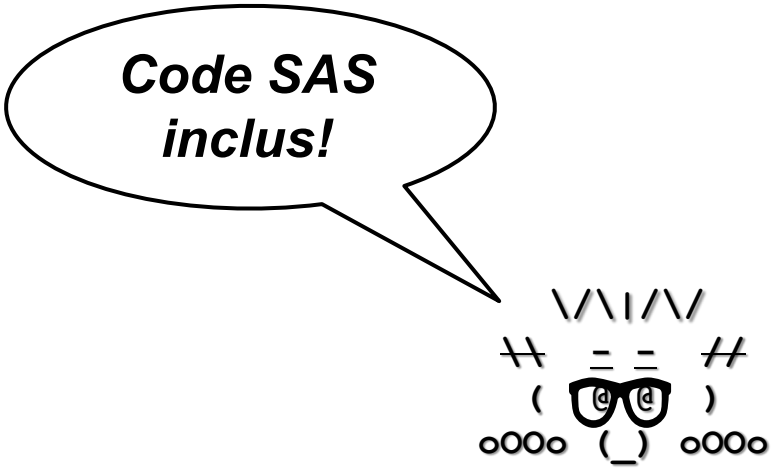
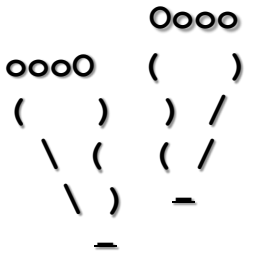
Générer du code SAS® avec CALL EXECUTE

(version longue)

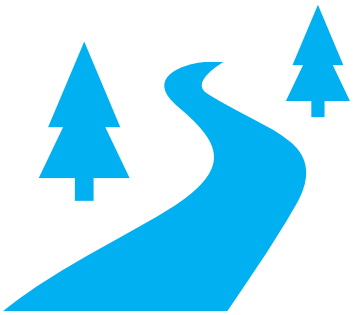


Complément utile ou non au langage %macro?



Alain Voyer
Utilisateur SAS® perplexe
Club des utilisateurs SAS de Québec, 3 février 2022

Parcours proposé



- **Mise en bouche**
- **CALL EXECUTE en bref**
- **Langage macro VS étape DATA**
- **3 cas à l'étude:**
  - **Importation/Exportation de feuilles Excel**
  - **Utilisation de fichier texte auto-documenté**
  - **Tracer le comportement de PROC REPORT (blocs COMPUTE)**
- **Alternatives, compléments, autres pistes**
- **Approche à éviter**
- **Conclusion**
- **Quelques références**
- **Questions, commentaires, suggestions, contact**

Mise en bouche

Au fil du temps, le langage SAS s'est **enrichi** de mécanismes divers comme le langage macro, de nouvelles fonctions, de nouvelles routines, de nouvelles procédures.

De nouvelles composantes se sont également ajoutées le tout dans le but de **simplifier** l'accès aux données et la production de résultats.

Il n'en demeure pas moins que parmi toutes les possibilités offertes, certaines sont demeurées un peu plus dans **l'ombre** que d'autres.

Ainsi en est-il de la routine **CALL EXECUTE** apparue dans la version 6.07 en avril 1991.

CALL EXECUTE est à prime abord une routine de l'étape DATA qui interagit avec le dispositif macro comme le font par exemple le **CALL SYMPUTX** et la fonction **RESOLVE**.

Peut-t'on en tirer des **bénéfices** dans nos traitements?

C'est à **vous** de découvrir la réponse à cette question.

CALL EXECUTE en bref

Exemple à partir de la bibliothèque SASHELP:

```
* Sans CALL EXECUTE il faudrait coder ceci: ;

proc print data=sashelp.class;
run;

proc print data=sashelp.cars;
run;

proc print data=sashelp.heart;
run;

* Avec CALL EXECUTE ;
* Générer des PROC PRINT pour chacune des tables ;

data _null_ ;
  do table = "class", "cars", "heart";
     call execute("proc print data=sashelp." ||table|| ";");
     call execute("run;");
  end;
run;
```

CALL EXECUTE en bref

Extrait du journal (log) résultant:

Le code SAS généré est exécuté après l'étape DATA.

Les **+** témoignent du code généré comme pour un %include

```
NOTE: CALL EXECUTE generated line.
1          + proc print data=sashelp.class;
2          + run;

NOTE: There were 19 observations read from the data set
        SASHELP.CLASS.
NOTE: PROCEDURE PRINT a utilisé (Durée totale du traitement) :
      real time              0.27 seconds
      cpu time               0.04 seconds

3          + proc print data=sashelp.cars ;
4          + run;

NOTE: There were 428 observations read from the data set
        SASHELP.CARS.
NOTE: PROCEDURE PRINT a utilisé (Durée totale du traitement) :

etc...
```

Alternatives, compléments, autres pistes

Exemple d'exécution conditionnelle d'un énoncé à portée globale avec le %IF %THEN %DO en "open code":

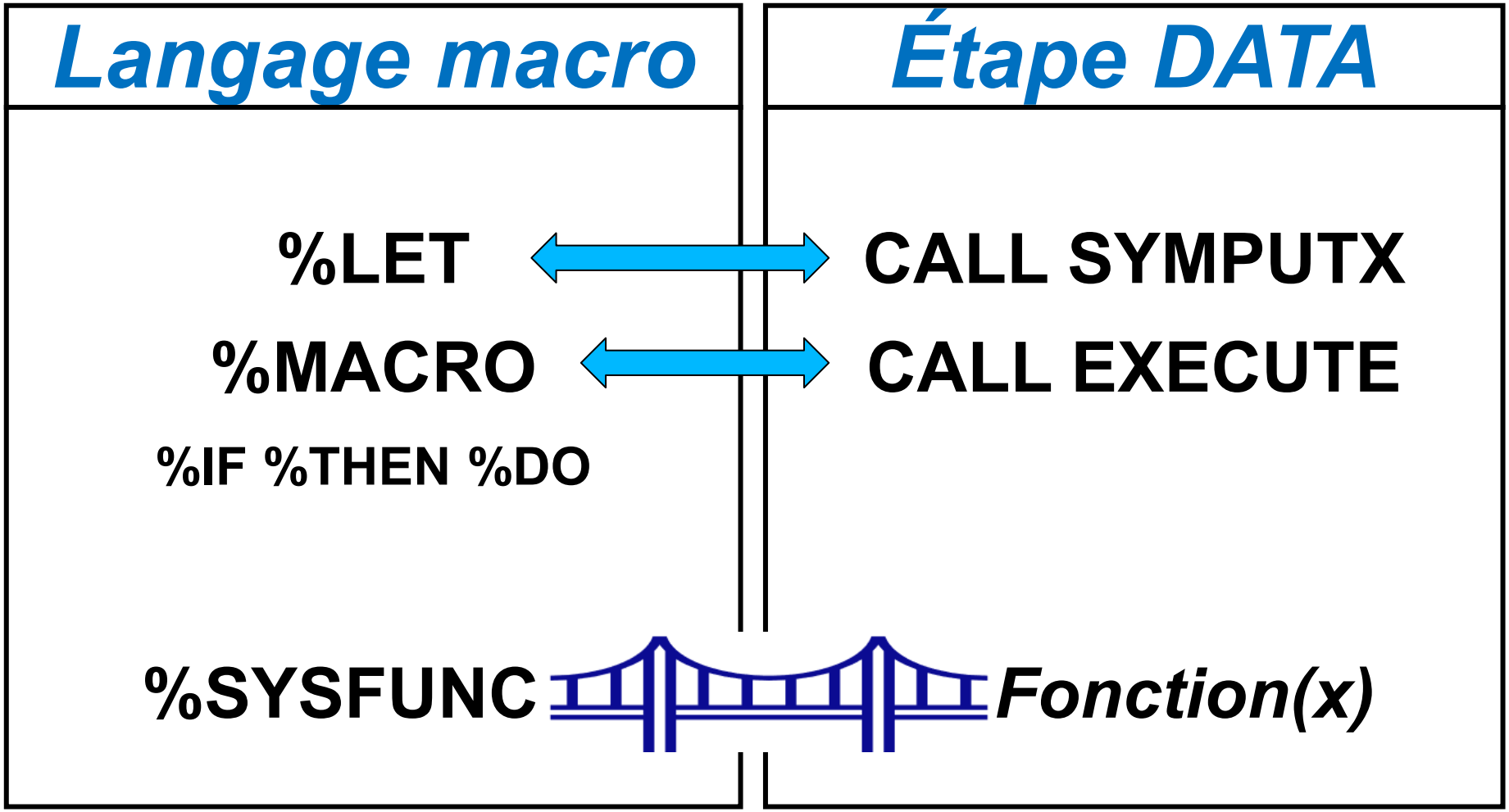
```
* Arrêt du traitement si table vide ;
* Version macro avec %if en open code (SAS 9.4M5) ;

%let dset = TableVide;
%let dsid = %sysfunc(open(&dset));
%let nobs = %sysfunc(attrn(&dsid,NOBS));
%if &nobs=0 %then %do;
        %let rc = %sysfunc(close(&dsid));
        %put NOTE: &=dset  &=dsid  &=nobs  &=rc;
        endsas;
    %end;
```

Dans le journal (log):

```
NOTE: DSET=TableVide  DSID=1  NOBS=0  RC=0
Error: La connexion au serveur a été perdue.  (Enterprise Guide)
Avec le Display Manager, la session se termine immmédiatement
```

Langage **%macro** VS étape **DATA**



Import / Export Excel

Exemple:

```
* Allouer la bibliothèque(classeur)Excel ;
libname MONEXCEL xlsx "C:\temp\SAS\excel\CLASSEUR1.xlsx";

* Générer des étapes DATA pour créer des tables SAS ;
data _null_;
  set sashelp.vmember (where=(libname="MONEXCEL"));
  call execute ( "data work."||memname||";      " );
  call execute ( "  set MONEXCEL."||memname||";" );
  call execute ( "run;                          " );
run;
```

CLASSEUR1.xlsx contient 2 feuilles: **C1F1** et **C1F2**
C1 signifie Classeur1 et Fx signifie Feuille x

	libname	memname	memtype	dbms_memtype	engine	index	path
1	MONEXCEL	C1F1	DATA	TABLE	XLSX	no	C:\temp\sas\excel\CLASSEUR1.xlsx
2	MONEXCEL	C1F2	DATA	TABLE	XLSX	no	C:\temp\sas\excel\CLASSEUR1.xlsx

Import / Export Excel

Extrait du journal (log) résultant:

```
NOTE: CALL EXECUTE generated line
1          + data work.C1F1 ;
2          +    set MONEXCEL.C1F1 ;
3          + run;
```

	A	B
1	C1F1_ColA	C1F1_ColB
2	C1F1_Cell_A2	C1F1_Cell_B2
3	C1F1_Cell_A3	C1F1_Cell_B3

	C1F1_ColA	C1F1_ColB
1	C1F1_Cell_A2	C1F1_Cell_B2
2	C1F1_Cell_A3	C1F1_Cell_B3

```
NOTE: The import data set has 2 observations and 2 variables.
NOTE: There were 2 observations read from the data set MONEXCEL.C1F1.
NOTE: The data set WORK.C1F1 has 2 observations and 2 variables.
NOTE: DATA statement a utilisé (Durée totale du traitement) :
      real time              0.02 seconds
      cpu time               0.03 seconds

4          + data work.C1F2 ;
5          +    set MONEXCEL.C1F2 ;
6          + run;
```

	A	B
1	C1F2_ColA	C1F2_ColB
2	C1F2_Cell_A2	C1F2_Cell_B2
3	C1F2_Cell_A3	C1F2_Cell_B3

	C1F2_ColA	C1F2_ColB
1	C1F2_Cell_A2	C1F2_Cell_B2
2	C1F2_Cell_A3	C1F2_Cell_B3

```
NOTE: The import data set has 2 observations and 2 variables.
NOTE: There were 2 observations read from the data set MONEXCEL.C1F2.
NOTE: The data set WORK.C1F2 has 2 observations and 2 variables.
...
```

Import / Export Excel

Exemple:

```
* Exporter plusieurs tables SAS dans un classeur Excel;

%let mypath=C:\temp\SAS\excel;

data _null_;
  set sashelp.vmember(
      where=(libname="SASHELP" and memname in("CLASS" "CARS" "RETAIL")) );
  call execute( "proc export data="||strip(libname)||"."||strip(memname));
  call execute( '      outfile="&mypath\CombiHelp.xlsx"'                    );
  call execute( "      dbms=xlsx replace;"                                  );
  call execute( "   sheet="||quote(strip(memname))||";"                     );
  call execute( "run;"                                                     );

run;
```

SASHELP.VMEMBER:

Obs.	libname	memname	memtype	nobs	nvar
1	SASHELP	CARS	DATA	428	15
2	SASHELP	CLASS	DATA	19	5
3	SASHELP	RETAIL	DATA	58	5

Import / Export Excel

Extrait du journal (log) résultant:

```
NOTE: CALL EXECUTE generated line.
1          + proc export data=SASHELP.CARS
2          +      outfile="C:\temp\SAS\excel\CombiHelp.xlsx"
3          +      dbms=xlsx replace;
4          +   sheet="CARS";
5          + run;

NOTE: The export data set has 428 observations and 15 variables.
NOTE: "C:\temp\SAS\excel\combihelp.xlsx" file was successfully created.
NOTE: PROCEDURE EXPORT a utilisé (Durée totale du traitement) :
      real time             1.97 seconds
      cpu time              0.03 seconds

6          + proc export data=SASHELP.CLASS
7          +      outfile="C:\temp\SAS\excel\CombiHelp.xlsx"
8          +      dbms=xlsx replace;
9          +   sheet="CLASS";
10         + run;
NOTE: The export data set has 19 observations and 5 variables.
NOTE: "C:\temp\SAS\excel\combihelp.xlsx" file was successfully created.
...etc.
```

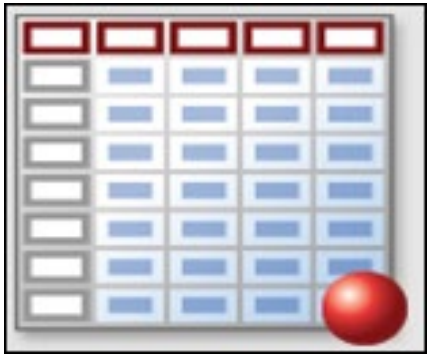
Fichier texte auto-documenté

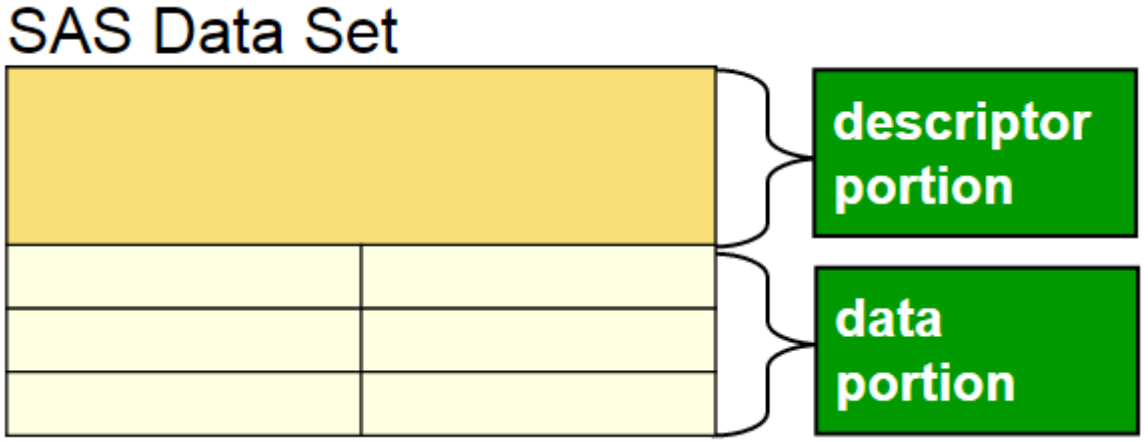
Scénario:

Un fichier texte **contient** des lignes de données qui sont précédées de lignes décrivant ces mêmes données.

En fait un tel fichier texte **ressemble conceptuellement** à une table SAS ou à une table de base de données relationelle. Dans ce genre de structure on retrouve les données elles-mêmes et l'information décrivant ces données.

Comment faire avec une étape DATA et à l'aide de la routine **CALL EXECUTE** pour lire un type de fichier texte contenant la description des données suivie des données elles-mêmes et le transformer en table SAS?





Fichier texte auto-documenté

Exemple de fichier texte:

En colonne 1 le "**V**" signifie description de **V**ariable contenant le nom de la variable et le format de lecture associé. Un commentaire peut optionnellement suivre.

Toujours en colonne 1, le "**D**" identifie une ligne de **D**onnées qui sera lue selon les modalités définies par les lignes "**V**" précédentes.

Les autres lignes sont ignorées – considérées comme des commentaires.

```
----+----1----+----2----+----3----+----4----+----5----+---  _n_
* Spécifications des Variables et des Formats:                1
* Nom........Format..Commentaire                              2
V NMVAR            3.  Variable numérique                     3
V DTVAR    YYMMDD10.  Variable de date                        4
V CHVAR      $CHAR4.  Variable de type caractère              5
* Les données suivent selon le format de lecture ci-haut:    6
D 148,1994-07-15,ABCD    <================================== 7
D 295,1993-12-12,WXYZ                                        8
D 321,2021-06-22,MNOP                                        9
```

Fichier texte auto-documenté

On aimerait retrouver du code SAS ressemblant à:

```
data TabOut;
  infile inVD dsd truncover fistobs=7;
  input @3
        NMVAR : 3.
        DTVAR : YYMMDD10.
        CHVAR : $CHAR4.
        ;
run;
```

```
 * Nom.......Format.
V NMVAR            3.
V DTVAR     YYMMDD10.
V CHVAR       $CHAR4.
```

```
infile inVD .... firstobs=7

D 148,1994-07-15,ABCD <== 7
D 295,1993-12-12,WXYZ
D 321,2021-06-22,MNOP
```

Table WORK.TABOUT

Obs.	NMVAR	DTVAR	CHVAR
1	148	1994-07-15	ABCD
2	295	1993-12-12	WXYZ
3	321	2021-06-22	MNOP

Fichier texte auto-documenté

Code SAS proposé – partie 1:

Génération des énoncés préalables pour créer une étape DATA

```
data _null_ ;
  length VarLu $ 32 FmtLu $ 32;
  * VarLu pour Variable Lue et FmtLu pour Format Lu;

  if _n_=1 then do;
           call execute("data TabOut;                    ");
           call execute("  infile inVD dsd truncover;");
           call execute("  input @3                      ");
           end;

  infile inVD;
  input RecType $upcase1. @;
```

Fichier texte auto-documenté

Code SAS proposé – partie 2:

Génération des éléments pour lire les variables et réajustement du **INFILE** afin de "pointer" sur la première ligne de données "D" avec **FIRSTOBS=**.

```
  if RecType = "V" then
    do; /* On va lire toutes les définitions de variables */
      input VarLu $ FmtLu $;
      call execute("         "|| catx(" ", VarLu, ":", FmtLu));
    end;
  else
  if RecType = "D" then
    do; /* Trouver la première ligne seulement */
      call execute("         ;                                    ");
      call execute("  infile inVD  firstobs="||put(_n_,z3.)||";");
      call execute("run;                                          ");
      stop; /* On arrête aussitôt */
    end;
run ;
```

Remarquer ceci

Fichier texte auto-documenté

Code SAS finalement généré:

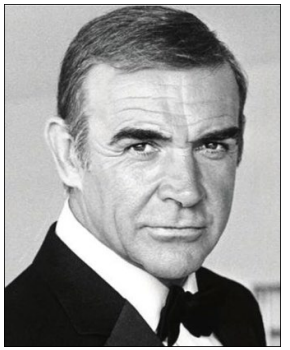
```
NOTE: CALL EXECUTE generated line.
1          + data TabOut;
2          +   infile inVD dsd truncover;
3          +   input @3
4          +         NMVAR : 3.
5          +         DTVAR : YYMMDD10.
6          +         CHVAR : $CHAR4.
7          +         ;
8          +   infile inVD  firstobs=007;
9          + run;
```

Première ligne de données

Rappel – le fichier texte à lire:

```
----+----1----+----2----+----3----+----4----+----5----+---  _n_
* Spécifications des Variables et des Formats:              1
* Nom.....Format..Commentaire                               2
V NMVAR          3.  Variable numérique                     3
V DTVAR   YYMMDD10.  Variable de date                       4
V CHVAR     $CHAR4.  Variable de type caractère             5
* Les données suivent selon le format de lecture ci-haut:   6
D 148,1994-07-15,ABCD    <================================ 7
D 295,1993-12-12,WXYZ                                       8
D 321,2021-06-22,MNOP                                       9
```

firstobs=



Tracer PROC REPORT

Scénario:

La procédure **REPORT** est **riche de fonctionnalités et de possibilités**. Cependant il est parfois **difficile** de bien comprendre ce qui se passe lorsqu'on construit un traitement avec cette procédure notamment avec l'utilisation des blocs **COMPUTE**.

Quand on veux **déboguer** on dispose d'options comme **LIST** et **SHOWALL** mais on n'a pas de mécanisme de traçage comme on peut en retrouver avec l'option **DEBUG** de l'étape DATA. Par contre, l'option **OUT** nous permet d'obtenir une table SAS qu'on peux examiner pour nous aider à mieux comprendre ce qui se passe.

De plus, nous pouvons aussi utiliser la routine **CALL EXECUTE** d'une manière **orignale et inusitée** dans les blocs COMPUTE et ainsi disposer d'un outil d'analyse dans notre quête du résultat recherché.

Cette approche **n'est pas une panacée** et n'est sans doute pas la première approche à utiliser quand on fait la mise au point d'un traitement PROC REPORT mais elle ouvre un sentier qui mérite d'être exploré.

Tracer PROC REPORT

Quelques considérations concernant les blocs COMPUTE:

Type de bloc COMPUTE vs énoncés IF de l'étape DATA:

COMPUTE BEFORE	IF _N_=1
COMPUTE AFTER	IF FIN (SET EOF=FIN)
COMPUTE BEFORE variable	IF FIRST.variable
COMPUTE AFTER variable	IF LAST.variable

Ordre d'exécution des blocs compute en général:

1. COMPUTE item-de-rapport;
2. COMPUTE BEFORE;
3. COMPUTE BEFORE cible;
4. COMPUTE BEFORE _PAGE_;
5. COMPUTE AFTER;
6. COMPUTE AFTER cible;
7. COMPUTE AFTER _PAGE_;

Tracer PROC REPORT

Code SAS proposé – partie 1:

Début de la procédure REPORT – les colonnes et leurs définitions:

```
proc report data=Eleves missing
            /* list showall */
            out=RptEleves;

  columns Genre Nom Age Age=AgeMoy;
  define  Genre  / order;
  define  Nom    / display;
  define  Age    / display;
  define  AgeMoy / mean noprint;

...Suite prochaine diapo
```

Table Eleves

Obs.	Nom	Genre	Age
1	Claude	M	17
2	Elodie	F	13
3	Samuel	X	18
4	Francois	M	12
5	Phong	F	14
6	Tiago	M	15
7	Mila	F	16

Tracer PROC REPORT

Code SAS proposé – partie 2:

Suite et fin de la procédure REPORT – les blocs **COMPUTE**:

```
   compute after;
     call execute('%put NOTE: Compute After;');
     %EcrireValeurs();
   endcomp;

   compute after Genre;
     call execute('%put NOTE: Compute After Genre;');
     %EcrireValeurs();
   endcomp;

   compute Age;
     call execute('%put NOTE: Compute Age;');
     %EcrireValeurs();
   endcomp;
run;
```

Tracer PROC REPORT

Code SAS proposé – partie 3:
Définition d'une macro utilitaire qui va générer l'information de tracage:

```
%macro EcrireValeurs();
  call execute(
       cat('%put INFO: >          '
           , ' _BREAK=' , substr(_break_,1,10)
           , ' Genre='  , Genre
           , ' Nom='    , Nom
           , ' Age='    , Age
           , ' AgeMoy=' , AgeMoy
           , ' ;'
          )   );
%mend EcrireValeurs;
```

Bien que présentée ici, la macro doit bien sûr être placée avant la procédure REPORT.
On verra plus loin que la macro va générer dans le journal (log) des messages comme celui-ci:

```
INFO: >          _BREAK=          Genre=F Nom=Elodie   Age=13 AgeMoy=.
```

Tracer PROC REPORT

Résultat produit:

Rappel des énoncés de la procédure:

```
proc report data=Eleves missing out=RptEleves;
  columns Genre Nom Age Age=AgeMoy;
  define  Genre  / order;
  define  Nom    / display;
  define  Age    / display;
  define  AgeMoy / mean noprint;
  compute after;
    call execute('%put NOTE: Compute After;');
    %EcrireValeurs();
  endcomp;
... etc....
  compute Age;
  call execute('%put NOTE: Compute Age;');
    %EcrireValeurs();
  endcomp;
run;
```

Rapport produit

Genre	Nom	Age
F	Elodie	13
	Mila	16
	Phong	14
M	Claude	17
	Francois	12
	Tiago	15
X	Samuel	18

Tracer PROC REPORT

Extrait du journal SAS – 90 lignes au total :

```
NOTE: Compute Age
INFO: >              _BREAK=            Genre=F Nom=Elodie   Age=13 AgeMoy=.
NOTE: Compute Age
INFO: >              _BREAK=            Genre=  Nom=Mila     Age=16 AgeMoy=.
NOTE: Compute Age
INFO: >              _BREAK=            Genre=  Nom=Phong    Age=14 AgeMoy=.
NOTE: Compute Age
INFO: >              _BREAK=Genre       Genre=F Nom=         Age=. AgeMoy=.
NOTE: Compute After Genre
INFO: >              _BREAK=Genre       Genre=F Nom=         Age=. AgeMoy=14.33
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
INFO: >              _BREAK=            Genre=X Nom=Samuel   Age=18 AgeMoy=.
NOTE: Compute Age
INFO: >              _BREAK=Genre       Genre=X Nom=         Age=. AgeMoy=.
NOTE: Compute After Genre
INFO: >              _BREAK=Genre       Genre=X Nom=         Age=. AgeMoy=18
NOTE: Compute Age
INFO: >              _BREAK=_RBREAK_    Genre=  Nom=         Age=. AgeMoy=.
NOTE: Compute After
INFO: >              _BREAK=_RBREAK_    Genre=  Nom=         Age=. AgeMoy=15
```

Tracer PROC REPORT

Fichier `RptEleves` produit par PROC REPORT

Rappel des énoncés de la procédure :

```
proc report data=Eleves missing out=RptEleves;
  columns Genre Nom Age Age=AgeMoy;
  define  Genre  / order;
  define  Nom    / display;
  define  Age    / display;
  define  AgeMoy / mean noprint;
  compute after;
    call execute('%put NOTE: Compu
    %EcrireValeurs();
  endcomp;
  ............
  endcomp;
  compute Age;
    call execute('%put NOTE: Compu
    %EcrireValeurs();
  endcomp;
run;
```

Fichier créé par la procédure

Obs.	Genre	Nom	Age	AgeMoy	_BREAK_
1	F	Elodie	13	13.0000	
2	F	Mila	16	16.0000	
3	F	Phong	14	14.0000	
4	F		.	14.3333	Genre
5	M	Claude	17	17.0000	
6	M	Francois	12	12.0000	
7	M	Tiago	15	15.0000	
8	M		.	14.6667	Genre
9	X	Samuel	18	18.0000	
10	X		.	18.0000	Genre
11			.	15.0000	_RBREAK_

Alternatives, compléments, autres pistes

Dans les deux premiers exemples on a utilisé le **CALL EXECUTE** pour générer du code SAS. La génération de code est une pratique qu'on retrouve de différentes façons dans beaucoup de traitements SAS.

Le langage **macro** est sans doute la manière la plus courante de créer du code SAS car c'est son rôle fondamental.

La création d'énoncés SAS dans un fichier avec une étape DATA et qu'on récupère avec un **%INCLUDE** est une autre façon de faire.

Plus récemment (SAS9.4M5), l'utilisation en "open code" du **%IF %THEN %ELSE** a ouvert une porte pour faciliter la génération/modification de code sans avoir à encapsuler le tout dans le duo **%MACRO + %MEND**.

La fonction **DOSUBL** apparue à la version 9.3 a poussé un peu plus loin la possibilité d'exécuter du code SAS sans attendre la fin de l'étape DATA comme c'est le cas pour **CALL EXECUTE**.

Alternatives, compléments, autres pistes

Exemple de génération de code avec une combinaison d'étape DATA et du %INCLUDE:

```
* Fichier texte temporaire pour accueillir le code SAS généré;
filename CodeSAS temp;

* Génération du code SAS;
data _null_;
  set TablesChoisies;
  file CodeSAS;
  put "data "  NomTable  ";" ;
  put "  set sashelp." NomTable ";";
  put "run;";
run;

%include CodeSAS /source2;
```

Alternatives, compléments, autres pistes

Résultat abrégé dans le journal (log):

```
NOTE: The file CODESAS is:
      Nom du fichier=C:\Users\...\SAS Temporary...,
NOTE: 9 records were written to the file CODESAS.
NOTE: There were 3 observations read from the data set
       WORK.TABLESCHOISIES.
NOTE: DATA statement a utilisé (Durée totale du traitement) :

62          %include CodeSAS /source2;

NOTE: %INCLUDE (level 1) file CODESAS is file C:\Users...

63         +data class ;
64         +  set sashelp.class ;
65         +run;

NOTE: There were 19 observations read from ... SASHELP.CLASS.
NOTE: The data set WORK.CLASS has 19 observations and 5 variables
... etc.
```

Alternatives, compléments, autres pistes

Clin d'oeil à la fonction DOSUBL (V9.3 en 2011) vs CALL EXECUTE:

```
* Exécution du code en différé;
data _null_;
  putlog "AAA";
  call execute('data _null_;    ');
  call execute('  putlog "BBB";');
  call execute('run;            ');
  putlog "CCC";
run;
```

Résultat dans le journal (log):

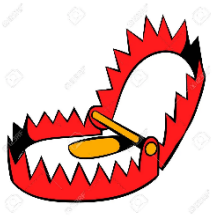
```
AAA
CCC
BBB
```

```
* Exécution du code dynamiquement;
data _null_;
  putlog "AAA";
  rc = dosubl('data _null_;     ' ||
              '  putlog "BBB"; ' ||
              'run;             ' );
  putlog "CCC";
run;
```

Résultat dans le journal (log):

```
AAA
BBB
CCC
```

Approche à éviter



Accès répété à la même source de données:

```
data _null_;
  set SASHELP.CARS;
   by MAKE;
  if first.MAKE then
     do;
       call execute('title1 "Manufacturier '||trim(MAKE)||'";');
       call execute('proc print data=SASHELP.CARS
                              (where=(MAKE="'||trim(MAKE)||'"));');
       call execute('  var MODEL TYPE;');
       call execute('run;');
     end;
run;
```

Approche de génération de code qui peut poser problème pour de **volumineuses tables**. Lecture **répétée** de la même table n'est généralement **pas souhaitable** quelque soit la méthode employée (étape DATA, macro, CALL EXECUTE).

Conclusion

- **CALL EXECUTE**: un outil pour **générer du code SAS**
- Pour compléter son **coffre d'outils** en la matière
- **Pas une panacée** ou un outil universel
- **Simple** d'utilisation
- Accessible pour les **personnes moins familières** avec le langage macro
- **Peut être utile** dans certaines circonstances
- Usage **non trivial** avec la procédure REPORT (résultat pas évident !)

Quelques références

CALL EXECUTE: How and Why
H. Ian Whitlock, Westat Inc.
Proceedings of the Twenty-Second Annual SAS Users Group International Conference, 1997

Batch File Exchange between SAS and Excel - the Magic of Call Execute
Dadong Li, Regeneron Pharmaceutical Inc.
Proceedings of the 2019 SAS Global Forum International Conference, paper 3114

How to conditionally execute SAS global statements
(CALL EXECUTE inside a compute block in PROC REPORT)
Jack Hamilton in Leonid Batkhan's SAS Users Blog, March 15, 2021

MAGIC WITH CALL EXECUTE
Bob Virgile, Robert Virgile Associates, Inc.
Proceedings of the Twenty-Second Annual SAS Users Group International Conference, 1997

Utilisation des commandes macro %IF %THEN %DO…à l'extérieur des routines macro
Patrice Bourdages, Desjardins et Club des utilisateurs SAS de Québec
Conférence du Club des utilisateurs SAS de Québec, 3 juin 2021

Using %IF-%THEN-%ELSE in SAS programs
Chris Hemedinger, SAS Institute Inc.
On The SAS Dummy Blog, July 5, 2018

Quelques références

Generating Code with PROC REPORT
Jack Hamilton, Kaiser Foundation Health Plan
Western Users of SAS Software 2008 Conference Proceedings

The REPORT Procedure: A Primer for the Compute Block
Jane Eslinger, SAS Institute Inc.
Proceedings of the SAS Global Forum 2015, paper 1642

Dictionnaire de données en SAS
Patrice Bourdages, CGI et Club des utilisateurs SAS de Québec
Conférence du Club des utilisateurs SAS de Québec, 1er novembre 2016

A brief understanding of DOSUBL beyond CALL EXECUTE
Ajay Sinha, Novartis, India, M Chanukya Samrat, Novartis, India
PharmaSUG 2020 - Paper QT-183

Get funky with %SYSFUNC
Elena Muriel, Amadeus Software Limited

Base SAS® 9.4 Procedures Guide, Seventh Edition
Cary, NC: SAS Institute Inc.

SAS® 9.4 Macro Language: Reference, Fifth Edition
Cary, NC: SAS Institute Inc.

Merci!

Questions ?
Commentaires ?
Suggestions ?

Générer du code SAS® avec CALL EXECUTE

Contact:

```
1 data _null_;
2   courriel=lowcase(cat(reverse("Alain"),reverse("Voyer"),"@gmail.com"));
3   putlog "NOTE: " courriel=;
4 run;

NOTE: courriel=nialareyov@gmail.com
```

Marques de commerce

SAS et tous les autres noms de produits et services de SAS Institute Inc. sont des marques de commerce de SAS Institute Inc. aux États-Unis et dans les autres pays. ® indique l'inscription aux États-Unis. Les autres noms de produits ou marques sont des marques déposées ou des marques de commerce de leurs compagnies respectives.