Paper CT15

Building Reusable Precision Logic in SAS Using PROC FCMP

Mariam Babayan, BioBrain Pro LLC, Yerevan, Armenia
Syuzanna Simonyan, BioBrain Pro LLC, Yerevan, Armenia

ABSTRACT

This paper introduces a focused and practical approach to utilizing PROC FCMP in SAS for building reusable, user-defined functions, demonstrating its power and flexibility through clear, real-world programming examples.

As analytical programming evolves, there's an increasing need for modular, transparent, and testable logic, especially for tasks that appear simple but grow complex in regulatory, clinical, or data reporting environments. This paper demonstrates the full power and flexibility of PROC FCMP through practical examples with broad relevance to real-world programming challenges.

Beyond the technical strategy, the paper emphasizes how reusable logic and structured programming practices can increase adaptability, reduce rework, and strengthen.

INTRODUCTION

Small tasks often reveal big opportunities for code optimization. In data reporting, even routine operations can benefit from consistent, centralized logic. Whether for formatting, validation, transformation, or comparison, having reusable and reliable functions enhances efficiency and clarity. This paper explores that concept through a clean, modular lens: build a function once, use it everywhere. With PROC FCMP, a single logic block can be applied across projects, datasets, and outputs—reducing the need for ad-hoc solutions and promoting consistency.

PROC FCMP

Tasks that appear simple at first—like formatting or data checks—can become more complicated in regulated reporting settings, where accuracy and consistency are critical. PROC FCMP makes it easier to manage this complexity by letting you create reusable functions that keep your code clean, consistent, and easy to maintain across different SAS programs. The primary advantage of using custom functions created with PROC FCMP over SAS macros is that FCMP functions can be invoked directly within DATA steps and many SAS procedures, without the need for macro expansion. The versatility of being able to be called from other procedures can provide more convenience than having to make a separate macro call in various situations. Although PROC FCMP is often introduced through functions, it’s important to know that subroutines can be used on their own — and in some advanced workflows, they’re the preferred structure for returning multiple values.

COMMON CHALLENGES WITH PROC FCMP HANDLING

When handling tasks in PROC FCMP, common challenges include inconsistent rules across datasets, format mismatches in regulatory outputs, managing reporting precision, distinguishing between truncated values, floating-point comparison issues, and handling inconsistent validation logic.
By turning these tasks into reusable functions, PROC FCMP will make the code more consistent, reduce repetition, and make it easier to maintain across SAS programs.

PROGRAMMING SOLUTIONS

By creating a single, reusable FCMP function, programmers gain flexibility and adaptability. This section explores the practical applications of such logic, including its role in:

- Dynamically formatting outputs
- Detecting rounding or precision issues
- Supporting variable-level consistency checks
- Ensuring numeric consistency in key variable comparisons

BASIC STRUCTURE

```
proc fcmp outlib=<libref.dataset.package>;

   /* Function: Returns a single value */
   function function_name(arg1, arg2, ...) return_type;
      /* Logic goes here */
      return(result);
   endsub;

   /* Subroutine: Returns multiple values */
   subroutine subroutine_name(arg1, arg2, ..., output1, output2, ...);
      outargs output1, output2, ...;
      /* Logic goes here */
   endsub;

quit;
```

***OUTLIB*:** Specifies the three-level name of an output package where functions and routines are stored. The first level specifies the library name, followed by the dataset where functions are stored, and then the specific package or group name.

***Function*:** Defines a custom function you want to use later.
***Return_type*:** Optional for numeric (default); use $ for character output (e.g., $20.).
***Return*:** Specifies the value that is returned by the function. Programming statements are a series of DATA step statements that describe the function's actions.
Outargs: Required in subroutines to define output variables
***ENDSUB*:** Closes the function or subroutine block.
***QUIT*:** Exits the procedure.

HOW USE A FUNCTION DEFINED IN PROC FCMP

To utilize a function that you have created with PROC FCMP, you need to set the CMPLIB= system option. This option points to the library and dataset where the compiled functions are stored, using a two-level name. The specified location in CMPLIB= must correspond exactly to the first two parts of the OUTLIB= name you provided in the PROC FCMP step.

```
options cmplib=libref.dataset;
```

AUTOMATING SUMMARY STATISTIC PRECISION USING PROC FCMP

In this step, we define a user-written function in PROC FCMP called count_decimals.
The function takes a number as input and converts it to a string. If the value has no decimal point, it returns 0. If a decimal point exists, it isolates the fractional part (digits after the dot). Finally, it returns the length of the fractional part, i.e., the number of decimal places.
By storing this function in a function library (myfuncs.dec.utils) and activating it with the cmplib option, we can reuse it throughout our program. This allows us to determine the maximum number of decimals per parameter and apply consistent formatting in summary statistics tables.

```
proc fcmp outlib=myfuncs.dec.utils;
  function count_decimals(num);
      length str $32;
      s2 = strip(put(num, best.));
      if s2 = '' then return(0);
      if index(s2, '.') = 0 then return(0);
      decpart = strip(scan(s2, 2, '.'));          /* part after the dot */
      return(length(decpart));

  endsub;
quit;
```

After identifying the maximum number of decimal places for each parameter, descriptive statistics are computed by parameter, visit, and treatment group. These statistics (N, Mean, SD, Median, Min, and Max) are then formatted according to the parameter-specific decimal precision. This ensures that all results are presented with consistent and accurate formatting, aligned with the level of precision observed in the raw data.

In most Statistical Analysis Plans (SAPs), we encounter a standard convention for reporting summary statistics of numerical variables:

Minimum and Maximum are displayed at the same level of precision as the raw data. Mean and Median are displayed to one additional decimal place beyond the raw data. Standard Deviation / Standard Error are displayed to two additional decimal places beyond the raw data. This ensures consistency and appropriate precision across all reported results.

To implement this convention, a custom PROC FCMP function was developed.

The function applies the appropriate number of decimal places to each statistic based on the maximum decimal precision identified per parameter.

This ensures that all reported results (e.g., Minimum, Maximum, Mean, Median, Standard Deviation) are automatically formatted according to the SAP requirements.

The approach guarantees both consistency and traceability, as the formatting directly reflects the precision observed in the raw data.

```
proc fcmp outlib=myfuncs.round_dec.utils;
function format_min(val, decimals, min);
  length result $20;
  if min = 0 then factor = 10**decimals;
  else if min = 1 then factor = 10**(decimals+1);
  else if min = 2 then factor = 10**(decimals+2);
  max_dec = 10**3;
  factor_min = min(factor,max_dec);
  rounded = round(val, 1/factor_min);
  result = strip(put(rounded, best32.));
  return(result);
endsub;
quit;
```

The format_min function was created in PROC FCMP to apply SAP precision rules when formatting descriptive statistics automatically.

Inputs:

- Val – the statistic value (e.g. mean, SD, min).
- Decimals- the maximum number of decimals observed for the parameter(maxdec).
- Stattype – the statistic type:
  0 = Min/Max -> same precision as data.
  1 = Mean/Median -> one extra decimal place.
  2 = SD/SE -> two extra decimal places.

Logic:

- Determines required precision by adjusting decimals based on the statistic type.
- Caps maximum precision at 3 decimals to avoid over-precision.
- Rounds the value to the correct number of decimals and converts it into a text string.

Output:

- Returns the statistic formatted consistently with SAP conventions.
- Ensures results are displayed with the appropriate level of precision across all parameters, visits, and treatment groups.

Creating and Reusing Derived Parameters Using PROC FCMP: An Example

The following example demonstrates how to define and calculate a custom parameter, such as a derived endpoint or transformation, and implement the logic within a reusable function or subroutine using PROC FCMP. This logic is then saved in a compiled library, allowing it to be used repeatedly across multiple programs. This approach enhances consistency, ensures validated results, improves code modularity, reduces duplication, and supports traceability within clinical programming workflows.

In this example, the Attention Composite Z-Score is derived from cognitive performance data collected across multiple study visits. The calculation is based on three cognitive test endpoints: CTB Detection, CTB Identification, and One Back – Speed of Performance. For each participant, scores recorded at Baseline and at Weeks 2, 4, 6, and 10 are transformed into Z-scores using the mean and standard deviation at Baseline, calculated across the analysis population.

Z-scores are calculated at Baseline, Weeks 2, 4, 6, and 10 using this formula:

$$z = \frac{x - \bar{x}}{SD}$$

Where:

- X = participant endpoint response,
- $\bar{x}$ = baseline mean for all participants in the analysis population,
- SD = baseline standard deviation for all participants in the analysis population).

Since lower values indicate better performance, each Z-score is multiplied by -1 to ensure higher scores = better performance.

Here are several records from the dataset used to calculate the Attention Composite Z-score.

Obs	SUBJID	PARAM	PARAMCD	AVAL	AVISIT	AVISITN
1	003-005	Detection	DET	2.68907	Baseline	1
2	003-005	Detection	DET	2.65156	Week 2	2
6	003-005	Identification	IDN	2.82111	Baseline	1
7	003-005	Identification	IDN	2.85957	Week 2	2
11	003-005	One Back Tests - Speed	ONBSPD	3.02192	Baseline	1
12	003-005	One Back Tests - Speed	ONBSPD	3.13437	Week 2	2

IMPLEMENTATION USING PROC FCMP

```
proc fcmp outlib=work.cognitive.utils;

   /* Z-score with reversed directionality */
   function zscore_reverse(x, mean, sd);
      if missing(x) or sd = 0 then return(.);
      z = (x - mean) / sd;

      return (z);
   endsub;

   /* Compute Attention Composite Score */
   function attention_composite(z1, z2, z3);
      count = 0;
      sum = 0;
      if not missing(z1) then do;
         sum + z1;
         count + 1;
      end;
      if not missing(z2) then do;
         sum + z2;
         count + 1;
```

```
        end;
        if not missing(z3) then do;
           sum + z3;
           count + 1;
        end;

        if count >= 2 then do;
             z4 = sum / count;
             return (-1*z4);
        end;
        else return(.);
    endsub;

quit;
```

This step creates two reusable functions using PROC FCMP:
zscore_reverse() calculates a Z-score and maintains the logic for reversing directionality (higher is better).
attention_composite() computes the average of the available Z-scores, only if at least two of the three are non-missing.
In the example, the procedure begins with PROC FCMP OUTLIB =, which specifies the location where the function will be stored. The next line starts with FUNCTION, followed by a customizable name that should be meaningful based on the function's purpose.
The function takes three arguments: X (the AVAL value), MEAN (the mean value of baseline records), and SD (the standard deviation of baseline records). These arguments are used in the programming statements to calculate the returned value.
Within the body of the function, a new variable is assigned to calculate the Attention Z-score. Finally, the function returns this value.

```
options cmplib=work.cognitive;
```

This option tells SAS where to find the user-defined functions from PROC FCMP so they can be used in subsequent data steps.

When the dataset is reshaped from long to wide format, each cognitive test (Detection, Identification, One Back) becomes a separate variable (e.g., Det, Idn, Onbspd) for each subject and visit.

Obs	SUBJID	AVISITN	AVISIT	_NAME_	_LABEL_	DET	IDN	ONBSPD
1	003-005	1	Baseline	AVAL	Analysis Value	2.68907	2.82111	3.02192
2	003-005	2	Week 2	AVAL	Analysis Value	2.65156	2.85957	3.13437
3	003-005	4	Week 4	AVAL	Analysis Value	2.65226	2.86338	3.09768
4	003-005	6	Week 6	AVAL	Analysis Value	2.62272	2.85730	3.14096
5	003-005	10	Week 10	AVAL	Analysis Value	2.75089	2.85943	3.03945

Below step computes the mean and standard deviation for each endpoint at Baseline, across all participants. These values will be used to calculate Z-scores for all subsequent visits. This final step merges the baseline reference values into the dataset and applies the custom functions to:

- Calculate Z-scores for each endpoint.
- Compute the Attention Composite Z-score for each subject and visit, only if at least two Z-scores are non-missing.

```
/* Step 3: Compute Baseline means and SDs for each endpoint */
proc means data=cognitive_data noprint;
   where aVisit = "Baseline";
   var Det idn Onbspd;
   output out=baseline_stats mean=mean_det mean_id mean_ob
                          std=std_det std_id std_ob;
```

```
run;

/* Step 4: Merge baseline stats into main dataset */
data cognitive_with_stats;
   if _n_ = 1 then set baseline_stats(keep=mean_: std_:);
   set cognitive_data;

   /* Compute Z-scores with direction reversed */
   z_det = zscore_reverse(Det, mean_det, std_det);
   z_id  = zscore_reverse(Idn, mean_id, std_id);
   z_ob  = zscore_reverse(Onbspd, mean_ob, std_ob);

   /* Compute composite */
   attn_composite = attention_composite(z_det, z_id, z_ob);
run;
```

And finally this dataset contains the calculated Z-scores and the final Attention Composite Z-score(attn_composite) for each participant at each visit.

Obs	mean_det	std_det	SUBJID	AVISITN	AVISIT	DET	IDN	ONBSPD	z_det	z_id	z_ob	attn_composite
1	2.58302	0.10096	003-005	1	Baseline	2.68907	2.82111	3.02192	1.05034	1.14177	1.34805	-1.18005
2	2.58302	0.10096	003-005	2	Week 2	2.65156	2.85957	3.13437	0.67887	1.69536	2.61220	-1.66214
3	2.58302	0.10096	003-005	4	Week 4	2.65226	2.86338	3.09768	0.68580	1.75020	2.19973	-1.54524
4	2.58302	0.10096	003-005	6	Week 6	2.62272	2.85730	3.14096	0.39322	1.66268	2.68628	-1.58073
5	2.58302	0.10096	003-005	10	Week 10	2.75089	2.85943	3.03945	1.66270	1.69334	1.54512	-1.63372
6	2.58302	0.10096	003-008	1	Baseline	2.55735	2.75765	2.98786	-0.25430	0.22834	0.96510	-0.31304
7	2.58302	0.10096	003-008	2	Week 2	2.58883	2.78907	2.95831	0.05755	0.68059	0.63296	-0.45703
8	2.58302	0.10096	003-008	4	Week 4	2.78243	2.81833	2.91833	1.97509	1.10176	0.18351	-1.08679
9	2.58302	0.10096	003-008	6	Week 6	2.61752	2.80574	2.98588	0.34171	0.92054	0.94290	-0.73505
10	2.58302	0.10096	003-008	10	Week 10	2.57955	2.79379	2.90943	-0.03437	0.74853	0.08346	-0.26587

CONCLUSION

PROC FCMP offers a modern, functional approach to SAS programming. A single, well-designed function can be widely reused to address various programming challenges with clarity and precision. This paper demonstrates how focusing on one practical function highlights the broader value of structured, adaptable coding practices.

REFERENCES

SAS Institute Inc. (2021). *SAS® 9.4 Functions and CALL Routines: Reference*. Cary, NC: SAS Institute Inc. https://documentation.sas.com/doc/en/pgmsascdc/v_064/proc/n16ijqsjbv5mdbn1c6w7j6elpky2.htm

ACKNOWLEDGMENTS

The authors thank the PHUSE community for providing a platform to share this work and for encouraging the exchange of innovative programming ideas.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Authors: Syuzanna Simonyan, Mariam Babayan
Company: Biobrain Pro LLC
Address: 2/2 A Mikoyan Street, 0054, Yerevan, Armenia
Emails: syuzie.ss@ biobrainpro.net, marbabayan07@biobrainpro.net
Website: biobrainpro.net