step to create formats from data is to create a format control dataset. The dataset should have these variables: label, start, end, fmtname, type and hlo.

An example dataset is:

Label	Start	End	Fmtname	Type	Hlo
Low	0	50	Wt_fmt	N	
Medium	51	100	Wt_fmt	N	
High	101	150	Wt_fmt	N	
Very High	151	1000	Wt_fmt	N	

If the above dataset is called `fmt_dset` the SAS code to create the format `Wt_fmt` is

```
Proc format cntlin=fmt_dset;
```

It is possible to create more than one format with a single `proc format` statement as long as all the format definitions are stored in the same dataset that is used as the input to the `proc format` statement.

Example 4 – Using data to write code to enhance HTML pages

SAS has the ability to create very nice HTML documents through the use of ODS. In some cases there is a need to create more customized HTML code than is generated by SAS ODS. The use of ODS provides a great deal of flexibility but sometimes it is necessary to go beyond basic ODS and add extra HTML tags to make the output look better. For example, if you are creating a codebook with ODS in most cases it is going to look something like this:

Variable	Variable Type	Variable Label	Variable Source	Values
Age	Num	Respondent Age	Main	5-100
Sex	Char	Sex	Main	M,F
State	Char	Address State	Main	AL, AK, CA

The code that generated this table used one variable for each cell. For the ‘Values’ column the data was compressed together and it does not look as good as it could. By using extra HTML tags the table could look like this:

Variable	Variable Type	Variable Label	Variable Source	Values
Age	Num	Respondent Age	Main	5-100
Sex	Char	Sex	Main	M,F
State	Char	Address State	Main	AL, AK CA

The way to create the second table that looks better is to add HTML tags as part of the data step processing so they can work along with the HTML that is created by the SAS ODS. The HTML code that will create the cell for state values looks like this:

```
<dl>
<dt>AL
<dt>AK
<dt>CA
</dl>
```

The important point to remember here is that even though ODS is going to generate HTML code that does not preclude a programmer from generating their own HTML code as well. Care should be taken to make sure any code that is generated through the data step or other programming measures does not conflict with the HTML code generated by ODS.

Example 5 – Using data to write or call a SAS Macro program

Most SAS programmers know that using the macro language is a very good way to save time by cutting down on the number of programming statements that are written by the programmer. When the same section of SAS code needs to be run for many different groups of data the macro language is the best way to get that done. The macro language also greatly reduces the amount of debugging needed since the source code only needs to be tested one time before it is run many times using different data. This example shows how data can be used to generate the macro calls that are used to run the SAS code stored in a macro program.

If you have a macro called run_report1 that takes 4 parameters – species, sex, vehicle, and route and the parameters are stored in a text file with this format:

```
rats    male   oral   water
rats    female oral   water
mice    male   air    inhalation
mice    female air    inhalation
rats    male   oral   food
rats    female oral   food
```

The SAS code to read in the parameters and write the macro calls would be:

```
Filename indata 'params.dat';
Filename outsas 'macro_statements.sas';
```

```
Data params;
  Infile indata dsd dlm='08'x; /* assumes data is tab delimited */
```

Input species \$ sex \$ route \$ vehicle \$;

File outsas;

Put “%run_report1(“ species “,” sex “,” route “,” vehicle “);”;

The output from the above SAS code will be:

```
%run_report1(rats ,male ,oral ,water );  
%run_report1(rats ,female ,oral ,water );  
%run_report1(mice ,male ,air ,inhalati );  
%run_report1(mice ,female ,air ,inhalati );  
%run_report1(rats ,male ,oral ,food );  
%run_report1(rats ,female ,oral ,food );
```

An important note about generating macro calls using this technique is that this will actually cause the macro to be executed. If you do not want the macro to be executed at the time this code is run the easy way to do that is to not have the macro code part of the code that is submitted to SAS. One reason you might not want the macro executed is if you are not yet done writing or debugging the macro. The other way to avoid the macro from being run while you are generating the macro code is to set the SAS system option macro to nomacro.

Example 6 – Using data that is generated in the program to write SAS code

The previous examples all use data that exists before the SAS code is run in order to generate SAS code. In some cases there may be a need to generate SAS code based on the results of data steps or procedure output. This allows for more flexibility since the code is self contained and does not need to rely on external files to run.

This example shows how to take the output from proc means and use the max value from one data set to change the values in another data set.

```
Proc means max noprint data=dset1;  
  Var height;  
  Output out=dset1_max max=ht_max;
```

```
Data _null_;  
  Set dset1_max;
```

```
  Call symput (“ht_max”, ht_max);
```

```
Data two;  
  Set dset2;
```

```
If height > &ht_max then height = &ht_max;
```

```
Run;
```

One problem that can occur in the above code is that the macro value ht_max is assumed to exist. If there was a problem in creating ht_max then the code in the second dataset will not work because &ht_max will have a blank value. In order to prevent this from happening the simplest technique is to use the %symexist function to make sure the macro value ht_max has been created before running the code. The syntax for the %symexist function for this example is:

```
%if %symexist(ht_max) %then %do;
```

```
    If height > &ht_max then height = &ht_max;
```

```
%end;
```

It is also possible to execute a group of code based on data that is output from a proc. That would be done using the following macro technique:

```
%if %symexist(ht_max) %then %do;
```

```
    %If &ht_max > 50 %then %do;
```

```
        (section of code to be executed if above statement is true)
```

```
    %end;
```

```
%end;
```

There are many different ways to use data from one part of a program to control the execution of another part of a program through the use of the macro language. As mentioned above it is important to make sure that the code does not assume macro variables are created because that can lead to problems. Use of the %symexist function can avoid most of the problems in this area.

Conclusion

The above examples give just a small glimpse into the different ways data can be used to help write SAS programs. Once a programmer starts using data to write programs they can normally find other techniques that are more complex than the ones mentioned in this paper. One note of caution is that not every group of data is suitable to be used to write programs. The programmer should weigh the pros and cons of using data to write

programs to determine if it is the best way to write a particular program. If the program is going to be reused it is probably a good idea to write the program as a macro with parameters so it can have flexibility to be used for many different types of data.

Contact Information

Kevin McGowan
Associate Technical Manager
SRA International Inc.
2605 Meridian Parkway
Durham NC 27713

(919) 313-7554

Kevin_mcgowan@sra.com

<http://www.sra.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.