That macro talk

(Paramétrisation des programmes SAS)

By Mathieu Gaouette

Prospective MG

Plan

- Introduction
- Definition of “Parametrization”
- Why must we use this
- What should we use this for
- Examples
- Going too far
- Other considerations

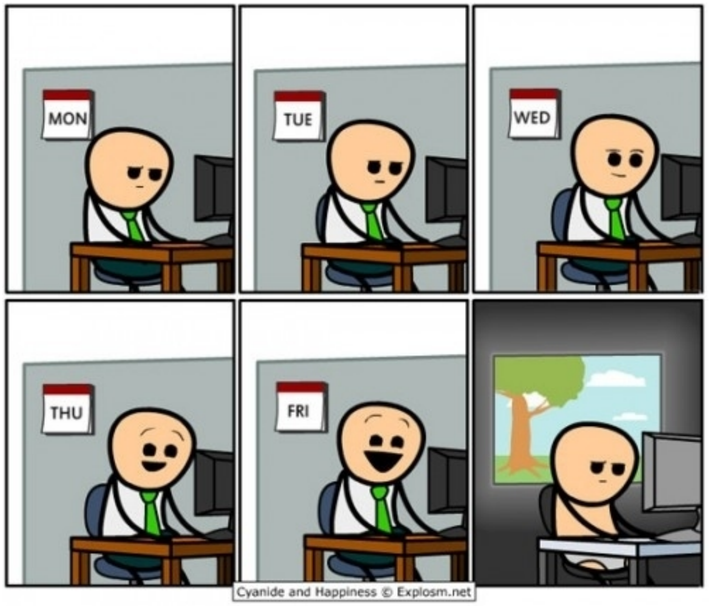
Introduction

Definition

- Lets define the SAS code parametrization as the process of extracting the variable component of a SAS code in order to obtain a static program.
- We generally group the extracted components in a common area to ease the housekeeping of the application.

Why must we do that?

- To prepare an application for production deployment.
- To avoid errors due to manual changes in the code.
- Because it is the right thing to do!



What should we use if for ?

The easiest first:

- User ids and/or password for connections.
- File names of input/output.
  (text/csv, excel, ...)

Then, something a little more complex:

- Dates.

Finally, what we often oversee:

- Product codes and similar reference codes.
  (postal codes, age groups, ...)

User ids and passwords

- Security groups will often require users to take out passwords from their codes. In some organization, this also extends to user ids.
- The pwencode procedure allows SAS users to encode string of data (such as a password).
- The encoding is not actual encryption therefore encoded data should be stored in a file located in a private area to increase security.
- Besides security benefits, such process allows easy sharing of SAS codes between people without sharing credentials.

Example 1: connection to a database

```
proc sql;
   connect to oracle(user=jimgoodnight
                                    password="sasrulez!"
                                    path=SASBD);
   Create table ActiveCustomers as
   select * from connection to oracle(
   Select customerid
   from custbase
   where status = 'ACTIVE'
);
quit;
```

Before

Example 1: connection to a database

```
/* Code header */
%let oracle_userid = jimgoodnight ;
%let oracle_path = SASBD ;
%decode_procedure(access=oracle,macroname=oracle_passwd) ;
...

proc sql;
    connect to oracle(user=&oracle_userid.
                            password="&oracle_passwd."
                            path=&oracle_path.);
    create table ActiveCustomers as
    select * from connection to oracle(
    select customerid
    from custbase
    where status = 'ACTIVE'
);
quit;
```

The decode_procedure function and another one to perform the encoding can easily be created using the pwencode and pwdecode procedures

After

Example 1: connection to a database

- If you are not fond of encoding, there are alternatives that still allow you to remove that sensible information from your code.
- It is possible to integrate a mechanism that prompts the user to input their password within SAS programs.
  - For SAS/PC users, the %window function is perfectly suited for that purpose.
  - For Enterprise Guide users, prompts can be used.

Filenames of inputs/outputs

- Reference files in csv format and excel/pdf reports generated by programs are only a few examples of such files.
- They might not flood your codes but you will find that they are present in almost all of them.
- Using macros for file names amongst other thing prevents you from using the wrong file name for input/output by mistake.

Example 2: Reading a flatfile

```
data postal_code_ref;
  infile 'D:\Canada\200811\cpstl.txt' ;
  input         @001 pstlcd $6.
                @007 regioncd $2. ;
run;
```

Before

Example 2: Reading a flatfile

```
%let src_postal_code = D:\Canada\200811\cpstl.txt ;
data postal_code_ref;
   infile "&src_postal_code." ;
   input @001 pstlcd $6.
                      @007 regioncd $2. ;
run;
```

After

Example 2: Reading a flatfile

```
%let period = 200811 ;
%let src_cd_postal = D:\Canada\&period.\cpstl.txt ;
%let src_cd_postal_inp = @001 pstlcd $6. @007 regioncd $2.;
data postal_code_ref;
   infile "&src_cd_postal." ;
   input &src_cd_postal_inp. ;
run;
```

Or even!

The dreaded dates!

- In most recurring SAS programs, different tasks are based on several dates in different formats.
- Usually, all different dates within a program can be derived from a single date (such as the last day of the month, same date last week).
- The best approach in these situations is to create a single macro variable which contains that key date and then add a few step to dynamically create the derived dates. Doing so only ask of the users to change a single date parameter before running the code.

Example 3: Merging tables with dates

```
data hist_prods_200702;
  merge mart1.cust200702t(in=a keep=custid produit)
              clients_ref_2007(in=b keep=custid dateref
                    where=(dateref='28FEB2007'd)) ;
  by custid;
  if a and b;
run;
```

Before

Prospective MG

Example 3: Merging tables with dates

```
%let period = 200702 ;
...
data _null_;
    tmpdt = input("&period.01",yymmdd8.) ;
    tmpdt = intnx('month',tmpdt,0,'end') ;
    call symput('dtref',compress("'"||put(tmpdt,date9.)||"'d")) ;
run;
data hist_prods_&period.;
    merge mart1.cust&period.t(in=a keep=custid produit)
                               clients_ref_%substr(&period.,1,4)(in=b
                               keep=custid dateref where=(dateref=&dtref.)) ;
    by custid;
    if a and b;
run;
```

After

Example 4: Using dates in a SQL query

```
proc sql;
   connect to oracle(user=&oracle_userid.
                                password="&oracle_passwd."
                                path=&oracle_path.);
   Create table te1_200702 as
   select * from connection to oracle(
   select count(*)
   from sales_customers salescust
   where salescust.periode =
            to_date('20070228','YYYYMMDD'));
quit;
```

Before

Example 4: Using dates in a SQL query

```
%let period = 200702 ;
data _null_;
      tmpdt = input("&period.01",yymmdd8.) ;
      tmpdt = intnx('month',tmpdt,0,'end') ;
      call symput('dtref',compress("'"||put(tmpdt,yymmddn8.)||"'")) ;
run;
proc sql;
      connect to oracle(user=&oracle_userid
                  password=“&oracle_passwd.“
                  path=&oracle_path.);
      create table te1_&period. as
      select * from connection to oracle(
      select count(*)
      from sales_customers salescust
      where salescust.periode =
                  to_date(&dtref.,'YYYYMMDD'));
quit;
```

But does it work?

maybe

After

Example 4: Using dates in a SQL query

```
%let period = 200702 ;
data _null_;
    tmpdt = input("&period.01",yymmdd8.) ;
    tmpdt = intnx('month',tmpdt,0,'end') ;
    call symput('dtref',compress("'"||put(tmpdt,yymmdd10.)||"'",'-')) ;
run;
%macro sqlpt;
proc sql;
    connect to oracle(auth info);
    create table te1_&period. as
    select * from connection to oracle(
    select count(*)
    from sales_customers salescust
    where salescust.periode =
                to_date(&dtref.,'YYYYMMDD'));
quit;
%mend ;
%sqlpt ;
```

But does it work?

After

yes

Example 5: Using multiple dates

```
data hist_produits_200708;
   merge mart1.cust200708t(in=a keep=custid produit)
               mart1.cust200707t(keep=custid produit
                           rename=(produit=produit_1))
               mart1.cust200706t(keep=custid produit)
                           rename=(produit=produit_2));
   by custid;
   if a;
run;
```

Before

Example 5: Using multiple dates

- To do this, we would need a tool that given a reference period (date in YYYYMM format) and number would:
  - Add the number provided in month to the reference period.
  - Return the resulting period.
  - The resulting period has be valid (ex: 201612 + 1 should return 201701 and note 201613)

Example 5: Using multiple dates

- Conceptually, what we want is:

```
%macro dateop(do_period,do_number);
intnx(month,&do_period.,&do_number,E)
%mend;
```

We have 2 problems to address though to make it work:

1. intnx is a datastep function
2. &do_period is not a date and intnx expects a date (The day is missing)

Example 5: Using multiple dates

- The “working” macro (version 2.0)

```
%macro dateop(do_period,do_number) ;
%sysfunc(
    intnx(month,
            %sysfunc(mdy(%substr(&do_period.,5,2),1,%substr(&do_period.,1,4))),
            &do_number.,E),
    yymmn6.) ;
%mend ;
```

Example 5: Using multiple dates

- And now the version 1.0 ... for your entertainment

```
%macro dateop(do_period,do_number);
  %if %eval(&do_number.<1 and &do_number.>-1)=1 %then &do_period. ;
  %else %if %eval(&do_number.>0)=1 %then %do;
    %if %substr(&do_period.,5,2)=12 %then
      %dateop(%eval(&do_period.+89),%eval(&do_number.-1)) ;
    %else %dateop(%eval(&do_period.+1),%eval(&do_number.-1)) ;
  %end;
  %else %if %eval(&do_number.<0)=1 %then %do;
    %if %substr(&do_period.,5,2)=01 %then
      %dateop(%eval(&do_period.-89),%eval(&do_number.+1)) ;
    %else %dateop(%eval(&do_period.-1),%eval(&do_number.+1)) ;
  %end;
%mend;
```

Example 5: Using multiple dates

```
%let period= 200708 ;
data hist_produits_&period.;
    merge
            mart1.cust&period.t(in=a keep=custid produit)
            mart1.cust%dateop(&period.,-1)t
              (keep=custid produit rename=(produit=produit_1))
            mart1.cust%dateop(&period.,-2)t
              (keep=custid produit rename=(produit=produit_2));
    by custid;
    if a;
run;
```

But does it work?

almost

After

Example 5: Using multiple dates

```
%let periode = 200708 ;
%let source1 = mart1.cust&period.t ;
%let source2 = mart1.cust%dateop(&period.,-1)t ;
%let source3 = mart1.cust%dateop(&period.,-2)t ;

data hist_produits_&period.;
    merge
            &source1.(in=a keep=custid produit)
            &source2.(keep=custid produit rename=(produit=produit_1))
            &source3.(keep=custid produit rename=(produit=produit_2));
    by custid;
    if a;
run;
```

And now, does it work?

yes

After

Product codes and other reference data

- We often underestimate the odds of these values changing over time.
- Usually, people know how often these values will change over time.
- The problem resides in the ability of the users to predict the actual lifetime of their programs.
- It’s therefore as important to ease the update of these value through macros.

Example 6: Product based report

```
%let period=200807 ;

title "Product A distribution" ;
title2 "commercial customers" ;
proc freq data=mart1.cust&period.t;
   table proda*region / list ;
   where cust_type in (111,112,113) ;
run;
```

Before

Prospective MG

Example 6: Product based report

```
%let period=200807 ;
%let rpt1_prod = A ;
%let rpt1_cust_type = 111,112,113 ; /* commercial */

title "Product &rpt1_prod. distribution" ;
title2 "commercial customers" ;
proc freq data=mart1.cust&period.t;
   table prod&rpt1_prod.*region / list ;
   where cust_type in (&rpt1_cust_type.) ;
run;
```

After

Prospective MG

Example 6: Product based report

```
/* Macro variable definition */
%let period=200807 ;
%let rpt1_prod = A ; /* A=comm, B=resi */
/* Constant variable definition */
%let rpt1_cust_typeA = 111,112,113 ; /* commercial */
%let rpt1_cust_descA = commercial ;
%let rpt1_cust_typeB = 101,102,103,104,105 ; /* residential*/
%let rpt1_cust_descB = residential ;

title "Product &rpt1_prod. distribution" ;
title2 "&&rpt1_cust_desc&rpt1_prod.. customers" ;
proc freq data=mart1.cust&period.t;
    table prod&rpt1_prod.*region / list ;
    where cust_type in (&&rpt1_cust_type&rpt1_prod..) ;
run;
```

Or even!

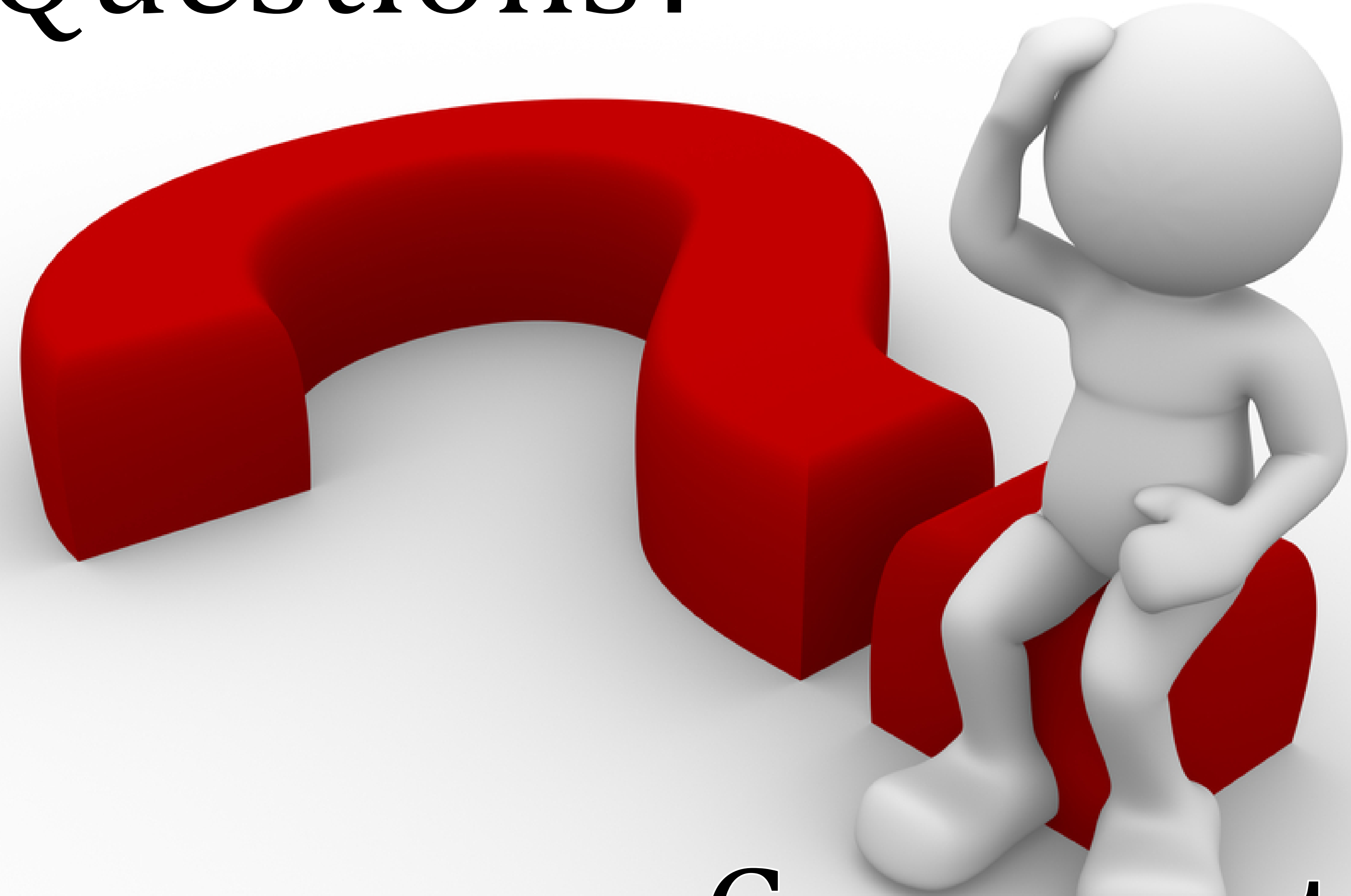
Is it possible to go too far?

- There is no real limit to the use of SAS macros to make code more dynamic and there lies the problem. We need to use common sense.
- There is a cost to the use of macros in a program: its simplicity. Understanding a program that overuses the macros will be more taxing than understanding a program that doesn't use them at all.
- The key to success for a proper use of macros is:
  - Ego under control for macro aficionados.
  - Macro usage level adapted to the requirement of the program (frequency of changes to be expected).
  - Gathering of all parameter type macro variables in a common area (ex: Beginning of the program, config file).
  - Documentation of the parameters and their expected values.

Other considerations

- Reference tables constitute another good way of taking out reference values (such as reference codes) of SAS codes.
- When building a reference table like this, it is strongly suggested to use a notion of history (ex: using a start and end date to track the lifetime of these values)

Questions?



Comments?