MWSUG 2024 - Paper BB-17

Just Stringing Along: FIND Your Way to Great User-Defined Functions

Richann Jean Watson, DataRich Consulting;
Louise S. Hadden, Independent Consultant

ABSTRACT

SAS® provides a vast number of functions and subroutines (sometimes referred to as CALL routines). These useful scripts are an integral part of the programmer's toolbox, regardless of the programming language. Sometimes, however, pre-written functions are not a perfect match for what needs to be done, or for the platform that required work is being performed upon. Luckily, SAS has provided a solution in the form of the FCMP procedure, which allows SAS practitioners to design and execute User-Defined Functions (UDFs). This paper presents two case studies for which the character or string functions SAS provides were insufficient for work requirements and goals and demonstrate the design process for custom functions and how to achieve the desired results.

INTRODUCTION

The first scenario to be addressed is the need to both order and deduplicate "words" within an existing string, creating a new string. This can be useful when dealing with validating a list of new or modified fields. This can be accomplished by using several different functions (SCAN, FIND, and COUNTW) as well as the SORTC subroutine within a DATA step, but a single function combining the actions these separate functions and subroutine perform is desirable. The second case study will delve into a scenario in which information regarding variable formats is required, and an existing SAS function is insufficient. The existing function, FMTINFO, only applies to SAS-supplied functions. (Hadden L. S., 2023). The task to create an enhanced version of the FMTINFO function that works for all functions can be performed using PROC FCMP to create a user-defined function. Examples are performed in SAS 9.4 version 7.

USING PROC FCMP TO CREATE COMPOSITE STRING FUNCTIONS

PROC FCMP (the SAS Function Compiler) allows SAS practitioners to use the SAS Language Compiler to create custom, reusable, functions, and subroutines. These custom pieces of code can be stored and reused by the DATA step, WHERE statements, ODS, and selected procedures. The examples described in this paper are functions that operate within the DATA step.

REORDER AND DEDUPLICATE WORDS WITHIN A STRING

While working on validation of patient profiles, one of the things that needed to be checked were fields or records that were added or modified from the previous run. Production stored all the modified fields in one variable. This variable was used to control the coloring scheme on the patient profiles, so it was important to confirm that the fields stored in this variable were accurate. In order to do so, validation needed to create a similar field and make sure there were no duplicate values and that it was properly sorted in order to compare with production. To achieve this a macro was created that utilized the DATA step with a DO loop. This macro was called for each profile data set generated. After further additions were made, it was decided that perhaps a user-defined function that will take a string de-dupe and order the values would be beneficial.

DATA STEP AND MULTIPLE FUNCTION SOLUTION

To illustrate the concept of the UDF, we first illustrate how this was done with the basic DATA step, using a simple data set as shown in Table 1. In the actual patient profile work, there were numerous data sets with numerous subjects and each subject having a variety of newly added and modified fields.

SUBJID	STRING
001	AVAL, BASEC, BASE, AVALC, BASEC, BASE, PCHG, CHG, R2BASE, CHG, PARAMCD, PARAMN, PARAM
002	AVALC, AVAL, AVALC, ANRLO, ANRHI, PCHG, CHG, PCHG

Table 1: Sample Data for Sorting and De-Duping Values in a Variable

The goal is to take STRING and reorder the individual tokens (words) as well as remove any duplicate tokens. Note that the duplicate tokens are noted in red text. This can be achieved in the DATA step as illustrated in Program 1.

```
data want (keep = SUBJID string newstring);
   set have;
   length found $2000;
   num = countw(string, ' ');  ❶
   array vals{700} $8 ;   ❷
   do i = 1 to num;
      vals{i} = scan(string, i, ', ');   ❸
      x = find(cats('#', found, '#'), cats('#', vals{i}, '#'), 'it');   ❹
      if x > 0 then call missing(vals{i});   ❺
      else found = catx('# ', found, vals{i});
   end;

   call sortc(of vals:);   ❻
   newstring = catx(' ', of vals{*});   ❼
run;
```

Program 1: DATA Step for Sorting and De-Duping Values in a Variable

❶ Determine the number of tokens in each string so that we can use this for the DO loop.

❷ Create an array that does not contain anything initially. The purpose is to create the array elements so that we can populate the elements with the individual tokens. The value of 700 was arbitrarily picked and has no significant meaning. It just needed to be large enough to capture each token in its own variable.

❸ For each iteration of the DO loop, the token at the i^{th} location in the string is extracted and saved as the i^{th} array element.

❹ Each token is then searched for in the FOUND variable. Note that FOUND is put into the PDV before the DO loop with the LENGTH statement so for the first iteration of the DO loop FOUND is null. We temporarily enclose each string with a delimiter in case the token can be found within another token. For example, if we searched for CHG, this would be found within PCHG and the value of X would be greater than 0.

❺ If the token is found within FOUND variable, then the array element is reset to null. If it is not found in the FOUND variable, then the array element is appended to the FOUND variable with each token separated by #.

❻ After the DO loop has completed all the array elements are sorted using the CALL SORTC subroutine.

❼ The sorted array elements are then concatenated and separated by a space using the CATX function.

The end results of Program 1 can be seen in Table 2.

SUBJID	NEWSTRING
001	AVAL AVALC BASE BASEC CHG PARAM PARAMCD PARAMN PCHG R2BASE
002	ANRHI ANRLO AVAL AVALC CHG PCHG

Table 2: WANT Sample Data for Sorting and De-Duping Processed in a DATA Step

While this logic can be 'wrapped' in a macro and the macro invoked within each of the programs used for the validation of the patient profile data sets, there is a more elegant solution.

USER-DEFINED FUNCTION (UDF) SOLUTION

The desired result was to have a function that could perform the de-duping of tokens in a string and re-order them without having to use the macro facility. In other words, we wanted a user-defined function. With the use of PROC FCMP, we were able to create a UDF that would be compiled and executed like any other SAS function.

```
Program 2 shows the general syntax for using PROC FCMP to create user-defined subroutines
and functions.
PROC FCMP outlib = libname.dataset.package  <options>;
   SUBROUTINE subroutine-name (argument-1 <, argument-2, …>);
      OUTARGS out- argument-1 <, out-argument-2, …>;
      <<< SAS STATEMENTS >>>
   ENDSUB;
   FUNCTION function-name (argument-1 <, argument-2, …>) <$>;
      OUTARGS out- argument-1 <, out-argument-2, …>;
      <<< SAS STATEMENTS >>>
      RETURN(expression)
   ENDFUNC;
QUIT;
```

Program 2: PROC FCMP General Syntax

PROC FCMP allows for the use of general SAS language statements, therefore, we are able to take the original program (Program 1), we can utilize most of the code within PROC FCMP with a few modifications as seen in Program 3.

```
libname fcmp '<<<DIR WHERE UDF IS STORED>>>';    ❶
options cmplib = fcmp.funcs;    ❷
proc fcmp outlib = fcmp.funcs.SORTDEDUPE;    ❸

   function sortvals(vlist $) $;    ❹
      length found sortlist $2000;    ❺
      len = countw(vlist, ', ');
      array temp[700] $8;    ❻
      do i = 1 to len;
         temp[i] = scan(vlist, i, ', ');
         if find(cats(',', sortlist, ','), cats(',', temp[i], ','), 'it') > 0
            then temp[i] = '';
❼        else sortlist = catx(',', sortlist, temp[i]);
      end;
      call sortc(of temp[*]);
      sortlist = catx(' ', of temp[*]);
      return (sortlist);    ❽
   endsub;
run;
```

Program 3: UDF for Sorting and De-Duping Values in a Variable

❶ Before we can create our UDF, we need to specify a location where the UDF will reside.

❷ In addition, we need to specify the system option CMPLIB and provide the name of the data set where the package that contains the UDF will be stored. The data set name is a two-level naming convention, <libname>.<dataset>.

❸ We need to provide the name of the package where the UDF will be stored for future use. The package name is a three-level naming convention, <libname>.<dataset>.<package>.

❹ Within PROC FCMP, we can define custom subroutines or functions. The appropriate statement is used to indicate which one is being created. Each custom subroutine or function requires a name and any arguments that are to be specified. If the input argument is character it needs to be followed by a $. If the function return value is character, then a $ also needs to be indicated after the closing parenthesis of the function name and argument list.

❺ Like the original program we need to initialize the length of the variable that is going to capture the list of unique tokens.

❻ We can use the array logic, similar to Program 2. However, notice the use of '[ ]' for arrays within PROC FCMP.

❼ The rest of the code is similar to Program 2.

❽ The RETURN statement is used to return a single value. In this case the single value is the de-duped and sorted list of variables stored in the SORTLIST variable.

Once the function is compiled, then it can be used like any other SAS function (see Program 4). Note that if the function is compiled and saved and you are starting a new SAS session, you need to include ❶ and ❷ from Program 3. The function yields the desired results as shown in Table 3.

```
data want_UDF (keep = SUBJID string newstring);
   set have;
   newstring = sortvals(string);
run;
```

Program 4: Using the UDF in a DATA Step

SUBJID	NEWSTRING
001	AVAL AVALC BASE BASEC CHG PARAM PARAMCD PARAMN PCHG R2BASE
002	ANRHI ANRLO AVAL AVALC CHG PCHG

Table 3: WANT_UDF Sample Data for Sorting and De-Duping Processed with UDF

WRITE A UDF TO RETURN METADATA ON SAS FORMATS

SAS formats are a vital part of any programmer's repertoire . There are a number of ways to obtain information about SAS formats, including from the CATALOG procedure, FORMAT procedure, SAS Dictionary Tables, and the FMTINFO() function. This information can be used to drive processing, for reporting purposes, for documentation, and more. No one source can provide all the required information for all types of formats. Specifically, the FMTINFO() function only works for SAS-supplied formats. Possibly the best arguments for writing UDFs are to enhance an existing functionality, and to replace a deficient function. To return metadata on ALL SAS formats, not just user-defined formats, a new UDF needs to be written, using PROC FCMP.

All SAS functions draw on existing SAS data sets, SAS subroutines, and SAS metadata. In most cases, this works seamlessly. However, for older SAS data stores (for example, metadata related to SAS formats) or for SAS data stores that may be updated (for example, geographic functions such as ZIPSTATE), this doesn't always work as expected. Older SAS-supplied formats are stored in a variety of data types. The FMTINFO() function retrieves metadata about formats in some of those data types, but not all. Specifically, format descriptions are not created automatically (although stored in catalogs for older SAS-provided formats) so collecting descriptions for formats via the FMTINFO() function doesn't work.

Similarly, if the SASHELP.ZIPCODE file stored in your version of SAS has not been updated, the results from any of the ZIP functions may be incorrect.

In order to create a UDF that will return the description of a user-defined format based on a custom format metadata data set, the following steps must be followed:

- Create or access a user defined format;
- Use PROC CATALOG to add a description to the user-defined format;
- Use PROC FORMAT CNTLOUT option to create a SAS data set from the SAS format catalog with the description;
- Collect format metadata from SAS dictionary tables via SAS Views or SQL procedure;
- Create custom format metadata data sets at the format and format value levels.

BUILD A METADATA DATA SET FOR A FORMAT CATALOG

The first step in building metadata for the format description UDF is to create user-defined format(s). In this example, a specialized date format and a categorical format is created as demonstrated in Program 5 and seen in Table 4.

```
proc format fmtlib;
   value dts '01jan1900'd='Invalid'
             '01jan1940'd='Still in'
             '01jan1960'd='SAS zero'
             other=[mmddyy10.];
   value yndkf 1='Yes'
               2='No'
               8='DK';
run;
```

Program 5: Creating User Defined Formats

FMTNAME	START	END	LABEL
DTS	-21914	-21914	Invalid
DTS	-7305	-7305	Still in
DTS	0	0	SAS zero
DTS	**OTHER**	**OTHER**	MMDDYY10.
YNDKF	1	1	Yes
YNDKF	2	2	No
YNDKF	8	8	DK

Table 4: User-Defined Formats Created via Program 5

Next, we need to add a description to the user-defined format(s) we just created. This is accomplished via PROC CATALOG (Hadden L. , 2015) using a MODIFY statement (refer to Program 6). Although a description could be added to a CNTLOUT format data set, if formats are to be stored in catalogs, it is best to apply the description(s) at the catalog level. Note the use of the ODS OUTPUT statement to create a SAS data set (CR) which contains the format catalog metadata, including descriptions, if any. Note that PROC CATALOG is the ONLY way to add a description for a format in a format catalog, unlike other catalogs such as macro and graphics catalogs, which allow the use of a DES= statement.

```
ods output catalog_random = cr;
proc catalog catalog = work.formats et = format;
      modify yndkf
        (desc = "1=Yes, 2=No, 8=DK, Other missing");
      modify dts
        (desc = "Specialized Date Format");
      contents;
run;
quit;
ods output close;
```

Program 6: Use PROC CATALOG to add a Description to a Catalog Entry

#	Name	Type	Create Date	Modified Date	Description
1	DTS	FORMAT	1/15/2024 22:41	1/15/2024 22:41	Specialized Date Format
2	YNDKF	FORMAT	1/15/2024 22:41	1/15/2024 22:41	1=Yes, 2=No, 8=DK, Other missing

Table 5: Contents of Catalog Modified by Program 6

Once the catalog has been updated with format descriptions, PROC FORMAT CNTLOUT option is used to output a data set that can recreate a format catalog on the fly via the PROC FORMAT CNTLIN option. SAS data sets are much more portable than SAS catalogs, across systems, platforms, etc.

```
proc format library = work cntlout = work.workfmts fmtlib;
run;
```

Program 7: Create a SAS Data Set from a SAS Catalog

LIBNAME	MEMNAME	OBJNAME	FMTNAME	FMTTYPE	SOURCE
WORK	FORMATS	DTS	DTS	F	C
WORK	FORMATS	YNDKF	YNDKF	F	C

Table 6: Selected Variables from Metadata on Format Catalog Derived from SASHELP.VFORMAT

SAS metadata on the format level is obtained from SAS dictionary tables, which are present on both the format and the catalog level. This metadata can be accessed via SAS Views (in the SASHELP folder) or via PROC SQL. The “view” version is demonstrated below in Program 8.

```
data subset_vformat;
   set sashelp.vformat;
   where libname = 'WORK' and memname = 'FORMATS' and
         fmtname IN ('YNDKF','DTS');
run;
```

Program 8: Use SAS Metadata on Catalogs from SAS Views to Obtain Format Specific Information

Using the ODS output data set from PROC CATALOG created in Program 6, the CNTLOUT output data set created in Program 7, and the format dictionary tables output created in Program 8, two custom format related data sets can be constructed as shown in Program 9. The first is on the format level, and contains information from the SAS dictionary tables on formats as well as the SAS PROC CATALOG information created in the CR data set. The second is on the format and format value level, and contains information from the SAS dictionary tables, PROC CATALOG, and PROC FORMAT CNTLOUT.

```
data CustomFormats;
   merge subset_vformat (keep = libname memname objname fmttype)
         cr (keep = objname crdate moddate desc);
   by objname;
run;

data CustomFormatDictValues;
   merge subset_vformat (keep = libname memname objname fmttype)
         cr (keep = objname crdate moddate desc)
         workfmts (keep = fmtname start end label type
                   rename = (fmtname = objname));
   by objname;
run;
```

Program 9: Create Custom Format Metadata Data Sets at the Format and Format Value Levels

LIBNAME	MEMNAME	OBJNAME	FMTTYPE	CRDATE	MODDATE	DESC
WORK	FORMATS	DTS	F	1/15/2024 22:41	1/15/2024 22:41	Specialized Date Format
WORK	FORMATS	YNDKF	F	1/15/2024 22:41	1/15/2024 22:41	1=Yes, 2=No, 8=DK, Other missing

Table 7: Custom Format Metadata Data Set at the Format Level

The output presented in Table 9 uses the less complex format level custom format data set (Table 7). It produces a format description given a variable name. It can be modified to take multiple arguments, and run through a loop to produce robust format documentation – and can be reused for any format catalog.

BUILD A USER DEFINED FUNCTION (UDF) TO RETURN FORMAT DESCRIPTION

A function called FULLDESC is created via PROC FCMP to return a format description including the format name when given a format name. Note the use of the LISTALL which ensures that the LISTCODE, LISTPROG, and LISTSOURCE options are available: LISTCODE shows the compiled program code (difficult to read) in the program listing; LISTPROG provides a somewhat more readable version of the compiled program code in the program listing; and LISTSOURCE specifies that source code statements are reported in the program listing.

```
proc fcmp outlib = work.funcs.fulldesc listall;
    function fulldesc(objname $, desc $) $;
    length description $200;
    description = catx(': ',objname,desc);
    return (description);
endsub;
```

Program 10: Create a User-Defined Function with PROC FCMP

Stmt	Line:Col	Statement as Parsed
0	213:63	**;
0	214:63	*** Example of use of Simple User Defined Function ***;
0	215:63	**;
0	206:05:00	function fulldesc(objname $, desc $) $;
1	207:05:00	length description $200;
1	208:05:00	description = CATX(': ', objname, desc);
2	209:05:00	_fulldesc_ = description;
3	210:01:00	ENDSUB;
4	214:63	*** Example of use of Simple User Defined Function ***;

Table 8: Listing of Complied Program Code from LISTPROG

APPLY THE USER DEFINED FUNCTION (UDF) AND RETURN FORMAT DESCRIPTION

Using the format-level custom metadata data set, we apply the UDF FULLDESC to create the format description, effectively recreating the FMTINFO() function with the "DESC" argument, with the added functionality of working with user-defined formats.

```
options cmplib = (work.funcs);
data fmtinfo411;
   set customformats;
   format_desc = fulldesc(objname,desc);
run;
```

Program 11: Demonstrate the Newly Created UDF FULLDESC

FORMAT_DESC	OBJNAME	DESC
DTS: Specialized Date Format	DTS	Specialized Date Format
YNDKF: 1=Yes, 2=No, 8=DK, Other missing	YNDKF	1=Yes, 2=No, 8=DK, Other missing

Table 9: New User-Defined Function, FULLDESC

CONCLUSION

PROC FCMP allows users to create new functions that can meet business needs, greatly increasing processing efficiency, and also allows users to create modified functions that correct deficiencies or increase efficiency in SAS-provided functions. The examples presented rely largely on DATA step processing, but PROC FCMP has the demonstrated potential for additional techniques, including RUN_MACRO, HASH, and FORMAT. We hope that you will consider making PROC FCMP an integral part of your SAS toolkit.

REFERENCES

Garcia, C. (2016). The Power of the Function Compiler: PROC FCMP. *WUSS*. San Francisco, CA. Retrieved from https://www.lexjansen.com/wuss/2016/118_Final_Paper_PDF.pdf

Hadden, L. (2015). PROC CATALOG, the Wish Book SAS® Procedure. Savannah, GA: SESUG. Retrieved from https://analytics.ncsu.edu/sesug/2015/CC-58.pdf

Hadden, L. S. (2022). Putting the Meta into the Data: Managing Data Processing for a Large Scale CDC Surveillance

Project with SAS. *WUSS.* Burlingame, CA. Retrieved from https://www.lexjansen.com/wuss/2022/WUSS-2022-Paper-20.pdf

Hadden, L. S. (2023). With a View to Make Your Metadata Function(al): Exploring the FMTINFO() Function. (p. 14). San Diego: WUSS. Retrieved from https://www.lexjansen.com/wuss/2023/WUSS-2023-Paper-145.pdf

Hughes, T. M. (2023). Make you Holla' Tikka Masala: Creating User-Defined Informats . *SESUG.* Charleston, NC. Retrieved from https://sesug.org/proceedings/sesug_2023_final_papers/Learning_SAS_I/SESUG2023_Paper_167_Final_PDF.pdf

Hughes, T. M. (2023). Sorting a Bajillion Variables: When SORTC and SORTN Subroutines Have Stopped Satisfying, User-Defined PROC FCMP Subroutines Can Leverage the Hash Object to Reorder Limitless Arrays. San Francisco, CA: PharmaSUG. Retrieved from https://www.lexjansen.com/pharmasug/2023/AP/PharmaSUG-2023-AP-094.pdf

Hughes, T. M. (2024). *PROC FCMP User-Defined Functions: An Introduction to the SAS Function Compiler.* Cary: SAS Institute Inc.

Langston, R. (2017). Finding Out About Formats and Their Attributes. *SAS Global Forum.* Orlando, FL. Retrieved from Finding Out About Formats and Their Attributes

Langston, R. (2017). Finding Out About Formats and Their Attributes. *SAS Global Forum.* Orlando, FL. Retrieved from https://support.sas.com/resources/papers/proceedings17/SAS0209-2017.pdf

Make You Holla' Tikka Masala: Creating User-Defined Informats Using the PROC FORMAT OTHER Option to Call User-Defined FCMP Functions That Facilitate Data Ingestion Data Quality. (2023). *SESUG.* Charlestown, NC. Retrieved from https://sesug.org/proceedings/sesug_2023_final_papers/Learning_SAS_I/SESUG2023_Paper_167_Final_PDF.pdf

SAS Institute Inc. (2023, Dec 11). *FCMP Procedure*. Retrieved Jan 2024, from SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p0urpv7yyzylqsn1g2fycva2bs3n.htm

SAS Institute Inc. (2023, Dec 13). *Most Commonly Used Functions*. Retrieved Dec 2023, from SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lefunctionsref/n1b7dkf9vhuqczn16qfd41j6dd54.htm

Watson, R. J. (2022). *Have a Date with ISO®? Using PROC FCMP to Convert Dates to ISO 8601*. Retrieved from https://tp1210.p3cdn1.secureserver.net/wp-content/uploads/2023/07/ISO8601_FCMP.pdf

Watson, R. J., & Hadden, L. S. (2022). *Functions (and More!) on CALL!* Retrieved from https://tp1210.p3cdn1.secureserver.net/wp-content/uploads/2023/01/Functions_and_More_on_CALL.pdf

CONTACT INFORMATION

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

Richann Jean Watson
DataRich Consulting
richann.watson@datarichconsulting.com
https://datarichconsulting.com/

Louise S. Hadden
Independent Consultant
saslouisehadden@gmail.com