



PHUSE US Connect 2025
Slides SM16

Table Numbers Presentation Made Easy with Custom Functions

SUNIL KUMAR PUSARLA, *SEATTLE, USA*

User-Defined functions (UDFs)

- SAS provides numerous built-in functions, but they may not always meet specific needs.
- Definition of UDFs: Custom functions created by users to extend SAS functionality and improve code efficiency.
- Created using: PROC FCMP
- Key Points About PROC FCMP:
 - Allows defining custom functions and subroutines
 - Functions can return values, while subroutines modify variables
 - Functions can be stored in libraries for reuse

Sample code

- Syntax: Within PROC FCMP, “FUNCTION” statement is used to define the function name, arguments, and the code to execute, and “RETURN” to specify the value to be returned
- Sample code:

```
PROC FCMP OUTLIB = libref.data-set.package;  
    Function –name (argument-1, ... argument-n);  
        Programming statements;  
        Return (expression);  
    ENDSUB;  
run;
```

Overview

- SAS statistical procedures generate results using default numeric formats like BESTw.
- Reporting often requires custom arrangements, e.g., Mean (SD), n (%), Confidence Limits (CLMs).
- Figures: via XAXISTABLE or by annotations.
- Programming specifications change for different use cases like regulatory submissions, conference abstracts, and reports.
- Necessitates repetitive use across multiple panels, parameters followed by post-processing steps.

Overview

- PROC FCMP enables the creation of custom functions and subroutines.
- Built-in SAS functions may require repetitive coding and post-processing.
- Custom functions ensure consistency, reduce errors, and improve efficiency.
- Functions can be referenced using CMPLIB system option for reusability.

Methodology

- **Components of building a robust custom function.**
 - Defensive coding (to account missing data).
 - Dynamic handling (to handle specification changes).
 - Post-processing/refining (for publication readiness).
- **Create formats for each statistic.**
- **Use N/NMISS to handle missing values.**
- **PUTN to handle formats dynamically.**
- **LENGTH, TRANWRD, SUBSTR etc. to refine.**

Methodology – Using PROC FORMAT & PUTN Function

- PROC FORMAT defines the required/specified formats.
- PUTN Function applies the correct format at runtime.
- Formatted created can be used inside the functions ensuring seamless future adjustments.

Creating required format based on specification	Using this format and SAS functions while creating user defined functions using PROC FCMP
<pre>proc format; value \$statfmt 'N' = 6. 'STD' = 6.3 'Mean' = 6.2 'Median' = 6.1 'Minimum' = 6.1 'Maximum' = 6.1 'Percent' = 6.1 'UCLM', 'LCLM' = 8.4; run;</pre>	<pre>/* To assign the format of Percent with 1 decimal from the \$statfmt created */ y1 = strip (putn (xxx, put ('Percent', \$statfmt))); /* To assign the format of Mean with 2 decimals & STD with 3 decimals from the \$statfmt created */ x1 = strip (putn (x, put ('Mean', \$statfmt))); y1 = strip (putn (y, put ('STD', \$statfmt)));</pre>

Use Case 1 – Presenting 'n (%)'

► Traditional Approach (Without User-Defined Functions)

- Converting the numeric into a character using the PUT function with the appropriate format.
- Removing leading and trailing blanks.
- Concatenate '(' when presenting combinations.
- Concatenate the combination stat result
- Append the above with a '%)'.

➤ Optimized Approach (With User-Defined Functions)

- Define a function for **n (%) formatting** using PROC FCMP.
- Apply across different tables and reports

Use Case – Presenting 'n (%)'

Without using the user defined function	Using user defined function
<pre>if n (placebo) eq 1 then col1 = strip (put (placebo, 4.)) '(' strip (put ((placebo / &col1.)*100, 6.1)) '%)'; else col1 = '0'; if (col1 eq '100.0%') then col1 = '100%'; else if (col1 eq '0(0.0%)') then col1 = '0';</pre>	<pre>col1 = npct (placebo, &col1);</pre>

Sample code

```
1  libname myflib 'C:\...\FCMP';
2
3  proc fcmp outlib = myflib.myfunc.report;
4
5  /* Concatenating the n with percent (after creating it) */
6
7      function npct(x,y) $40;
8      length want x1 y1 y2 $100;
9
10     if (n(x) eq 1) then x1 = strip(put(x,best.));
11     else                x1 = '0';
12     if ((n(x,y) eq 2) and
13         (y ne 0)) then Pct = (x/y)*100;
14     if n(pct) eq 1 then y1 = strip(putn(Pct,
15                                         put('Percent',
16                                             $statfmt.)
17                                         ));
18
19     if n(pct) eq 1 then y2 = strip(put(pct, best.));
20     if (length(y2) lt length(y1)) then y1 = y2;
21
22     if ((n(x) ne 1) or
23         (x eq 0) or
24         (y eq 0) or
25         (n(y) ne 1)) then y1 = '';
26
27     if ((length(y1) ge 3) and
28         (substr(y1,1,3) eq '100')) then y1 = '100';
29
30     want = cats(x1,'(',y1,'%');
31     if (y1 eq '') then want = '0';
32     want = tranwrd(want,'(',' ');
33     return(want);
34 endsub;
35 run;
```

Use Case 2 – Presenting Mean (SD)

➤ Traditional Approach

- Convert Mean and SD to character variables.
- Apply individual PUT functions for each format.
- Concatenate the Mean and STD values.
- Post-processing necessary.

➤ Optimized Approach (Using Custom Function)

- Automatically applies specified formatting.
- Can yield results with different formats.

Use Case 2 – Presenting Mean (SD)

Without using the user-defined function	Using user-defined function
<pre>length meanc stdc meansd \$40; if n (mean) eq 1 then meanc = strip (put (mean, 8.2)); if n (std) eq 1 then stdc = strip (put (std, 8.3)); meansd = strip (meanc) '(' strip (stdc) ')'; /* Post-processing steps required */ meansd = tranwrd (meansd, '()', ''); meansd = scan (meansd, 1, '.');</pre>	<pre>col2 = <i>meanstd</i> (mean, std); /* Can yield results with different formats */ Visit 13: 25.72 (3.693) Visit 14: 26.12 (3.8) Visit 15: 25 (4.382) Visit 16: 23.1 (3.42) Visit 17: 23</pre>

Sample code

```
1  libname myflib 'C:\...\FCMP';
2
3  proc fcmp outlib = myflib.myfunc.report;
4
5  /* Concatenating the mean and standard deviation */
6
7      function MeanStd(x,y) $40;
8      length want x1 y1 x2 y2 $100;
9
10     if (n(x) eq 1) then do;
11         x1 = strip(putn(x,put('Mean', $statfmt.)));
12         x2 = strip(put(x,best.));
13         if (length(x2) lt length(x1)) then x1 = x2;
14     end;
15
16     if (n(y) eq 1) then do;
17         y1 = strip(putn(y,put('STD', $statfmt.)));
18         y2 = strip(put(y,best.));
19         if (length(y2) lt length(y1)) then y1 = y2;
20     end;
21
22     if (n(x,y) eq 2) then want = cats(x1,'(',y1,')');
23     if ((n(x) eq 1) and (n(y) ne 1)) then want = x1;
24     if f (n(x,y) lt 1) then want = '';
25     want = tranwrd(want,'(',' ');
26     return(want);
27 endsub;
28 run;
```

Conclusion

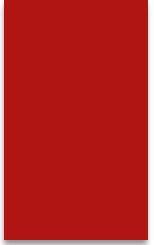
- Custom functions in SAS improve formatting efficiency and minimize manual intervention.
- Using PROC FCMP, PROC FORMAT, and PUTN, programmers can create flexible, reusable functions.
- Key Takeaways:
 - Simplifies code complexity.
 - Reduces risk of errors in large-scale reports.
 - Enhances adaptability to evolving specifications.
 - Incorporates defensive coding
 - Ensures consistency across programs.

Disclaimer

- The opinions expressed in this paper are those of the author and do not represent the views of Omeros Corporation.



Questions?



Author: Sunil Kumar Pusarla

Enterprise: Omeros Corporation

Address: 201 Elliott Avenue West, Seattle, WA 98119

Work Phone: 206-676-5000

E-mail: skumar@omeros.com

Thank you