**BASUG Q1 2019 Webinar Q&A Responses**

**A Long-Time SAS® Programmer Learns New Tricks,** by Lisa Horwitz, SAS Institute Inc.

Q1: Will savedata include just data sets from the work library, or also other permanent data sets?

A1: Just datasets from the work library. According to Base SAS 9.4 Procedures Guide, "When this procedure is invoked at the end of a SAS session, all of the global statements and macro variables are written to a file. The Work data sets and the macro catalog are written to an auxiliary directory."

Be careful, though: the PRESENV procedure will delete ALL datasets already in the SAVEDATA library before copying the datasets from WORK at the end of your session, even if those datasets were already in the library because they were previously permanently saved to that location. This is to ensure that the datasets copied from WORK are the most current versions of these datasets. I suggest that the library referenced by SAVEDATA is used only as a repository for saved WORK datasets.

Q2: Have you used the PRESENV OPTION and PROC for interacting with, or in lieu of, checkpoint-restart capabilities?

A2: I have not but at least one place where it can be used for something like a checkpoint-restart is in a grid environment. According to the Base SAS 9.4 Procedures Guide, "This functionality is very useful in a grid environment where an Enterprise Guide (EG) session needs to be terminated and started again on another node at a later time."

Q3: What was the command after %include for Presenv? (Sascode? Savedata?)

A3: The command is %include 'saved-code'; where saved-code is the location where the SAS code to restore the environment is stored. In my example, the SAS code was saved in 'c:\presenv\code\store.sas', so I used %include to rerun that code and restore my environment:

```
%include 'c:\presenv\code\store.sas';
```

Q4: Can you clarify what COMPGED is doing? 300 for difference between the 1st two seems very large

A4: The COMPGED function, according to the SAS 9.4 Functions and Call Routines: Reference, is a "generalization of Levenshtein edit distance, which is a measure of dissimilarity between two strings." The documentation shows the exact default cost in units for each type of change needed to advance from the character in the lookup to the character in the compared string. For example, an extra character in the compared string is equal to a cost of 100, so you can see how the total cost can get quite large quite quickly!

Q5: How are the number of steps determined? I would think that the change of 3 letters (Inc vs Org) would result in a difference of 3, but that doesn't seem to be the case.

A5: Please see my answer to Q4, above. The function determines a "cost" based on the differences between the strings, according to a set of criteria.

Q6: Can the Macro Variable viewer and Data Step debugger be used in the non-Enterprise Guide product of SAS?

A6: The data step Debugger has been available for quite some time in Display Manager, and only more recently in Enterprise Guide. As for the Macro Variable Viewer, I don't believe there is an equivalent, but in Display Manager you can submit %put _all_;  to see all current automatic and user defined macro variables and their values.

Q7: Is there a reason to use compged over spedis?

A7: The spedis function is definitely another way to compare strings, and for your purposes, it may provide you with better accuracy than compged. However, according to the SAS 9.4 Functions and Call Routines: Reference,"The SPEDIS function is similar to the COMPLEV and COMPGED functions, but COMPLEV and COMPGED are much faster, especially for long strings." You'll need to test them both out to see which one gives you better results.

And in case you were wondering, the year in question (Ronald Reagan was president, Villanova won the Men's Basketball Championship, etc.) is 1985!

**PROC FCMP: A Powerful  Procedure You Should Be Using,** by Bill McNeill, Andrew Henrick, Mike Whitcher, and Aaron Mays, SAS Institute Inc. Presented by Bill  McNeill

Q8: Can functions be compiled to hide source code when distributed?

A8: The answer is yes. In SAS 9.4 Maintenance release 1 and later there is a FCMP procedure option called HIDE (you will need to scroll down a little to see the option description at this link) that will encode the source code of all the functions created in the procedure step.

I have included below an example using the tax function I used in the presentation. I ran this code at SAS 9.4 Maintenance release 3 (where the STATIC statement has support). I created two versions of the function: one hidden and the other not hidden. I then print out the data set where the functions are stored. You can see in the highlighted cells that the source code for the hidden function is replaced with random characters. I then used both functions to verify that they work. Finally, I called both functions from the FCMP procedure with the TRACE option. Note that the function with the hidden source code does not display any of the source code.

```
1    PROC FCMP OUTLIB=work.paper.basics;

2       FUNCTION priceWithTax_nohide(price);
3          STATIC taxRate;
4          IF (taxRate EQ .)
5            THEN taxRate = (7.25 / 100);
6          taxedPrice = (price * (1 + taxRate));
7          RETURN (taxedPrice);
8          ENDSUB;
9       QUIT;

NOTE: Function priceWithTax_nohide saved to work.paper.basics.
NOTE: PROCEDURE FCMP used (Total process time):
      real time           4.95 seconds
      cpu time            2.51 seconds

10
11   PROC FCMP OUTLIB=work.paper.basics hide;
12      FUNCTION priceWithTax_hide(price);
13         STATIC taxRate;
14         IF (taxRate EQ .)
15           THEN taxRate = (7.25 / 100);
16         taxedPrice = (price * (1 + taxRate));
17         RETURN (taxedPrice);
18         ENDSUB;
```

```
19
20      QUIT;

NOTE: Function priceWithTax_hide saved to work.paper.basics.
NOTE: PROCEDURE FCMP used (Total process time):
      real time           0.12 seconds
      cpu time            0.03 seconds

21      proc print data=work.paper; run;

NOTE: There were 19 observations read from the data set WORK.PAPER.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.11 seconds
      cpu time            0.06 seconds
```



| Obs | _Key_ | Owner | Sequence | Type | Subtype | Name | Continue | NValue | Encoded | Value |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 0 | Prototype | FCmp | basics | 0 | . | . | 00128800008192000643 2 |
| 3 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 1 | Header | Function | | 0 | . | . | 1869924236.3170 |
| 4 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 2 | Statement Source | Executable | FUNCTION | 0 | 65 | . | FUNCTION priceWithTax_nohide (price); |
| 5 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 3 | Statement Source | Declarative | STATIC | 0 | 102 | . | STATIC taxRate; |
| 6 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 4 | Statement Source | Executable | IF | 0 | 2 | . | IF (taxRate EQ .) THEN |
| 7 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 5 | Statement Source | Executable | ASSIGN | 0 | 1 | . | taxRate = (7.25 / 100); |
| 8 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 6 | Statement Source | Executable | ASSIGN | 0 | 1 | . | taxedPrice = (price * (1 + taxRate)); |
| 9 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 7 | Statement Source | Executable | RETURN | 0 | 1 | . | RETURN (taxedPrice); |
| 10 | F.BASICS.PRICEWITHTAX_NOHIDE | CMP | 8 | Statement Source | Executable | ENDSUB | 0 | 14 | . | ENDSUB; |
| 11 | BASICS | CMP | 0 | Header | Package | | 0 | . | . | 1869924241.7711 |
| 12 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 0 | Prototype | FCmp | BASICS | 0 | . | . | 00128800008192000643 2 |

| Obs | _Key_ | Owner | Sequence | Type | Subtype | Name | Continue | NValue | Encoded | Value |
|---|---|---|---|---|---|---|---|---|---|---|
| 13 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 1 | Header | Function | | 0 | . | . | 1869924241.7711 |
| 14 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 2 | Statement Source | Executable | FUNCTION | 0 | 65 | 56 | °K°-<br>†CÓ Ø£B{-£Ï®o‹@<br>¡O”"ô-\!f)Ý<br>Tq*5e=¶ |
| 15 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 3 | Statement Source | Declarative | STATIC | 0 | 102 | 36 | Á?Fg=}üð½ G •Ù<br>Þ¾fiÆ<br>VöÀÍëz |
| 16 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 4 | Statement Source | Executable | IF | 0 | 2 | 44 | Zìê8Áï®©Æ2/®™àuß—<br>˜ý³±q?ì¯ ™™¾ Ig~ |
| 17 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 5 | Statement Source | Executable | ASSIGN | 0 | 1 | 56 | cøÜ©DôÐ œjg---<br>W4Å¨U®<hv!K† }¢Ä<br>·™I„OõHŠ ™™ |
| 18 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 6 | Statement Source | Executable | ASSIGN | 0 | 1 | 64 | þ© Öê™<br>'Nù©T•6®ø¹nl• iÖñ<br>eŽø1a•Áü²<br>Ü%~åÌS'Ü€Â»d©\|±k |
| 19 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 7 | Statement Source | Executable | RETURN | 0 | 1 | 40 | † œjg--- e Š½³±<br>qÐHßþ\³ }M˜â´ |
| 20 | F.BASICS.PRICEWITHTAX_HIDE | CMP | 8 | Statement Source | Executable | ENDSUB | 0 | 14 | 24 | ç---W4<br>—š É4°‹Â° |

```
22
23      option cmplib=work.paper;
24
25      data _null_;
26        hiddenPrice=4.50;
27        hiddenTaxPrice=priceWithTax_hide(hiddenPrice);
28        put hiddenTaxPrice=;
29        unhiddenPrice=6.75;
30        unhiddenTaxPrice=priceWithTax_nohide(unhiddenPrice);
31        put unhiddenTaxPrice=;
32        run;


hiddenTaxPrice=4.82625
unhiddenTaxPrice=7.239375

NOTE: DATA statement used (Total process time):
      real time           0.79 seconds
      cpu time            0.32 seconds

33    proc fcmp inlib=work.paper trace;
34        hiddenPrice=4.50;
35        hiddenTaxPrice=priceWithTax_hide(hiddenPrice);
36        put hiddenTaxPrice=;
37        unhiddenPrice=6.75;
```

```
38        unhiddenTaxPrice=priceWithTax_nohide(unhiddenPrice);
39        put unhiddenTaxPrice=;
40        run;

NOTE: PROCEDURE FCMP used (Total process time):
      real time           0.20 seconds
      cpu time            0.11 seconds


                  The SAS System        15:23 Wednesday, April 3, 2019   1
                                       The FCMP Procedure


--- Program Execution Starting.
    1    1 (35:6)    Executing Stmt            : ASSIGN hiddenPrice =
    1      (35:17)   hiddenPrice = 4.5
    1    2 (36:6)    Executing Stmt            : ASSIGN hiddenTaxPrice =


--- Subroutine priceWithTax_hide Execution Starting.
--- Subroutine priceWithTax_hide Execution Finished.



    1      (36:38)   hiddenTaxPrice = priceWithTax_hide( hiddenPrice=4.5 ) = 4.82625
    1    3 (37:6)    Executing Stmt            : PUT
hiddenTaxPrice=4.82625

    1      (37:6)    PUT (2 items)
    1    4 (38:6)    Executing Stmt            : ASSIGN unhiddenPrice =
    1      (38:19)   unhiddenPrice = 6.75
    1    5 (39:6)    Executing Stmt            : ASSIGN unhiddenTaxPrice =


--- Subroutine priceWithTax_nohide Execution Starting.

    1    1 (0:5)     Executing Stmt            : FUNCTION
    1    2 (6:8)     Executing Stmt            : IF
    1      (0:23)    _temp1 = (taxRate=.) = .
    1    3 (7:15)    Executing Stmt            : ASSIGN taxRate =
    1      (0:31)    taxRate = 7.25 / 100 = 0.0725


    1    4 (8:8)     Executing Stmt            : ASSIGN taxedPrice =
    1      (0:33)    _temp1 = 1 + (taxRate=0.0725) = 1.0725
    1      (0:28)    taxedPrice = (price=6.75) * (_temp1=1.0725) = 7.239375
    1    5 (9:8)     Executing Stmt            : RETURN _priceWithTax_nohide_ =
    1      (0:16)    _priceWithTax_nohide_ = (taxedPrice=7.239375) = 7.239375
    1    6 (10:8)    Executing Stmt            : ENDSUB

--- Subroutine priceWithTax_nohide Execution Finished.



    1      (39:42)   unhiddenTaxPrice = priceWithTax_nohide( unhiddenPrice=6.75 ) = 7.239375
    1    6 (40:6)    Executing Stmt            : PUT
unhiddenTaxPrice=7.239375

    1      (40:6)    PUT (2 items)
--- Program Execution Finished.
```