

Die Macro Die!

Daniel Olguin, First Coast Service Options, Jacksonville, FL

ABSTRACT

Have you ever tried to work your way through some convoluted SAS® Macro Language code and thought to yourself, there's got to be an easier way. SAS Macro language syntax misleads many a SAS newbie into thinking in terms of subroutines and functions. It comes as quite a shock when they learn that SAS Macro languages sole purpose is to produce text! If the goal is to produce text and every programming language in existence can produce text, it's a good bet that there are better tools for the job. This paper will demonstrate how Ruby and StringTemplate, both open-source tools, can make life simpler.

Keywords: Ruby, JRuby, macro, open-source, stringtemplate

INTRODUCTION

Alan Kay is quoted as saying, "Simple things should be simple. Complex things should be possible". This has become an oft quoted basic tenet of Software Design. Generating text has to be one of the simplest of programming tasks and therefore it should be simple to do. Yet, if we're to judge by the questions posed on SAS-L it's clear that the SAS Macro Language poses more challenges for SAS users than would be expected for so simple a task. I believe that can be attributed to the Macro languages misleading syntax and a poor design which fails to clearly separate the template (i.e. the text to be generated) from the code that uses the template, resulting in unessential complexity. I'll briefly give examples of these issues then show how Ruby's simplicity and StringTemplate's well thought out design can be used as alternatives.

Misleading syntax

A common newbie mistake is to attempt to use SAS Macro Language as a subroutine or function. Look at the following code:

```
%macro subroutine();  
%put In subroutine;  
%mend;  
data _null_;  
    put 'Before subroutine';  
    call execute('%subroutine()');  
    put 'After subroutine';  
run;
```

If macros were subroutines or functions, one would expect that the order of text in the log would be:

Before subroutine
In subroutine
After subroutine

If fact, that is the order produced by the above code. Therefore, we're safe in assuming that macros can be used as subroutines or functions, right? Wrong!

We can see this if we make a small addition to the macro:

```
%macro subroutine();
%put In subroutine;
data _null_;
    Put 'In data step inside subroutine';
run;
%mend;
data _null_;
    put 'Before subroutine';
    call execute('%subroutine()');
    put 'After subroutine';
run;
```

Now running this produces:

```
Before subroutine
In subroutine
After subroutine
In data step inside subroutine
```

Clearly our expectations (influenced by our knowledge of subroutines and functions in other languages) and the actual interpretation and execution of the SAS Macro Language differ.

Of course, there are those who would argue that the fault is lack of understanding which can be rectified by taking any number of expensive courses, buying any number of expensive books, or by paying for any number of expensive contractors.

I say, nonsense! The syntax should NOT imply a standard usage by its structure and then deviate from it. This violates the Principle of Least Astonishment.

Unessential vs. Essential Complexity

An example from a popular text on Macro programming is given below:

```
%let names=%nrstr(0%'DONOVAN&0%'HARA&0%'MALLEY);
%let name3=%bquote(%scan(%bquote(&names),3));
%put &names;
%put NAME 3 is: &name3;
```

One is immediately struck by the noisy syntax. Is it really necessary for all that noise when all we want is the 3rd name in the string? How much of that code is Essential Complexity; that is, complexity inherent in the problem itself and how much is Unessential Complexity; the complexity of poor design? If you think I'm being too harsh, simply attempt

the same thing in the DataStep. You'll find that you can do it with much less noise; i.e. unnecessary syntax. Ask yourself why that should be so. Why two different syntaxes, from the same vendor, to accomplish essentially the same thing?! Why is a second language required at all?

SEPARATION OF CONCERNS

One of the causes of Unessential Complexity is the failure to follow the principle of Separation of Concerns. This fundamental principle of design states that the solution to any problem of sufficient complexity has multiple and separable concerns which should be isolated, as much as is possible, to reduce complexity.

The fundamental flaw of the SAS Macro language is that it fails to separate the template, the text structure, from the logic. Note that there are many templating engines which make the same mistake, but as your mother used to say, if your friends jumped off a cliff...

RUBY

The Ruby programming language developed by Yukihiro Matsumoto, Matz, was purposely designed to exclude unessential syntax. As such, it can be used to create succinct programs which clearly express their intent. It's interesting to note that when a language is designed with first principles in mind it is a pleasure to use, and to contrast this with the experience of working with a poorly designed language.

STRINGTEMPLATE

Terence Parr developed StringTemplate because he needed a way to generate text and didn't like the way existing template engines mixed creation of the text with the template itself. StringTemplate is an open-source project originally developed in Java which has since been ported to many languages. The Java version is of interest to us because through the use of the `javaobj` we can access the features of StringTemplate.

StringTemplate separates the concerns of text structure, the template, and the logic of processing. Look at the example template below.

```
hello.st
Hello $name$!
```

Notice it is simple text with markers (\$) to identify the beginning and ending of a named attribute. No logic just text. There is nothing to fool you into believing your executing a function. It's eminently clear what is static text, and dynamic fill-in the blank. What's more it's scalable because it's never more complex than static text and fill-in the blank.

Logic is handled by the programming language referencing the template. In Ruby it looks like this:

```
# A group is mapped to a directory where the template(s) can be found;
st_group = ST::StringTemplateGroup.new("group_name_here", "path_to_template")
# Retrieve a template (the .st is understood);
st = st_group.getInstanceOf("hello")
# Give a value to each attribute in the template;
st.setAttribute("name", "John")
# Get the resulting String from applying the attributes to the template;
result = st.to_s
```

The first 2 lines locate and retrieve the template. Following that, any number of lines of logic; leading up to the filling-in of the blanks in the template. Lastly, retrieval of the result. You use the same syntax to process the template as you use for any other task in the language. No special cases. No temporal conditions to worry about. No special syntax.

The above code adapted to SAS DataStep syntax for use with the javaobj:

```

* A group is mapped to a directory where the template(s) can be found;
script = 'st_group = ST::StringTemplateGroup.new("group_name_here",
"path_to_template")';
link Eval;

* Retrieve a template (the .st is understood);
script = 'st = st_group.getInstanceOf("hello")';
link Eval;

* Give a value to each attribute in the template;
script = 'st.setAttribute("name", "John")';
link Eval;

* Get the resulting String from applying the attributes to the template;
script = 'result = st.to_s';
link EvalToString;

```

Note that the verbosity is a result of the way the interface to the javaobj has been implemented. This could be alleviated by future designs of the javaobj interface.

What's in it for me?

Clearly there are those for whom complex is good. The more complex the software, the more that companies will require their services or need to buy their books. There will also be those who believe that SAS doesn't make design errors and we should all just accept the status quo.

If however, you're a customer who would like to use tools that are simple, elegant, and ease your tasks rather than getting in the way, you might consider the following:

- 1) Ruby, thanks to JRuby, and StringTemplate can be used anywhere that templating is needed; with or without SAS
- 2) Ruby, JRuby and StringTemplate are free and open-source
- 3) JRuby through the javaobj interface, not only gives you the ability to easily generate text but also user-defined functions, Object-Oriented classes and objects, GUIs; in fact anything you'd expect from a modern programming language. No half or three-quarters of a language. No costly add-ons.

You have the choice between software that is difficult to use and can only be used in limited context and at substantial cost or software that is easy to use has unlimited freedom and is open and freely available.

CONCLUSION

I'm a fan of SAS software but I don't allow that to blind me to its weaknesses. Where there exists a better choice it only makes sense to take the option. The combination of JRuby and StringTemplate make a better toolset for text generation than does the current incarnation of the SAS Macro Language. It is certainly possible that in the future SAS could bundle JRuby and StringTemplate into its Base SAS software and transition away from that very old, very tired workhorse, the SAS Macro Language.

Regardless, you can begin the transition now to a better, easier, more enjoyable programming experience.

REFERENCES

Carpenter's Complete Guide to the SAS Macro Language, Art Carpenter, SAS Press, 2004

SAS® Macro Programming Made Easy, Michelle M. Burlew, Cary NC, 2002

[A Functional Language For Generating Structured Text](#)

<http://www.stringtemplate.org/>

See also my other SESUG papers describing how you can benefit from using open-source software with SAS.

RECOMMENDED READING

Bringing Design to Software, Terry Winograd, Addison-Wesley, Inc. 1996

Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design, Larry L. Constantine and Lucy A.D. Lockwood, ACM Press, 1999

Don't Make Me Think: A Common Sense Approach to Web Usability, Steve Krug, New Riders Press, 2005

CONTACT INFORMATION

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

Daniel Olguin
2408 Spring Vale RD
Jacksonville, FL 32246
E-mail: olguind@bellsouth.net

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.

Complete Program Listings may be downloaded from <http://olguind.clearwire.net/downloads>