PhUSE 2011

Paper TS11

GENERICITY
- New Metadata Concepts Applied to SAS Macro Programming -

Wolf-Dieter Batz, Phenix-MTK GmbH, Kelkheim, Germany

ABSTRACT
Generic Programming is a coding technique that avoids or totally eliminates the use of explicit (hardcoded) information in the source code of a program. Instead, the program is equipped with some "search algorithm" or intelligence to make it find the required information itself from the runtime environment. The presentation will shortly introduce the ideas behind generic programming and then show or demonstrate live an application using these concepts. Special emphasis will be made to an implicit metadata structure referred to as RSDS (Report Specific Data Structure), which is highly efficient in reducing the number of parameters to be passed for a macro call.

INTRODUCTION
This paper is not about Macro Programming. It will not attempt to introduce the audience to the universe of the SAS Macro Facility, discussing latest how-to's dealing with macro triggers, symbol tables, runtime scopes, variable resolution algorithms or the various interfaces to enter or leave this universe. This paper is about Communication. It will raise the question of how to maintain control over of a macro's behavior when it gets more and more flexible, complex, or powerful in some other sense. The author claims that structural properties of the dataset processed can be utilized to control the macro. A basic approach is presented, of how a macro may extract information from a dataset's structure and behave accordingly.

MACRO COMMUNICATION

EXAMPLE CODE
Throughout this paper there will be a number of examples that all make use of the same coding structure and typographical conventions: (i) All code is embedded in a macro definition block; (ii) all macro processor triggering tokens are upper case; (iii) all other tokens are lower case.

```
%MACRO ts11;
… Code lines go here …
%MEND ts11;
options mprint mlogic;
%TS11;
options nomprint nomlogic;
```

Since this structure suits quite well for presentation purposes it is not claimed to be state-of-the-art programming or otherwise recommended in a production environment.

TALK TO ME!
People writing SAS code using the SAS Macro Facility do this for several reasons. One of these definitely emerges from a very personal desire to communicate. On first sight this may appear a bit strange to the naive everyday's character, but there is strong arguments to do so.
Assume that you want to output a well known dataset in partitions defined by one criteria and this criteria shall be values used to code the variable 'sex'. The following approach is likely to not work:

```
%MACRO ts11;
proc print
     data = sashelp.class
;
by sex;
run;
%MEND ts11;
options mprint mlogic;
%TS11;
```

```
options nomprint nomlogic;
```

Since the SAS System expects sorted data for BY group processing you might think for a second to precede the Proc Print with a Proc Sort to avoid unappreciated runtime behavior. But if you think twice, another option might appear a suitable path to go. Why not tell the macro explicitly what you want it to do? The resulting code could look like this:

```
%MACRO ts11(sex);
proc print
     data = sashelp.class
    (where = (sex = upcase("&SEX.")))
;
run;
%MEND ts11;
options mprint mlogic;
%TS11(m);
%TS11(f);
options nomprint nomlogic;
```

The macro now has some limited 'understanding' of what it shall do. Since there is a 'memory cell' that it checks during runtime, the macro has gained functional variability that appears as externally controlled behavior. The macro now provides a communication channel the user can utilize. Be aware, that, whatever terms we invent here, this program is far from being intelligent, and behavior is something commonly attributed to living systems. With respect to this everything presented here is almost likely based on illusion.
Nevertheless, it will definitely pay to move a bit further in investigating the options to communicate with a program by providing it with some kind of information.
The program already illustrates how information emerges from a simple data token. Since the program internally compares the given parameter values 'm' and 'f' to values of 'sex' these values denotate to 'male' and 'female' respectively. They have become information for the program, not merely data. It's quite easy to illustrate this difference:

```
%MACRO ts11(name);
proc print
     data = sashelp.class
    (where = (name = upcase("&NAME.")))
;
run;
%MEND ts11;
options mprint mlogic;
%TS11(m);
%TS11(f);
options nomprint nomlogic;
```

This variation will produce no output. Of course. The difference in behavior is a result from the altered denotation attempt that failed because the given data at invocation will return a message like "no rows affected" in the SAS LOG, at least as long as there are no single digit names coded. It was kind of easy to remember and use values from the dichotomized sex variable before, but this method will constantly fail when using an open set of values as is the case with names.

SEARCH!

At this point we are ready to extend the capabilities of the macro. We now add a few lines of code that do not enrich or otherwise improve the report, but code functionality to 'actively' generate information by doing research on the environment in reach. To keep it simple let's start using the gender grouping again.

```
%MACRO ts11(field);
%LOCAL lib dsn var val valuelist n_values i_values;
%LET lib = %SCAN(&FIELD.,1,.);
%LET dsn = %SCAN(&FIELD.,2,.);
%LET var = %SCAN(&FIELD.,3,.);
proc sql noprint;
select distinct &VAR.
  into :valuelist separated by '-'
  from &LIB..&DSN.
;
```

```
quit;
%LET n_values = &SQLOBS.;
%DO i_values = 1 %TO &N_VALUES.;
%LET val = %SCAN(&VALUELIST.,&I_VALUES.,-);
proc print
     data =  &LIB..&DSN.
    (where = (upcase(&VAR.) = upcase("&VAL.")))
;
run;
%END;
%MEND ts11;
options mprint mlogic;
%TS11(sashelp.class.sex);
options nomprint nomlogic;
```

Cool, isn't it? With very few lines of additional code and one explicit loop we enabled the macro to operate single value driven. Try this with other procedures apart from Proc Print or check out sorting the value list from Proc SQL by referring additional variables in an ORDER BY clause or use the values to dynamically populate titles footnotes and more.

GENERICITY

This macro is already generic, i.e. it does not refer to names and values inside the macro definition. Hence, the term 'generic' is used as opposed to 'hardcoded' from which follows that these terms are mutually exclusive. Nonetheless we may experience some unappreciated surprise when relying on this property if we, for example, try to use it with the name variable:

```
%MACRO ts11(field);
%LOCAL lib dsn var val valuelist n_values i_values;
%LET lib = %SCAN(&FIELD.,1,.);
%LET dsn = %SCAN(&FIELD.,2,.);
%LET var = %SCAN(&FIELD.,3,.);
proc sql noprint;
select distinct &VAR.
  into :valuelist separated by '-'
  from &LIB..&DSN.
;
quit;
%LET n_values = &SQLOBS.;
%DO i_values = 1 %TO &N_VALUES.;
%LET val = %SCAN(&VALUELIST.,&I_VALUES.,-);
proc print
     data =  &LIB..&DSN.
    (where = (upcase(&VAR.) = upcase("&VAL.")))
;
run;
%END;
%MEND ts11;
options mprint mlogic;
%TS11(sashelp.class.name);
options nomprint nomlogic;
```

Useless result from a correct run? Well, partially, as long as the dataset is small and, hence, as long as the names are unique. So let's fine tune the selection logic in order to liberalize addiction to single byte values:

```
%MACRO ts11(field);

%LOCAL lib dsn var val valuelist n_values i_values;
%LET lib = %SCAN(&FIELD.,1,.);
%LET dsn = %SCAN(&FIELD.,2,.);
%LET var = %SCAN(&FIELD.,3,.);
proc sql noprint;
select distinct substr(&VAR.,1,1)
  into :valuelist separated by '-'
  from &LIB..&DSN.
;
```

```
quit;
%LET n_values = &SQLOBS.;
%DO i_values = 1 %TO &N_VALUES.;
%LET val = %SCAN(&VALUELIST.,&I_VALUES.,-);
proc print
     data =  &LIB..&DSN.
    (where = (upcase(substr(&VAR.,1,1)) = upcase("&VAL.")))
;
run;
%END;
%MEND ts11;
options mprint mlogic;
%TS11(sashelp.class.name);
options nomprint nomlogic;
```

Since this paper is not aimed at being a training for interestees in the SAS Macro Facility, coding details and functionality will not be discussed here.
Still, the topic focused is communication, concentrating on how to tell a given macro what to do, i.e. to provide which facet of the implemented functionality after invocation. We're now closing the 'genericity' part of this 'macro communication' chapter with one more example. Titles added make use of the generated substrings and also illustrate a single-quote-in-string-in-string quoting scenario.

```
%MACRO ts11(field);
%LOCAL lib dsn var sig val valuelist n_values i_values;
%LET lib = %SCAN(&FIELD.,1,.);
%LET dsn = %SCAN(&FIELD.,2,.);
%LET var = %SCAN(&FIELD.,3,.);
%LET sig = %SCAN(&FIELD.,4,.);
proc sql noprint;
select distinct substr(&VAR.,1,&SIG.)
  into :valuelist separated by '-'
  from &LIB..&DSN.
;
quit;
%LET n_values = &SQLOBS.;
%DO i_values = 1 %TO &N_VALUES.;
%LET val = %SCAN(&VALUELIST.,&I_VALUES.,-);
title1 "Listing of all ""&VAL.%STR(%')s"" ";
proc print
     data =  &LIB..&DSN.
    (where = (upcase(substr(&VAR.,1,&SIG.)) = upcase("&VAL.")))
;
run;
%END;
%MEND ts11;
options mprint mlogic;
%TS11(sashelp.class.name.2);
options nomprint nomlogic;
```

Whereas in this example no crucial flexibility was added, it clearly shows that increased functionality is paid with an increasing effort for communication, thus raising complexity of parameter passing.
If no solution to circumvent this correlation is found then macros either remain on a somehow limited level of flexibility or do require massive communication efforts bearing a risk of malfunction or inappropriate use in general.

METADATA

Since metadata are assumed to be widely known to the PhUSE community this chapter restricts to relevant aspects for this paper only. Metadata tables are referred to as 'metatables'.

HANDLING COMPLEXITY

The parameter passing or 'communication' method used in the preceding chapter was based on the concept of segmented lists which is very powerful und brilliantly supported by the SAS System's Macro Facility which in turn comes out of the box with a number of custom methods and formats to supply parameters to a macro at invocation as well as runtime.

The issue raised at the preceding chapter's end though, is complexity. In advanced reporting where a high level of control over the appearance of output elements down to single values is required, complexity of parameter passing extends by far the options provided by explicit parameter supply, be they segmented, positioned or keyed.

The concept of metadata opens multidimensional universe of environment properties to a running program where there were only one-dimensional lists before. To put it in a picture, it is by far more efficient to paint a picture than to describe it, moreover, there may exist pictures, not at all accessible by verbatim description. Thus, the metadata approach indeed offers a new quality of macro communication (parameter passing) that would not be available otherwise.

MORE THAN SEARCH

The 'search' part of macro source code introduced before becomes relevant here since this is the place to process metadata. Whereas the beginner's search behavior was limited to get some helpful information about the dataset to be processed or value properties therein, the read-only processing of source datasets is now extended to read, process and write data structures which are specialized or aimed at describing or documenting source data, processing status and output properties, hence metadata. Closing picture: The animal behavior asking for the water pond has changed to the conscious and systematic workflow of a scientist in his lab or an accountant in his office.

A CASE STUDY WORKFLOW

Following this metaphor we start with the source data, i.e. with a metatable describing the source data. The following is an excerpt information that is taken from the so called data-dictionary, a repository that is maintained by most database systems and, of course, by the SAS System. To be precise here, this metatable called dictionary.columns gives the SAS user information about all the columns of all tables currently found in his active libraries apart from name, type, length and format which we use here. With a where clause applied to the columns LIBNAME and MEMNAME the table dictionary.columns lists all the columns of the thus specified table in the order they are stored.

Column Name	Column Type	Column Length	Column Format
AE_CO_ID	num	8	
DRUG	char	200	$200.
PT_CUR	char	250	$250.
PRIM_RR_COUNTRY_CODE_RNK	num	8	
AGE_AT_ONSET_YEARS_RNK	num	8	
GENDER_CODE_RNK	num	8	
RECEIVED_AT_COMPANY_DT_RNK	num	8	
CONSUMER_REPORT_FLAG_RNK	num	8	
RECEIVED_AT_COMPANY_DT	num	8	DATETIME20.
AGE_AT_ONSET_YEARS	num	8	7.2
GENDER_CODE	char	10	$10.
PRIM_RR_COUNTRY_CODE	char	10	$10.
CONSUMER_REPORT_FLAG	char	1	$1.
REPORT_ID	num	8	11.
DRSN	num	8	4.
AESN	num	8	
DRUG_ID	num	8	11.
DRUG_TYPE_CODE	Char	10	$10.

Table 1: Data Source Metadata

Next table to look at is the report definition metatable. This is already a specially taylored table designed and created manually by the programmer or one of his programs. In the given case study this report definition is used to read all runtime required environment into macro variables using a DATA _NULL_ step with the SYMPUT call routine. Since this table controls relevant aspects of the programs behavior, it will be stored in a place suitable for program control, be it an EXCEL worksheet, an ORACLE table or a textfile accessed with a DATA STEP VIEW.

MVAR	MVAL
hme	&HME.
src	p403
tbl	QTT024646
rpt	PRR
use	50000
fmt	rtf
hdr	OUTSIDE
rtp	LL
fct	PRR_2_5
rst	RC_
ori	LANDSCAPE
psz	A3

Table 2: Report Definition Metadata

Whatever the names and values may mean or be used for I want to tear the readers' attention to the macro variable FCT bearing the value "PRR_2_5". This value tells the called program that it shall calculate a PRR statistic for all combinations of two variables controlled (stratified) by additional five variables.

The most obvious way to pass this information would be to specify the names of these variables as additional macro variables. To retain the programs flexibility this would result in a parameter structure of increasing and difficult to handle complexity.

REPORT SPECIFIC DATA STRUCTURE

So, why not use information from the metatable already presented before by referring to it? If the parameter value "PRR_2_5" is not only used to trigger a specific function but could also be exploitet as reference to specific variables, then an explicit reference to particular variable names could become obsolete.

Exactly this is used when the values of "2" and "5" are not only read as number of variables (cardinalities) but also as the position of variables (ordinality) to be used to feed the function as arguments. With respect to this it is the values of "DRUG" and "PT_CUR" to form the base of a series of 2 x 2 frequency tables for

which PRR values are caluculated. In our case study the 1st variable is used to define the count granularity for the frequency tables, therefore variables number “2” and “3” are the constituents for the frequency tables. According to the 3rd parameter fragment “5” the five variables found at positions “4” to “8” are used for stratification (control) of the coefficients calculated.

Before everything is written to an output destination the program creates a 1st proposal of how the values calculated shall be organized as paper output, hence the report structure. This again is an arbitrarily structured metatable stored somewhere which is completely left to the programmers ideas and concepts.

In our case study the program offers to organize the output in a three-dimensional “space” built from PAG, COL and OBS. There’s also an estimate given for the most appropriate format of the output values along with the type and length.

FLD	LBL	TYP	FMT	DIM	PAG	COL	OBS	LTH	PREFIX	IN_FIX	SUFFIX	OTP
DRUG	DRUG	CHAR	78	c	01	01	01	78				NULL
PT_CUR	PT_CUR	CHAR	25	c	01	02	01	25				NULL
A	CELL A	NUM	1.	c	01	03	01	1				NULL
B	CELL B	NUM	1.	c	01	04	01	1				NULL
C	CELL C	NUM	3.	c	01	05	01	3				NULL
D	CELL D	NUM	3.	c	01	06	01	3				NULL
CMH_CRUDE	PRR CRUDE	NUM	12.8	c	01	07	01	12				NULL
CMHLCRUDE	PRR CRUDE LOWER CL	NUM	11.8	c	01	08	01	11				NULL
CMHUCRUDE	PRR CRUDE UPPER CL	NUM	13.8	c	01	09	01	13				NULL
CMH_STRAT	PRR STRATIFIED	NUM	11.8	c	01	10	01	11				NULL
CMHLSTRAT	PRR STRATIFIED LOWER CL	NUM	11.8	c	01	11	01	11				NULL
CMHUSTRAT	PRR STRATIFIED UPPER CL	NUM	16.8	c	01	12	01	16				NULL

Table 3: Report Structure Metadata

Of course all these values from the metatable are left to be changed by the user when kept in an appropriate format as already mentioned before. The program then will make use of the applied changes and write an output table to the destination given.

But this is another paper…

CONCLUSION

The small case study along with the given derivation of parameter passing as “macro communication” shows another cornerstone of flexible and easy-to-maintain software engineering techniques using procedures and technology provided by the SAS System.

Unlike other technologies the multi-level-processing of code performed by the SAS Macro Facility makes it easy to jump-start with implementing software concepts that are incredibly honored by the functional return from working hours invested.

RECOMMENDED READING

Canopus in Argos: Archives III, Doris Lessing, London (1981)

CONTACT INFORMATION

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

Author Name Wolf Dieter Batz
Company Phenix-MTK GmbH
Address Wiesengrund 8
City / Postcode D-69234 Dielheim
Work Phone: +491772163609
Fax: +496222770095
Email: batz@phenix-mtk.com
Web: www.phenix-mtk.com