Paper CT05

Using the FCMP Procedure to Streamline the Production of Display Statistics in Output Tables

Paul Wicks, MSD, London, UK
Anasooya Anoop, MSD, London, UK
Anna Socha, MSD, London UK

ABSTRACT

The SAS Function Compiler (FCMP) enables SAS programmers to create and store frequently used, complex, or repetitive code into reusable and customised SAS functions and subroutines. Because of its versatility, it is useful for beginners as well as more experienced programmers. The PROC FCMP routines can be reused in any program that has access to the data set where the function or routine is stored. Although PROC FCMP has many options, this paper focuses on how to create new functions to streamline the production of display statistics in output tables. We will cover the basics on how to create, store, delete, retrieve, and use these functions.

INTRODUCTION

Programming in SAS often involves repetitive coding across reporting packages. For instance, within the code of different output tables in the same package, or across different packages. The use of user defined SAS macros is a recognized way to repeat coding within and between packages. In addition, a simple way to introduce efficiency into a package is to convert lines of repetitive code into a function, and, using the SAS Function Compiler (FCMP) to save this function into a permanent, accessible data set. Once SAS saves this function into a data set, a programmer can link to the data set, and reference the functions saved in it to reduce the amount of coding lines. This also ensures a degree of standardization for packages with many outputs.

For instance, if the confidence interval is displayed in many tables of a package, PROC FCMP can be used to compile the code for concatenating the components of the confidence interval (the type of brackets, the display of the lower and upper limit and the comma, dash or semicolon between the limits). This allows streamlining of the code by referring only to the compiled function name in the data step or PROC SQL that calls the compiled function CI to create the confidence interval. In addition, each output that contains confidence intervals created using the compiled function CI will have the same style, creating a level of standardization across outputs.

CREATING A FUNCTION

EXAMPLE: THE CONFIDENCE INTERVAL

The confidence interval is used as an interval estimate for many statistics, and the PROC FCMP code can be used to standardize presentation of the confidence interval values.

```
proc fcmp outlib = work.functions.statistics;
    function ci(lcl, ucl) $;
    length CONCATP $100;
    CONCATP = cats( "[", put(lcl,8.2),'; ', put(ucl,8.2), "]");
    return(CONCATP);
    endsub;
quit;
```

The OUTLIB statement shows where and how the function CI is saved. The code for the function is compiled and stored as a SAS data set ***work.functions***, where ***work*** is the library name and ***functions*** is the data set name . Within this data set, the function CI is organized into a data set 'package' called ***statistics.***

In this example, the function CI is created using input variables LCL and UCL, corresponding to the lower and upper limits of the confidence interval. The $ at the end of the function statement determines that the value of CONCATP output by SAS in the return statement is expected to be a string.

This example would create a confidence interval where the upper and lower limit values have two decimal places, inside square brackets and separated by a semicolon.

In order to use this function, first we need to identify the FCMP data set containing the function using the CMPLIB option.

```
options cmplib=work.functions;
```

The function CI is then called in a data step or a PROC SQL command. The input data set need not use the same variable names LCL and UCL, as used when the function is compiled. For example.

```
proc means data=sashelp.class clm alpha=0.05;
    var weight;
    class sex;
    output out=clmweight (where=(_TYPE_=1)) mean=mean1 lclm=lclm uclm=uclm;
run;

data mci;
    set clmweight;
    mci = ci(lclm, uclm);
run;
```

In this example, the upper and lower limit of the weight variable are saved as LCLM and UCLM in the PROC MEANS output data set ***clmweight***, and the standardized confidence interval is created by using LCLM and UCLM as arguments to the function CI in the data step.

CREATING A FUNCTIONS DATA SET

The primary reason for using PROC FCMP code is to create a permanent data set of reference functions. This may be saved, for instance, within a package folder for a suite of outputs; or in a higher level folder to be used across multiple packages. For example, the function CI can be saved in any location the user has access to using PROC FCMP with an updated OUTLIB option, such as:

```
proc fcmp outlib = oncology.functions.statistics;
```

Oncology is the library where the FCMP data set is saved. The data set can be viewed using a PROC PRINT on ***oncology.functions.*** The most informative variables are: _KEY_, TYPE, SUBTYPE and VALUE

In the FCMP data set, the function name is stored within the variable _KEY_ as <F.Package Name.Function Name> and our executable code is saved in the variable VALUE as executable and declarative types where the TYPE variable is Statement Source.

For example:

KEY	TYPE	SUBTYPE	VALUE
STATISTICS	Header	Package	<L n="Header"><S n="Version"><![CDATA[1.1]]></S><N
F.STATISTICS.CI	Prototype	FCmp	<L n="Prototype"><S n="Name"><![CDATA[ci]]></S><S
F.STATISTICS.CI	Prototype	FCmp	<N n="Flag7">0</N><N n="Flag8">0</N></L></L></L>
F.STATISTICS.CI	Header	Function	<L n="Header"><S n="Version"><![CDATA[1.1]]></S><N
F.STATISTICS.CI	Statement Source	Executable	function ci(lcl, ucl) $;
F.STATISTICS.CI	Statement Source	Declarative	length concatp $100;
F.STATISTICS.CI	Statement Source	Executable	concatp = cats("[", strip(put(lcl,8.2)),'; '
F.STATISTICS.CI	Statement Source	Executable	return(concatp);
F.STATISTICS.CI	Statement Source	Executable	endsub;
F.STATISTICS.CI	Symbol		<L n="Symbol"><S n="Name"><![CDATA[_HOSTNAME_]]></
F.STATISTICS.CI	Symbol		<L n="Symbol"><S n="Name"><![CDATA[_ci_]]></S><S n
F.STATISTICS.CI	Symbol		<L n="Symbol"><S n="Name"><![CDATA[concatp]]></S><

Additional functions can be added to this data set which will be assigned a different value of the variable _KEY_. It is possible to add functions with the same function name under a different package name in the same data set.

ADDING, DELETING, AND RENAMING FUNCTIONS

Once a FCMP data set is established, it's possible to manage a suite of functions by adding new functions, deleting redundant functions and renaming current functions.

EXAMPLE: ADDING A FUNCTION THAT CREATES A CONFIDENCE INTERVAL WITH USER DEFINED DECIMAL PLACES

In the function CI described above the number of decimal places is fixed. This function can be generalized to allow user-defined number of decimal places. This information can either be saved as a variable in the input data set or passed as an argument to the function. However, the FCMP procedure has its limitations and does not have the same flexibility as creating a macro. For instance, the following code tries to add a user defined constant to the format in the put statement and will fail at compilation.

```
function cidp(lcl, ucl, dec) $ ;
length CONCATP $100;
CONCATP = cats( "[", put(lcl,8.dec),'; ', put(ucl,8.dec), "]");
return(CONCATP);
endsub;
```

In this instance, SAS will attempt to compile the function and will not recognize the format of 8.dec. However, it is possible to code an **if-then** block to check for user-defined number of decimal places and create the required confidence interval value.

```
function cidp(lcl, ucl, dec) $ ;
length CONCATP $100;
if dec=1 then CONCATP=cats("[",put(lcl,8.1),'; ',put(ucl,8.1),"]");
if dec=2 then CONCATP=cats("[",put(lcl,8.2),'; ',put(ucl,8.2),"]");
if dec=3 then CONCATP=cats("[",put(lcl,8.3),'; ',put(ucl,8.3),"]");
if dec=4 then CONCATP=cats("[",put(lcl,8.4),'; ',put(ucl,8.4),"]");
if dec=5 then CONCATP=cats("[",put(lcl,8.5),'; ',put(ucl,8.5),"]");
return(CONCATP);
endsub;
```

Alternatively, the **if-then** block can be circumvented by setting a standard format in the function statement.

```
proc fcmp outlib = oncology.functions.statistics;
    function cidp(lcl, ucl, fmt ) $ ;
    length CONCATP $100;
    CONCATP=cats("[", putn(lcl,fmt),'; ', putn(ucl,fmt),"]");
    return(CONCATP);
    endsub;
quit;
```

This will add another function CIDP in the ***statistics*** data package to the ***functions*** data set in the ***oncology*** folder. The existing function CI can be deleted using the **DELETEFUNC** option within the FCMP procedure. Option CMPLIB is required at this step to define the current library.

```
options cmplib=oncology.functions;

proc fcmp outlib= oncology.functions.statistics;
   deletefunc ci;
run;
```

There is no simple method to rename an existing FCMP function. However, a function can be overwritten although this will produce a WARNING in the log file. In order to prevent the WARNING message, it is preferred to delete the existing function before a new function is created using separate PROC FCMP steps.

CREATING A NEW FUNCTION USING AN EXISTING FUNCTION

Any data step or procedure, including the PROC SQL statement, can use a function from the functions data set provided that procedure can access the functions data set.

EXAMPLE: DISPLAYING A STATISTIC WITH THE CONFIDENCE INTERVAL

One procedure that can access this functions data set is the FCMP procedure itself. Therefore, a PROC FCMP step can create a new function using an existing function. For example, the following function displays the statistic alongside the confidence interval with a user defined format for the number of decimal places for both the statistic and its confidence interval.

```
options cmplib=oncology.functions;
proc fcmp outlib = oncology.functions.statistics;
    function xci(val, lcl, ucl, fmt) $ ;
    length CONCATP $100;
    CONCATP = catx(' ', putn(val,fmt), cidp(lcl, ucl, fmt)) ;
    return(CONCATP);
    endsub;
quit;
```

This example creates a new function XCI used to concatenate the statistic and its confidence interval where the confidence interval is created using a previously compiled function CIDP. Function XCI can then be called in either a data step or a PROC SQL procedure to create required variables.
The following example is using previously defined PROC MEANS output data set ***clmweight*** to create required variables with examples using both data step and PROC SQL procedures.

```
data mci;
    set clmweight;
    mci = xci(mean1, lclm, uclm, 8.2);
run;

proc sql noprint;
    create table mci as select xci(mean1, lclm, uclm, 8.2) as mci from clmweight;
quit;
```

CONCLUSION

Proc FCMP can be used as a tool to add efficiencies to output packages, encourage standardization across outputs and reduce programming time spent on repetitive tasks. PROC FCMP allows the construction of a library of user defined functions, stored as SAS data sets accessible to multiple users and can be implemented across multiple projects producing consistent and standardized outputs.

REFERENCES

Carpenter's Guide to Innovative SAS® Techniques
Garcia, Christina. 2016. "The Power of the Function Compiler: PROC FCMP", published in the Proceedings of WUSS 2016 Conference.

CONTACT INFORMATION

Paul Wicks
MSD
MRL Innovation Hub
Two Pancras Square
London N1C 4AG
United Kingdom
paul.wicks@msd.com

Anasooya Anoop
MSD
MRL Innovation Hub
Two Pancras Square
London N1C 4AG
United Kingdom
anasooya.anoop@msd.com

Anna Socha
MSD
MRL Innovation Hub
Two Pancras Square
London N1C 4AG
United Kingdom
anna.socha@msd.com