La “paramétrisation” de programmes SAS

Mathieu Gaouette
Vice-président Monsug
mathieu.gaouette@gmail.com

Plan

- Introduction
- Définition
- Pourquoi paramétrer
- Quoi paramétrer
- Exemples concrets
- Limites de la paramétrisation
- Autres considérations

Introduction

Définition

- La paramétrisation des codes SAS consiste à extraire les donnés variables de ces derniers pour obtenir des codes statiques.
- On regroupe généralement les donnés extraites en un unique endroit pour faciliter la modification de ceux-ci.

Pourquoi paramétrer?

- Pour préparer un code pour une mise en production.
- Pour éviter les erreurs dû à des changements manuels de valeurs dans le code.
- Parce que c’est la bonne façon de faire!

Que doit-on paramétrer?

Le plus simple d'abord:

- Un code d'usager et/ou mot de passe de connexion.
- Un nom de fichier source ou extrant.

  (texte, excel, …).

Un peu plus complexe:

- Une date.

Puis ce que l'on a tendance à négliger:

- Un code produit ou autre code de référence.

  (codes postaux, classes d'âge, …)

Les codes d'usagers et mots de passe

- Le niveau de sécurité de votre organisation peut exiger le retirait des mots de passes de vos programmes et peut-être même les codes usagers.
- La procédure pwencode permet d'encoder une donnée (telle un mot de passe) dans un fichier.
- L'encodage n'est pas une encryption et il est donc préférable de placer le fichier contenant le mot de passe encodé dans un endroit privé.
- Au dela de la sécurité, une telle paramétrisation permet de partager des programmes sans partager ses accès.

Exemple 1: connexion à une BD

```
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;
```

Avant

Exemple 1: connexion à une BD

```
/* Début du code */
%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;
```

Après

La fonction decode_procedure et une autre pour effectuer l'encodage peuvent-être créées facilement en utilisant la fonctionnalité de la procédure pwencode.

Exemple 1: connexion à une BD

- Si on n'aime pas la solution impliquant l'encodage d'un mot de passe, il y a des alternatives pour qui permettent la paramétrisation.
- On peut ajouter au début de la session SAS une fonctionnalité qui demande à l'utilisateur son mot de passe.
  - Pour les utilisateurs de SAS/PC, la fonction %window est parfaite pour implémenter une telle fonctionnalité de capture de mot de passe.
  - Pour les utilisateurs d'Enterprise Guide, il y a les invites que l'on peut créer.

Les noms de fichiers sources/extrants

- Les tables de références en format csv, les rapports excels ou pdf produits par nos programmes ne sont que quelques exemples de ces fichiers.
- Ils ne sont pas nombreux dans les programmes mais sont utilisés dans pratiquement tous ceux-ci.
- Paramétrer permet entre-autre d'empêcher que l'on écrase une ancienne version d'un tel rapport par un nouveau parce que l'on a oublié de changer le nom de l'intrant.

Exemple 2: Lecture d'un fichier texte

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

Avant

Exemple 2: Lecture d'un fichier texte

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

Après

Exemple 2: Lecture d'un fichier texte

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

Ou encore!

Les dates

- Dans la plupart des programmes récurrents, on doit effectuer des tâches basées sur des dates sous différents formats.
- En général, c'est la même date que l'on utilise à travers le code ou d'autres dates relatives à celle-ci (par exemple, la fin du mois courant, la date courante plus ou moins un mois).
- Une façon de fonctionner dans ces cas est de créer une variable macro qui contient la date puis ajouter quelques étapes pour générer les autres dates à partir de cette dernière. Ainsi on a seulement une date à changer et non plusieurs.

Exemple 3: Lecture d'un fichier texte

```
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;
```

Avant

Exemple 3: Lecture d'un fichier texte

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

Après

Exemple 4: Utilisation de dates dans un sql

```
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;
```

Avant

Exemple 4: Utilisation de dates dans un sql

```
%let periode = 200702 ;
data _null_;
    tmpdt = input("&periode.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_&periode. as
    select * from connection to oracle(
    select count(*)
    from sales_customers salescust
    where salescust.periode =
            to_date(&dtref.,'YYYYMMDD'));
quit;
```

mais est-ce que ça fonctionne?

non

Après

Exemple 4: Utilisation de dates dans un sql

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

mais est-ce que ça fonctionne?

oui

Après

Exemple 5: Utilisation de plusieurs 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;
```

Avant

Exemple 5: Utilisation de plusieurs dates

- Nous avons besoin d'un outil permettant de générer une période (YYYYMM) à partir d'une période fournie et d'un nombre de mois à ajouter (ou soustraire si négatif).
- Les dates générées doivent être valides.

```
%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;
```

Exemple 5: Utilisation de plusieurs dates

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

mais est-ce que ça fonctionne?

presque

Après

Exemple 5: Utilisation de plusieurs dates

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

data hist_produits_&periode.;
      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;
```

mais est-ce que ça fonctionne?

oui

Après

Un code produit ou autre code de référence

- Ce type de donnée est le plus sous-estimé quand à ses chances de changer à travers le temps.
- Généralement, les gens savent estimer correctement la durée de vie utile de la valeur explicite dans leurs programme.
- L'erreur est plutôt au niveau de l'estimé de la vie utile du programme mis en place.
- Il est donc tout aussi important sinon plus de paramétrer ces valeurs.

Exemple 6: Rapport sur un produit

```
%let periode=200807 ;

title "Distribution du produit A";
title2 "pour les clients commerciaux" ;
proc freq data=mart1.cust&periode.t;
    table proda*region / list ;
    where cust_type in (111,112,113) ;
run;
```

Avant

Exemple 6: Rapport sur un produit

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

title "Distribution du produit &rpt1_prod." ;
title2 "pour les clients commerciaux" ;
proc freq data=mart1.cust&periode.t;
    table prod&rpt1_prod.*region / list ;
    where cust_type in (&rpt1_cust_type.) ;
run;
```

Après

Exemple 6: Rapport sur un produit

```
/* Définition des variables */
%let periode=200807 ;
%let rpt1_prod = A ; /* A=comm, B=part */
/* Définition des constantes */
%let rpt1_cust_typeA = 111,112,113 ; /* commercial */
%let rpt1_cust_descA = commerciaux ;
%let rpt1_cust_typeB = 101,102,103,104,105 ; /* particuliers */
%let rpt1_cust_descB = particuliers ;

title "Distribution du produit &rpt1_prod." ;
title2 "pour les clients &&rpt1_cust_desc&rpt1_prod.." ;
proc freq data=mart1.cust&periode.t;
    table prod&rpt1_prod.*region / list ;
    where cust_type in (&&rpt1_cust_type&rpt1_prod..) ;
run;
```

ou encore!

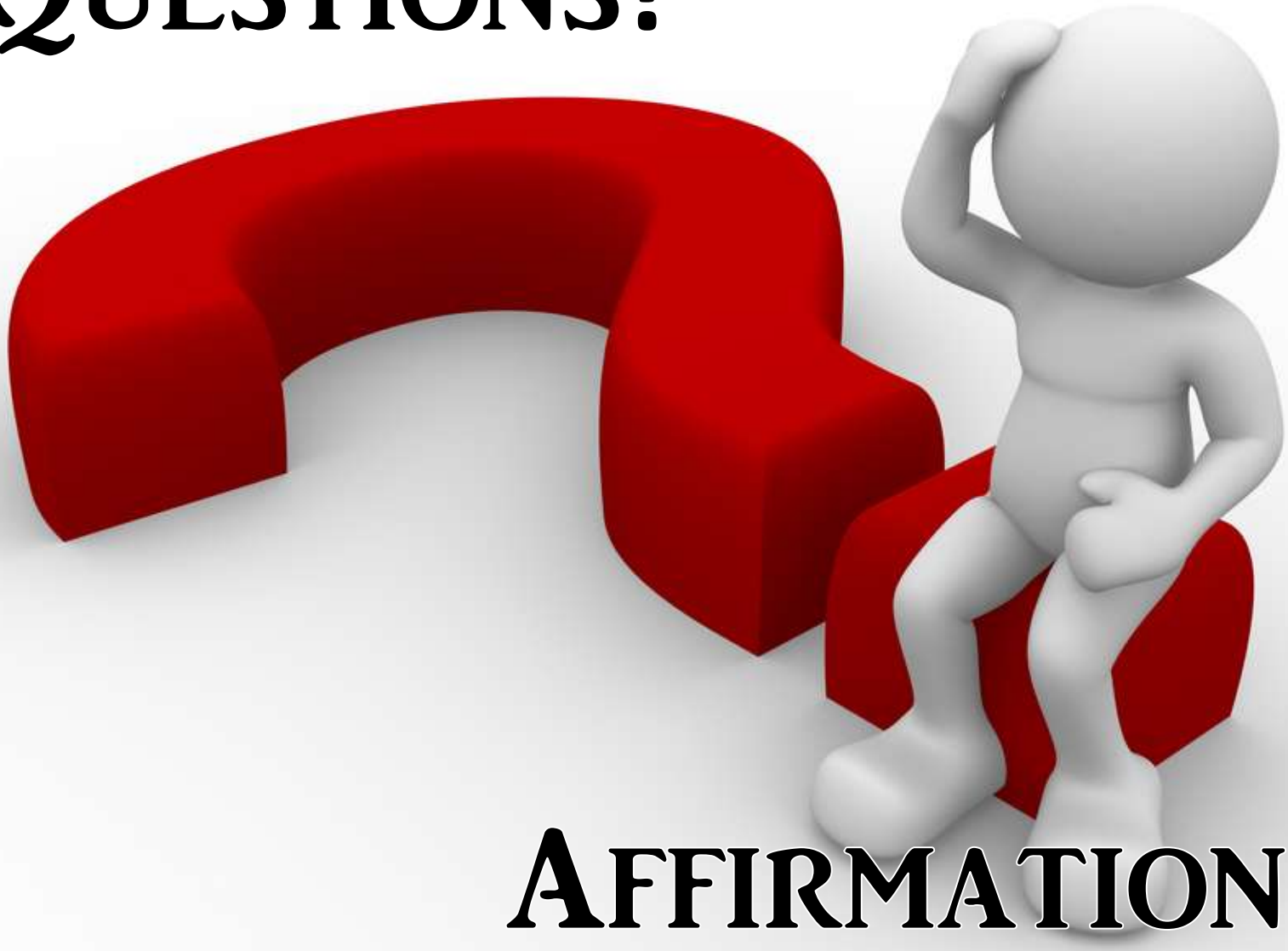
Les limites de la paramétrisation

- Il n'y a pas vraiment de limites et c'est là qu'est le danger. Il faut user de gros bon sens.
- La paramétrisation a un coût, la simplicité du code. Comprendre ce que fait un programme sur-paramétré est plus exigent qu'un programme non-paramétré.
- Les outils du succès pour une bonne paramétrisation sont les suivants:
  - Un égo sous contrôle pour les spécialistes macro.
  - Un niveau de paramétrisation raisonnable et adapté aux besoin du programme (fréquence des changements à prévoir).
  - Centralisation des définitions de macro variables pour les paramètres (ex: au début de l'autoexec, dans un fichier config).
  - Bonne documentation des paramètres.

Autres considérations

- Les tables de références constituent une bonne façon d'isoler les codes de références.
- Lors de la création d'une telle table, il est fortement recommandé d'assurer l'information historique (ex: l'utilisation de champs start et end date, un champ date, …)

Questions?



Affirmations?