Paper SAS013-2014

Using Base SAS® to Extend the SAS® System

Mark L. Jordan, SAS Institute Inc., Cary, NC

ABSTRACT

This paper demonstrates how to use Base SAS® tools to add functional, reusable extensions to the SAS® system. Learn how to do the following:

- Write user-defined macro functions that can be used inline with any other SAS code.
- Use PROC FCMP to write and store user-defined functions that can be used in other SAS programs.
- Write DS2 user-defined methods and store them in packages for easy reuse in subsequent DS2 programs.

INTRODUCTION

SAS programmers often wish that SAS provided a function or feature that enables us to easily complete routine tasks. If the function we desire is specific to our group or a single company, it is unlikely the feature will become part of Base SAS. But Base SAS provides powerful tools that enable us to extend the SAS system ourselves. Using the SAS macro facility, the FCMP procedure, and the DS2 procedure, we can craft custom blocks of reusable code that are deployed and leveraged just like the native functions and CALL routines built into SAS. This paper focuses on designing and writing reusable code modules that can be called inline with other SAS code.

DESIGN CONSIDERATIONS

When designing reusable SAS code modules, you want to make sure that your coding techniques and deployment methods make reuse as simple and problem-free as possible. Considerations include the following:

1. The modules should be stored in a permanent storage location, with SAS configured such that they can be called by the user without requiring additional setup code.

   For example, for the SAS installation on my laptop:
   a. Custom macro source programs are saved in a folder in my Windows profile.
   b. Personal functions created with PROC FCMP are stored in my SASUSER library.
   c. DS2 methods in personal packages are also stored in my SASUSER library.
   d. My SAS configuration file is modified to include options making SAS aware of my customizations.

2. Variables used internally by custom functions should be encapsulated so that their values do not inadvertently replace variable values in the main SAS program's Program Data Vector (PDV).
   a. Macros should explicitly declare internal variables as local.
   b. PROC FCMP function parameters are encapsulated by design. At times you might want to write a module that modifies a main SAS program variable value at the call site (such as a Base SAS CALL routine). Main SAS program variable values can be modified at the call site with the OUTARGS statement.
   c. DS2 method parameters are also encapsulated by design. To modify a SAS program variable at the call site, use an IN_OUT parameter.

3. As these modules will be frequently reused, we want to make them as efficient to execute as possible.

For more transparent reuse of custom SAS extensions, the requisite LIBNAME and options statements can be added to the SAS configuration or autoexec files, making your customizations available as seamlessly as the built-in SAS functions. See the 'Customizing Your SAS Session by Using Configuration and Autoexec Files' topic in the SAS companion documentation for your operating system for more information.

THE SAS MACRO FACILITY

The macro facility is a tool designed for extending and customizing SAS. The SAS macro facility enables you to create named character strings or blocks of SAS code. SAS macro is text-based language designed to process and produce text. Macro definitions can be used to produce flexible, self-modifying, data-driven SAS programs.

WHAT DAY IS %TODAY?

The Base SAS DATA step TODAY function is a handy way of looking up today's date. I've frequently wanted to write the current date in a TITLE or FOOTNOTE, to produce a report like this, for example:

Hybrids as of 13JAN2014

Make	Model	Origin	Horsepower
Honda	Civic Hybrid 4dr manual (gas/electric)	Asia	93
Honda	Insight 2dr (gas/electric)	Asia	73
Toyota	Prius 4dr (gas/electric)	Asia	110

Figure 1: Report with Today's Date in the Title

However, the TODAY function does not work in global SAS statements. The SAS macro facility includes an automatic macro variable (SYSDATE), which contains the date on which the SAS system was invoked. However, if your SAS session stays open for extended periods of time, the current date will likely be different from the value stored in SYSDATE. There is no %TODAY function included in the SAS macro language. Fortunately, we can very easily build our own.

The %SYSFUNC and %QSYSFUNC macro functions allow us to execute SAS DATA step functions from the SAS macro facility and return the results as text to the SAS macro facility. The general syntax for these functions is:

%SYSFUNC(*function*(*argument(s)*)<, *format*>)

%QSYSFUNC(*function*(*argument(s)*)<, *format*>)

In most cases I prefer %QSYSFUNC because the results are returned as quoted text and are never inadvertently interpreted as code by the macro facility. Our first iteration might look something like this:

```
%macro today;
   %qsysfunc(today(),date9.)
%mend;
/* Test the macro */
%put Today is %today;
```

The result in the SAS log indicates the macro is working:

```
Today is 13JAN2014
```

Output 1. Output from the %PUT Statement and the New %today Macro Function

This macro would be more useful if we could control the format of the value returned, so let's modify the macro code to include a parameter that allows us to specify the format:

```
%macro today(fmt);
   %if %superq(fmt)= %then %let fmt=date9.;
   %qsysfunc(today(),%superq(fmt))
%mend;
/* Test the macro */
%put Today is %today();
%put Today is %today(worddate.);
```

The result in the SAS log indicates the macro is working, but the leading spaces produced by the WORDDATE format's default width make the results less than pleasing to the eye:

```
Today is 13JAN2014
Today is   January 13, 2014
```

Passing the results to the STRIP function will remove any leading or trailing blanks before returning the result:

```
%macro today(fmt);
   %if %superq(fmt)= %then %let fmt=date9.;
   %qsysfunc(strip(%qsysfunc(today(),%superq(fmt))))
%mend;
```

And these test results look perfect!

```
Today is 13JAN2014
Today is January 13, 2014
```

A well-designed macro function should include parameter validation to avoid unexpected system error messages. Here is an example of a more production-ready version of the %today macro definition, with parameter validation and a self-documenting HELP feature included:

```
%macro today(fmt);
  /************************************************************************
   Created by Mark Jordan - http://go.sas.com/jedi
   Save the macro source code file (today.sas) in the AUTOCALL path.
  ************************************************************************/
   /* Set format to DATE9. default if user did not supply a format */
   %if %superq(fmt)= %then %let fmt=date9.;
   /* If the user requests help, supply documentation in the SAS log */
   %if %qupcase(%qsubstr(%superq(fmt),1,%sysfunc(MIN(5,%length(%superq(fmt))))))
      =!HELP %then
   %do;
         %let MsgType=NOTE;
         %PUT ;
         %PUT &MSGTYPE:  &SYSMACRONAME MACRO &MSGTYPE ***************************;
   %Syntax:
         %PUT ;
         %PUT &MSGTYPE-  SYNTAX: %NRSTR(%%)&sysmacroname%str(%(<format>%));
         %PUT &MSGTYPE-     format: any valid SAS Date format;
         %PUT &MSGTYPE-     If left blank, date9. format is used;
         %PUT &MSGTYPE-     Use !HELP to produce this syntax help in the SAS log;
         %PUT ;
         %PUT &MSGTYPE-  Example: ;
         %PUT &MSGTYPE-  %NRSTR(%%)&sysmacroname%str(%(MMDDYY10.%));
         %PUT &MSGTYPE-  %NRSTR(%%)&sysmacroname%str(%(!HELP%));
         %PUT ;
         %PUT &MSGTYPE-  ****************************************;
         %PUT ;
         %RETURN;
   %end;
   /* Validate FMT is a valid SAS name */
   %if %LENGTH(%qsysfunc(compress(%superq(fmt),_.,AD))) %then
   %do;
      %let MsgType=ERROR;
         %PUT ;
      %PUT &MSGTYPE:  &SYSMACRONAME MACRO &MSGTYPE ***************************;
      %PUT &MSGTYPE-  &FMT is not valid as a SAS format name.;
      %goto Syntax;
   %end;
   /* Produce today's date in the requested format */
   %qsysfunc(strip(%qsysfunc(today(),%superq(fmt))))
%mend;
```

DEPLOYING THE %TODAY MACRO

There are two methods for storing and reusing macro definitions. Both require minor configuration to make the SAS system aware of the new functionality.

Stored Compiled Macro Facility

A compiled macro is usually saved in the catalog WORK.SASMACR, which is deleted when the SAS session terminates. The stored compiled macro facility enables you to save the compiled code to a catalog in a permanent library that you specify. To store compiled macros in the stored compiled macro facility:

1. Ensure the permanent library in which you will store your compiled macros has been allocated. You can use a pre-defined permanent library such as SASUSER, or you can use a LIBNAME statement to allocate a user-defined permanent storage location.
2. Activate the SAS system option MSTORED.
3. Set the value of SAS system option SASMSTORED= to point to the permanent library libref.
4. Compile the macro using the /STORE option on the %MACRO statement.

Example:

```
libname extend "&path";
options MSTORED SASMSTORE=extend;
%macro today(fmt)/store;
   %if %superq(fmt)= %then %let fmt=date9.;
   %qsysfunc(strip(%qsysfunc(today(),%superq(fmt))))
%mend;
```

To reuse the compiled macros in a subsequent SAS session:

1. Ensure the permanent library in which you will store your compiled macros has been allocated. You can use a pre-defined permanent library such as SASUSER, or you can use a LIBNAME statement to allocate a user-defined permanent storage location.
2. Activate the SAS system option MSTORED.
3. Set the value of SAS system option SASMSTORE= to point to the permanent library libref.

The compiled macros can now be called and executed without requiring access to the source code.

Example:

```
libname extend "&path";
options MSTORED SASMSTORE=extend;

TITLE1 "Hybrids as of %today(worddate.)";
TITLE2 'using %today(worddate.)';
proc print data=extend.cars (where=(type='Hybrid')) noobs;
   var Make Model Origin Horsepower;
run;
```

Hybrids as of January 20, 2014
using %today(worddate.)

Make	Model	Origin	Horsepower
Honda	Civic Hybrid 4dr manual (gas/electric)	Asia	93
Honda	Insight 2dr (gas/electric)	Asia	73
Toyota	Prius 4dr (gas/electric)	Asia	110

Figure 2: Report with Today's Date in the Title Using Stored Compiled Macros

Autocall Macros

SAS is shipped with a host of autocall macros associated with various SAS products. For example, programmers who have used SAS/GRAPH might have used the %helpano macro to provide help for the SAS/GRAPH annotate macros. Without having to know the location of the **`helpano.sas`** source code or take any special step to set up your SAS session, you can submit the code **`%helpano(all)`** The macro definition program file is located and the macro is compiled and executed without any intervention required by the user. This is because the SAS program file containing the macro definition (**`helpano.sas`**) is stored in an autocall library in the autocall path.

A SAS macro autocall library is a directory containing individual SAS program files, each of which contains one macro definition and is named the same as the macro it defines. The autocall path (SASAUTOS) is a concatenation of the locations of the known autocall libraries.

If you add your own autocall library (directory) to the autocall path, your custom macros become available to the SAS system just like the ones that shipped with SAS. For example, let's deploy the %today macro definition as a custom autocall macro in our autocall library directory, **`C:\MyMacros`**.

1. Save the macro definition source code as **`C:\MyMacros\today.sas.`**
2. To use the macros in your autocall library:
   a. Add your autocall library to the autocall path.
   b. Call the macro to execute it.

Example:

```
options SASAUTOS=('C:\MyMacros', SASAUTOS);
%put Today is %today(worddate.);
```

To reuse your autocall macros in a subsequent SAS session:

1. Add your autocall library to the autocall path.
2. Call the macro to execute it.

Example:

```
options SASAUTOS=('C:\MyMacros', SASAUTOS);
TITLE1 "Hybrids as of %today(weekdate.)";
TITLE2 'using %today(weekdate.)';
proc print data=extend.cars (where=(type='Hybrid')) noobs;
   var Make Model Origin Horsepower;
run;
```

Hybrids as of Monday, January 13, 2014
using %today(weekdate.)

Make	Model	Origin	Horsepower
Honda	Civic Hybrid 4dr manual (gas/electric)	Asia	93
Honda	Insight 2dr (gas/electric)	Asia	73
Toyota	Prius 4dr (gas/electric)	Asia	110

Figure 3: Report with Today's Date in the Title Using AUTOCALL

THE SAS FUNCTION COMPILER

The SAS Function Compiler procedure (PROC FCMP) creates SAS functions and CALL routines using syntax that is similar to the DATA step. The functions can then be executed just like the built-in SAS functions and CALL routines in your SAS code. PROC FCMP function and CALL routine definition source code libraries are stored as tables and, if stored in permanent SAS libraries, can easily be reused in subsequent SAS sessions.

HOT OR COLD?

I grew up northeast Brazil and moved to the US permanently at the age of 15. I often struggled with the weather reports up there in Maine. Thirty degrees sounded so nice and warm to me, yet somehow it was freezing outside! The ability to convert between Celsius and Fahrenheit became very important. As a SAS programmer, I have often wished for a Fahrenheit to Celsius conversion function. With PROC FCMP, creating my own function is surprisingly easy.

A quick search of the Internet yields the formulas we'll need:

- $T_c = (5/9)*(T_f-32)$
- $T_f = ((9/5)*T_c)+32$

In the following examples, I'll be using a subset of the SASHELP.HUMID data set, which contains Fahrenheit temperature measurements—both dry bulb (AirTemp) and wet bulb (BulbTemp)—and the calculated relative humidity (Humidity) value.

BUILDING THE CUSTOM FUNCTIONS

The program to create the C2F and F2C functions looks like this:

```
proc fcmp outlib=sasuser.fun.functions;
function c2f(Tc) ;
   return(((Tc*9)/5)+32);
endsub;
function f2c(Tf);
   return((5/9)*(Tf-32));
endsub;
run;
```

You can also use PROC FCMP to list the functions in a library:

```
proc fcmp listfuncs inlib=sasuser.fun.functions;
run;
```

Functions in the SASUSER.FUN.FUNCTIONS library

The FCMP Procedure

FCMP Function/Subroutine Listing			
Name	**Type**	**Returns**	**Prototype**
c2f	FCMP Function	Num	c2f(Tc);
f2c	FCMP Function	Num	f2c(Tf);

Figure 4: List of Functions in the SASHELP.FUN.FUNCTIONS Library

TESTING THE CUSTOM FUNCTIONS

First we need to set the CMPLIB option to let SAS know where to find our custom function library, and then we just use the functions in our code:

```
options cmplib=sasuser.fun;
title 'Fahrenheit converted to Celsius using f2c function';
proc sql;
select AirTemp 'Air Temp (F)'
      ,f2c(AirTemp) 'Air Temp (C)' format=5.1
      ,BulbTemp 'Bulb Temp (F)'
      ,f2c(BulbTemp) 'Bulb Temp (C)' format=5.1
   from extend.humid
;
quit;
```

Fahrenheit converted to Celsius using f2c in PROC SQL

Air Temp (F)	Air Temp (C)	Bulb Temp (F)	Bulb Temp (C)
30	-1.1	5	-15.0
60	15.6	19	-7.2
75	23.9	8	-13.3
75	23.9	27	-2.8
80	26.7	6	-14.4
80	26.7	19	-7.2

Figure 5: Results of F2C Function Executed in PROC SQL

DEPLOYING THE CUSTOM FUNCTIONS

To reuse your custom functions in a subsequent SAS session:

1. Ensure the permanent library containing your custom function library has been allocated.
2. Set the value of SAS system option CMPLIB= to point to the libref containing your function library.

Example:

```
options cmplib=sasuser.fun;
data Celsius1;
  set extend.humid;
  BulbTempC=f2c(BulbTemp);
  AirTempC=f2c(AirTemp);
  format BulbTempC AirTEmpC 5.1;
run;
title 'Fahrenhiet converted to Celsius using f2c function';
proc print data=Celsius1;
   var AirTemp AirTempC BulbTemp BulbTempC;
run;
```

Fahrenhiet converted to Celsius using f2c in DATA Step

Obs	AirTemp	AirTempC	BulbTemp	BulbTempC
1	30	-1.1	5	-15.0
2	60	15.6	19	-7.2
3	75	23.9	8	-13.3
4	75	23.9	27	-2.8
5	80	26.7	6	-14.4
6	80	26.7	19	-7.2

Figure 6: Results of F2C Function Executed in a DATA Step

Knowing that functions can be used in custom format definitions, wouldn't it be fun to deploy this function as a format? Then we could skip the DATA step and just the display the temperatures in Celsius using PROC PRINT.

Example:

```
/*options cmplib=sasuser.fun;*/
proc format;
  value   c2f (default=5) other=[c2f()];
  value   f2c (default=5) other=[f2c()];
run;

title 'Celsius printed directly from Farenheit data using f2c format';
proc print data=extend.humid(obs=10);
  format AirTemp BulbTemp f2c5.1;
run;
```

Celsius printed directly from Farenheit data using f2c format

Obs	BulbTemp	Humidity	AirTemp
1	-15	46	-1.11
2	-7.22	9	15.56
3	-13.3	66	23.89
4	-2.78	4	23.89
5	-14.4	75	26.67
6	-7.22	32	26.67

Figure 7: Results of F2C Function Executed as a Custom Format in PROC PRINT

DS2

The new DS2 SAS proprietary programming language enables programmers to easily create programs that run in parallel using syntax similar to the DATA step. DS2 uses methods to encapsulate all executable code. Besides the three system-defined methods (INIT, RUN and TERM), DS2 makes it easy to write user-defined methods, which can be executed in DS2 DATA steps just like functions and CALL routines are executed in Base SAS DATA steps. DS2 user-defined methods and can be stored in DS2 packages for easy reuse in subsequent SAS sessions.

For the purposes of this paper, all DS2 code is executed using PROC DS2. PROC DS2 is available as part of Base SAS in SAS 9.4.

WHAT'S IN A NAME?

In my programming career, I have frequently encountered name data stored in a single variable but in last name, first name order. The names must be rearranged to conventional first name/last name order for processing. For example, the Names data set contains a variable LFName with people's names in last/first order:

Obs	LFName
1	Abbott, Jr., Raymod G.
2	Baker, Gabriele
3	Campbell, Carston W.
4	Dannin, Freda
5	van Gogh, Vincent Willem
6	Tudor VIII, Henry

Figure 8: LFName Variable Values Are in Last/First Order

Haven't we all wished for a NameFix function at some point in our programming careers? Let's use DS2 and a user-defined method to make that wish come true. Inside the method definition, we'll use the SCAN function with a comma as the only delimiter. Names that contain generational suffixes (for example, Jr.) might contain an additional comma, and we'll take that into account as we build our method.

BUILDING THE USER DEFINED NAMEFIX METHOD

This DS2 program creates and uses the NameFix method to manipulate the data in the LFName variable:

```
proc ds2;
data test (overwrite=yes);
   dcl char(50) Name;
   method NameFix (varchar(50) N) returns varchar(50);
      return
        if countc(N,',') > 1 then
           CATX(' ',scan(N,3,','),CATX(', ',scan(N,1,','),scan(N,2,',')))
        else CATX(' ',scan(N,2,','),scan(N,1,','));
   end;
   method run();
      set names;
      Name=NameFix(LFName);
   end;
enddata;
run;
quit;

title "NameFix Method: Inline Definition";
proc print data=test;
run;
```

NameFix Method: Inline Definition

Obs	Name	LFName
1	Raymod G. Abbott, Jr.	Abbott, Jr., Raymod G.
2	Gabriele Baker	Baker, Gabriele
3	Carston W. Campbell	Campbell, Carston W.
4	Freda Dannin	Dannin, Freda
5	Vincent Willem van Gogh	van Gogh, Vincent Willem
6	Henry Tudor VIII	Tudor VIII, Henry

Figure 9: Results from Executing the NameFix Method Using an Inline Definition

This is all well and good for the current DS2 DATA step, but the code would have to be copied into subsequent programs to reuse the method in its current form. For ease of reuse, we'll want this method stored in a package.

DEPLOYING USER-DEFINED METHODS IN DS2 PACKAGES

DS2 packages are collections of variables and methods stored as encrypted source code in a regular SAS library table. Packages stored in permanent libraries are easily reused in subsequent DS2 programs and threads. The PACKAGE/ENDPACKAGE code block is used to create a package:

```
proc ds2;
package sasuser.ds2_methods /overwrite=yes;
   method NameFix (varchar(50) N) returns varchar(50);
      return
        if countc(N,',') > 1 then
           CATX(' ',scan(N,3,','),CATX(', ',scan(N,1,','),scan(N,2,',')))
        else CATX(' ',scan(N,2,','),scan(N,1,','));
   end;
endpackage;
run;
quit;
```

To reuse user-defined methods in this package in subsequent SAS sessions and DS2 DATA steps:

1. Ensure the permanent library containing your custom function library has been allocated.
2. Declare a named instance of the package in your DS2 DATA step.
3. Call the method using name.method notation.

Example:

```
proc ds2;
data test (overwrite=yes);
   dcl package sasuser.ds2_methods MyFunc();
   dcl char(50) Name;
   method run();
      set names;
      Name=MyFunc.NameFix(LFName);
   end;
enddata;
run;
quit;

title "NameFix Method: From Permanent Package";
proc print data=test;
run;
```

NameFix Method: From Permanent Package

Obs	Name	LFName
1	Raymod G. Abbott, Jr.	Abbott, Jr., Raymod G.
2	Gabriele Baker	Baker, Gabriele
3	Carston W. Campbell	Campbell, Carston W.
4	Freda Dannin	Dannin, Freda
5	Vincent Willem van Gogh	van Gogh, Vincent Willem
6	Henry Tudor VIII	Tudor VIII, Henry

Figure 10: Results from Executing NameFix Method from a DS2 Package

DS2 methods stored in packages can also be used in the SAS FedSQL language. PROC FedSQL is a SAS proprietary implementation of the ANSI SQL:1999 core standard. It supports new data types and other ANSI 1999 core compliance features as well as a few SAS proprietary extensions. PROC FedSQL is scalable, threaded, and high-performance. It was designed to access, manage, and share relational data across multiple data sources. When possible, FedSQL queries are optimized with multi-threaded algorithms.

To use DS2 package methods in FedSQL queries, use library.package.method syntax.

Example:

```
PROC FedSQL;
title "NameFix method: PROC FedSQL";
SELECT LFName
      ,sasuser.ds2_methods.NameFix(LFName) as Name
   from extend.names;
;
quit;
```

NameFix method: PROC FedSQL

LFName	NAME
Abbott, Jr., Raymod G.	Raymod G. Abbott, Jr.
Baker, Gabriele	Gabriele Baker
Campbell, Carston W.	Carston W. Campbell
Dannin, Freda	Freda Dannin
van Gogh, Vincent Willem	Vincent Willem van Gogh
Tudor VIII, Henry	Henry Tudor VIII

Figure 11: Results from Executing the NameFix Method in PROC FedSQL

CONCLUSION

Base SAS includes the SAS macro facility, PROC FCMP, and DS2. These powerful and flexible tools make it easy to extend the capabilities of the SAS system to meet your business requirements.

RECOMMENDED READING

Ellis, Dylan. 2013. "RUN_MACRO Run! With PROC FCMP and the RUN_MACRO Function from SAS® 9.2, Your SAS® Programs Are All Grown Up." *Proceedings of the SAS Global Forum 2013 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings13/033-2013.pdf.

Rhoads, Mike. 2012. "Use the Full Power of SAS® in Your Function-Style Macros." *Proceedings of the SAS Global Forum 2012 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings12/004-2012.pdf.

SAS Technical Documentation, available at http://support.sas.com/documentation/:

- *Base SAS® Procedures Guide*, "*FCMP Procedure"*
- *SAS® 9.4 DS2 Language Reference*
- *SAS® 9.4 FedSQL Language Reference*
- *SAS® 9.4 Macro Language Reference*

The SAS Training Post. Available at http://blogs.sas.com/content/sastraining/:

- Jordan, Mark. "Jedi SAS Tricks: FUNC(y) Formats." Weblog entry. *The SAS Training Post*. 05/10/2012.
- Jordan, Mark. "Jedi SAS Tricks: Roll Your Own Function." Weblog entry. *The SAS Training Post*. 05/03/2012.
- Simon, Jim, "Don't let your macros crash and burn." Weblog entry. *The SAS Training Post*. 12/04/2012.
- Simon, Jim. "Macro programmers: watch out for the Dead Man's Curve!" Weblog entry. *The SAS Training Post*. 10/22/2012.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Mark Jordan
SAS Institute Inc.
100 SAS Campus Drive, H1142
Cary, NC 27513
E-mail: Mark.Jordan@sas.com
Twitter: @SASJedi
Blog: http://go.sas.com/jedi