

APPEND, EXECUTE and MACRO

**Jim Groeneveld,
OCS Biometric Support,
's Hertogenbosch, Netherlands.**

PhUSE 2010 – CC05

AGENDA / CONTENTS

1. Drawbacks of PROC APPEND
2. Consequences of drawbacks
3. Alternatives for PROC APPEND
4. SAS macro %_Append_
5. Application of %_Append_
6. Drawbacks of CALL EXECUTE
7. %_Append_ and CALL EXECUTE
8. Macro calls from CALL EXECUTE

What does PROC APPEND?

- 1. Concatenate two datasets that have the same structure: variables, their types, character lengths**
- 2. With the FORCE option datasets may have different structures, PDV of 1st dataset retained, consequences:**
 - a) extraneous lengths in 2nd dataset discarded
 - b) additional variables in 2nd dataset discarded
 - c) variable attributes in 2nd dataset discarded, only keeping those of the 1st dataset

Three example datasets to be concatenated

```
DATA One;  
  LENGTH One $6;  
  One = 'abcdef';  
  Format One $6.;  
  INFORMAT One $CHAR6.;  
  LABEL One='This is One';  
RUN;
```

```
DATA Two;  
  LENGTH One Two $8;  
  One = '12345678';  
  Two = '12345678';  
RUN;
```

```
DATA Three;  
  LENGTH One $4 Two $12;  
  Two = 'ABCDEFGHijkl';  
  Three = 3;  
  FORMAT One 10.;  
  LABEL One='New label One';  
  FORMAT Two $CHAR12.;  
  INFORMAT Two $12.;  
  LABEL Three='This is Three';  
RUN;
```

Result of (twice) PROC APPEND

```
DATA Appended;
  SET One;
RUN;
PROC APPEND BASE=Appended
DATA=Two FORCE;
RUN;
PROC APPEND BASE=Appended
DATA=Three FORCE;
RUN;
```

```
TITLE "twice PROC APPEND";
PROC PRINT DATA=Appended;
RUN;
```

The PROC PRINT result is:

Obs	One
1	abcdef
2	123456
3	<i>{missing value}</i>

The PROC CONTENTS result is:

Obs	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD
1	One	2	6	1	This is One	\$	6	0	\$CHAR	6	0

Consequences of drawbacks:

- 1. Not generally useful unless datasets have identical structures**
- 2. Yet often used by programmers failing to know about these restrictions**
- 3. This may lead to unintended, incorrect results (lacking variables, truncated character values)**

Alternatives for PROC APPEND

1. SAS data step, PDV retention:

- a. **new** variables in second dataset **retained**, but:
- b. variable **attributes** from **first** dataset **where occurring** retained, so **extended** character **lengths** in **second** dataset may be **truncated**

```
DATA Appended;
  SET One Two Three;
RUN;
```

The PROC CONTENTS result is:

Obs	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD
1	One	2	6	1	This is One	\$	6	0	\$CHAR	6	0
2	Three	1	8	3	This is Three		0	0		0	0
3	Two	2	8	2		\$CHAR	12	0	\$	12	0

The PROC PRINT result is:

Obs	One	Two	Three
1	abcdef		.
2	123456	12345678	.
3		ABCDEFGH	3

Alternatives for PROC APPEND

2. PROC SQL (OUTER UNION CORR):

- a. **new** variables in second dataset **retained, OK;**
- b. variable **labels** from **first** dataset **where occurring** retained, **OK;**
- c. Character variable lengths **maximised: OK!**
- d. (in)formats retained from **only first** dataset, other (longer) formats in second dataset are discarded, **NOK;**
- e. **new** variables in second dataset thus have **no** associated (in) **formats, NOK.**

Alternatives for PROC APPEND

2. PROC SQL example:

```
PROC SQL;
  CREATE TABLE OneTwo AS
    SELECT * FROM One
  OUTER UNION CORR
    SELECT * From Two;
  CREATE TABLE OneTwo3 AS
    SELECT * FROM OneTwo
  OUTER UNION CORR
    SELECT * FROM Three;
QUIT;
```

```
PROC PRINT DATA=OneTwo3;
RUN;
```

The PROC PRINT result is:

Obs	One	Two	Three
1	abcdef		.
2	123456	12345678	.
3		ABCDEFGHJKLM	3

The PROC CONTENTS result is:

Obs	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD
1	One	2	8	1	This is One	\$	6	0	\$CHAR	6	0
2	Three	1	8	3	This is Three		0	0		0	0
3	Two	2	12	2			0	0		0	0

SAS macro %_Append_

1. Variables from both datasets retained;
2. Character variable lengths maximised;
3. Character (in)formats from maximum occurring length (not from existing (in)formats, those are discarded);
4. Numerical (in)formats from:
 1. firstly occurring **named** (in)format or;
 2. maximum **unnamed** width and decimals.
5. Variable labels from first occurrence.

SAS macro %_Append_

The PROC CONTENTS result is:

Obs	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD
1	ONE	2	8	1	This is One	\$	8	0	\$CHAR	8	0
2	THREE	1	8	3	This is Three		0	0		0	0
3	TWO	2	12	2		\$	12	0	\$	12	0

Remarks

Character lengths determined from stored (reserved) length, not from maximum occupied length

Neither one of the discussed methods appends datasets with same named variables of different types

Application of macro %_Append_

1. Usual code to append datasets:

```
%_Append_ (Base=Base_dataset, Data=Data_dataset)
```

2. From CALL EXECUTE in a data step:

```
DATA Reference;  
  INPUT Dataset $16.;  
  CARDS;  
dataset1  
dataset2  
dataset3  
;  
RUN;
```

```
DATA _NULL_;  
  SET Reference;  
  CALL EXECUTE ('%_Append_  
    (Base = Appended,  
    Data = ' ||  
    TRIM(Dataset) || ');'  
  );  
RUN;
```

appending 3 datasets to Appended

Macro %_Append_ from CALL EXECUTE

1. Macro call %_Append_ within single quotes to delay its resolution;
2. Yet, once such a macro runs all macro code is executed immediately while the embedded SAS code still is delayed. This is no problem unless:
3. Some macro code was intended to be dependent on SAS code results (after RUN; or QUIT;), then erroneous! This is a drawback of CALL EXECUTE.

Example macro:

```
%MACRO TargetMacro;  
  DATA _NULL_;  
    PUT '==1==';  
  RUN;  
  %PUT ==2==;  
  DATA _NULL_;  
    PUT '==3==';  
  RUN;  
  %PUT ==4==;  
%MEND TargetMacro;
```

printing numbers 1 to 4 from
data step and macro code.

Calling macro:

```
%PUT Directly called;  
%TargetMacro
```

prints in order 1 - 2 - 3 - 4.

```
%PUT Via CALL EXECUTE;  
DATA _NULL_;  
  CALL EXECUTE  
    ('%TargetMacro');  
RUN;
```

prints in order 2 - 4 - 1 - 3.

Rewritten macro %_Append_

- 1. All conditional code is SAS code;**
- 2. All macro code meant to be processed in the beginning, controlling SAS code, not the other way around;**
- 3. SAS code may generate macro values using CALL SYMPUT, that can be processed conditionally.**

Calling macros from CALL EXECUTE

- 1. avoiding conditional macro code, like rewritten macro `%_Append_`;**
- 2. without caring for conditional macro code, delaying all macro code until after the data step ends, like the SAS code, using the `%NRSTR` macro function.**

Delaying macros from CALL EXECUTE

General way to delay macro code such that the order of numbers output in the example is 1-2-3-4:

```
%PUT Via CALL EXECUTE and %NRSTR(%%)NRSTR;  
DATA _NULL_; * Delaying resolution as well;  
    CALL EXECUTE(' %NRSTR(%TargetMacro) ');  
RUN;
```

Yet the whole call within single quotes!
%_Append_ needed no revision after all.

QUESTIONS & ANSWERS

SASquestions@ocs-consulting.com

Jim.Groeneveld@ocs-biometricsupport.com

http://jim.groeneveld.eu.tf/_append_