Schöne neue Welt
So können Sie fehlende SAS-Funktionalitäten mit PROC FCMP nachrüsten

Andreas Menrath
HMS Analytical Software GmbH
Rohrbacher Str. 26
69115 Heidelberg
Andreas.Menrath@analytical-software.de

Zusammenfassung

Schon wieder eine neue SAS Prozedur? Lohnt sich da der Einarbeitungsaufwand?
Die Antwort hierauf ist ein eindeutiges JA!

„*Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.*“, wusste schon Paul Watzlawick. In diesem Sinne bekommt der moderne SAS Programmierer mit PROC FCMP neben der DataStep Programmierung und der Makrosprache ein weiteres mächtiges Werkzeug an die Hand, mit dem sich einige Probleme aus den unterschiedlichsten Problemdomänen elegant lösen lassen.
Dieser Beitrag stellt die wichtigsten Konzepte hinter PROC FCMP vor und zeigt Beispiele auf, wie sich mit wenig Aufwand die Funktionalität von SAS beträchtlich erweitern lässt.

Schlüsselwörter: PROC FCMP, Funktionen erstellen, DLL Bibliotheken einbinden

1 Überblick

Mit PROC FCMP lassen sich ab SAS Version 9.2 auf einfache Weise eigene Funktionen und CALL Routinen selbst programmieren. Bisher konnte man dies nur mit erheblichem Aufwand mittels SAS/Toolkit und viel C Programmierung erreichen.
Komplizierte Berechnungslogik musste man daher entweder als Makro kapseln, oder sich mit LINK-Statements im DataStep behelfen. PROC FCMP bietet hierzu nun eine echte Alternative.

Das Besondere an der Prozedur FCMP ist, dass sie eine eigene Ausführungsumgebung mitbringt. In dieser Umgebung stehen viele mächtige Funktionen bereit, die nach wie vor im DataStep nicht möglich sind [1]. Umgekehrt lassen sich nahezu alle in einem DataStep verwendbaren Funktionen und CALL Routinen auch innerhalb der FCMP Runtime aufrufen [2]. PROC FCMP kann somit als deutliche Bereicherung der DataStep Funktionalität gesehen werden.

Im Verlauf dieses Beitrags wird aufgezeigt, wie man eine eigene Funktion definiert und wie man sie anschließend aufrufen kann. Einige ausgewählte Problemdomänen sollen hier als Beispiel dienen und den Mehrwert der Prozedur verdeutlichen. Hierzu werden die In-Memory-Verarbeitung von Datasets, die verschachtelte Ausführung von SAS

Code und der Zugriff auf Funktionen innerhalb von externen DLL-Bibliotheken betrachtet.

1.1 Syntax

Der einfachste Aufruf der Prozedur enthält Anweisungen, die sofort ausgeführt werden sollen:

```
proc fcmp;
  attrib text length=$15;
  text = cat("Hallo ", "Welt!");
  put text=;
run;
```

Dieser Quellcode definiert eine Variable, verwendet die CAT-Funktion und gibt den Inhalt der Variable aus. Eigentlich alles wie gewohnt in einem normalen DataStep.

Einen wirklichen Mehrwert gewinnt man jedoch erst, wenn man die soeben entwickelte Funktionalität wiederverwendbar machen kann. Hierzu definiert man innerhalb von PROC FCMP eigenständige Funktionen und Subroutinen. Eine Funktion gibt immer genau einen Rückgabewert an den Aufrufer zurück, eine Subroutine entspricht in ihrer Funktionalität dem bekannten Konzept einer CALL Routine und liefert kein direktes Ergebnis zurück, kann aber auf die Variablen, die als Parameter übergeben wurden, zugreifen und deren Werte verändern.
Da im o.g. Beispiel kein Rückgabewert benötigt wird, kann es als Subroutine mit einem Parameter mit Namen „Param“ und dem Datentyp Text ($) wie folgt umgebaut werden:

```
proc fcmp;
  subroutine greet(param $);
     attrib text length=$15;
     text = cat("Hallo ", param, "!");
     put text=;
  endsub;

  call greet("KSFE");    *** Subroutine aufrufen ***;
run;
```

Einen Nachteil hat dieser Quellcode noch: die Subroutine kann nur innerhalb dieser FCMP Prozedur verwendet werden und noch nicht in einem DataStep oder Makro. Um dies zu ermöglichen muss noch eine Bibliothek für den Funktionscompiler angegeben werden.
Hierzu gibt man PROC FCMP per **OUTLIB**-Option den Namen einer Ausgabetabelle mit (in der Formatierung: *<LIBREF>.<Dataset>.<logischer Gruppenname>*). In der SAS Sitzung muss zusätzlich noch definiert werden, an welchem Ort sich die Definitionen von unbekannten SAS Funktionen befinden: hierzu dient die **CMPLIB** Option.

Trifft der SAS Compiler nun auf eine unbekannte Funktion, schlägt er in dieser Tabelle die Implementierung der unbekannten Funktion nach.

```
proc fcmp outlib=work.functions.sample;
  subroutine greet(param $);
     attrib text length=$30;
     text = cat("Hallo ", param, "!");
     put text=;
  endsub;
run;

options cmplib=work.functions;  *** Bibliothek registrieren ***;

data _null_;
   call greet("aus Datastep");
run;
```

Verallgemeinert erfolgt ein typischer PROC FCMP Aufruf nach diesem Muster:

```
proc fcmp outlib=...;
  subroutine mySub(...);
     ...
  endsub;
  function myFunc(...);
     ...
  endsub;
run;
```

Per OUTLIB Option wird die Ausgabebibliothek angegeben. Im Rumpf von PROC FCMP werden dann beliebig viele Funktionen und Subroutinen angegeben.
Eine Funktion und Subroutine kann keine oder beliebig viele Eingabeparameter erwarten. Der Standardtyp für die Eingabeparameter und den Rückgabewert von Funktionen ist eine Fließkommazahl mit der Länge 8. Für Parameter oder Funktionsrückgabewerte vom Typ Text, muss zusätzlich ein Dollarzeichen ($) hinter dem Parameternamen angegeben werden.

1.2 Aufrufmöglichkeiten

Eigene Erweiterungen lassen sich nach ihrer Definition wie gewöhnliche SAS Funktionen und CALL Routinen im DataStep, in Makros und einigen unterstützten Prozeduren verwenden:

```
/* DataStep */
data _null_;
   Result = myFunc(...);
   call mySub(...);
run;
```

```
/* Makro */
%let result = %sysfunc(myFunc(...));
%syscall mySub(...);

/* Prozeduren */
proc sql;
  SELECT myFunc(...) FROM ...;
quit;
```

Ab SAS Version 9.3 kommt noch eine weitere Aufrufmöglichkeit hinzu: bei selbst erstellten Formaten kann mit dem **OTHER** Statement eine Berechnungsfunktion hinterlegt werden, die den formatierten Ausgabewert liefert. Das folgende Beispiel zeigt, wie ein solches Format definiert wird. Man beachte, dass der Funktion keine Parameter übergeben werden (da der zu formatierende Wert der Funktion automatisch als Parameter übergeben wird):

```
proc format;
   value $myFormat other=[myFunc()];
run;
```

2 Anwendungsmöglichkeiten

Wie bereits erwähnt wurde, bringt PROC FCMP eine ganze Reihe neuer Funktionalitäten mit, die sich in den unterschiedlichsten Szenarien einsetzen lassen. Im Folgenden werden drei Anwendungsszenarien vorgestellt, die den Mehrwert von PROC FCMP verdeutlichen.

2.1 In-Memory Datenverarbeitung

Da die PROC FCMP Ausführungsumgebung nicht an die Einschränkungen eines DataSteps gebunden ist, bieten FCMP Variablen und insbesondere Arrays komplett neue Anwendungsmöglichkeiten. Mit der **READ_ARRAY** Funktion lässt sich der komplette Inhalt eines SAS Datasets in ein zweidimensionales Array einlesen. Auf die Daten kann dann direkt über den Arrayindex zugegriffen werden; die Daten müssen daher nicht (wie in einem DataStep) sequentiell abgearbeitet werden.
FCMP Arrays sind im Gegensatz zu einem DataStep Array auch nicht statisch und können zur Laufzeit mit der **CALL DYNAMIC_ARRAY** Routine nach Belieben vergrößert oder verkleinert werden.
Als Gegenstück zur READ_ARRAY Funktion, schreibt die **WRITE_ARRAY** Funktion ein mehrdimensionales Array wieder zurück in ein Dataset.

Das folgende Beispiel zeigt, wie sich die Funktionalität von PROC TRANSPOSE mit Hilfe der Array-Funktionen In-Memory implementieren lässt:

```
proc fcmp outlib=work.functions.test;
  subroutine pivot(datasetname $);
    /* Arrays definieren */
    array sourcearray[1] /NOSYMBOLS;
```

```
    array targetarray[1] /NOSYMBOLS;

    /* Quelldaten einlesen */
    rc = read_array(datasetname, sourcearray);

    /* Größe des Ausgabearrays dynamisch festlegen*/
    call dynamic_array(targetarray
                      , dim(sourcearray,2) /* Anzahl Spalten */
                      , dim(sourcearray,1) /* Anzahl Zeilen */
                      );

    /* Ausgabearray befüllen */
    do row = 1 to dim(sourcearray,1);
      do column = 1 to dim(sourcearray,2);
        targetarray[column, row] = sourcearray[row, column];
      end;
    end;

    /* Ausgabearray speichern */
    rc = write_array(datasetname, targetarray);
  endsub;
run;
options cmplib=(work.functions);
data test2;
  input var1-var5;
  datalines;
1 2 3 4 5
6 7 8 9 10
;
run;
data _null_;
  call pivot("test2");
run;
proc print data=test2;
run;
```

2.2 SAS Code ausführen

Eine weitere Besonderheit der FCMP Laufzeitumgebung besteht darin, dass sich SAS Code aus Dateien (analog zum %include Statement) und SAS Makros innerhalb von PROC FCMP ausführen lassen. Das mag zwar zunächst nicht besonders aufregend klingen, auf den zweiten Blick hat es diese Funktionalität aber in sich.
Sie kennen vielleicht das Problem von Datengetriebenen Anwendungen, wenn Sie für jede Zeile in einem SAS Dataset eine Aktion durchführen möchten, die aus einem oder mehreren PROC- und/oder DataSteps besteht. Mit den bisherigen Mitteln kann in SAS immer nur ein einziger Data- oder Proc-Schritt aktiv sein. In der FCMP Umgebung jedoch gilt diese Einschränkung nicht!
So lassen sich mit der **RUN_MACRO** Funktion Makros aufrufen, oder mit der **RUN_SASFILE** Funktion gleich der ganze Quelltext innerhalb einer Fileref ausführen.

Kapselt man diese Aufrufe wiederum in einer FCMP Subroutine oder Funktion, kann man Data- und ProcSteps beliebig ineinander schachteln.

Der folgende Quelltext liefert ein lauffähiges Beispiel für dieses Konzept. Es wird ein Makro %countAge definiert, das die Businesslogik darstellen soll und die Häufigkeit des Alters in der SASHELP.CLASS Tabelle berechnet. Dieses Makro soll nun innerhalb eines datengetriebenen DataSteps verwendet werden. Das Makro wird hierzu in einer FCMP Funktion mit RUN_MACRO aufgerufen. Als Parameter werden der Name des Makros und eine Auflistung an FCMP Variablen übergeben.
Die FCMP Ausführungsumgebung legt automatisch neue Makrovariablen mit den Namen der FCMP Variablen an, schreibt den Inhalt der FCMP Variablen in die gleichnamigen Makrovariablen, führt anschließend das Makro aus und schreibt zum Schluss den Inhalt der Makrovariablen wieder in die FCMP Variablen zurück.

```
proc fcmp outlib=work.functions.test;
  function countAge(age);
    attrib result length=8;
    rc = run_macro('countAge', age, result);
    return (result);
  endsub;
run;
options cmplib=(work.functions);

%macro countAge;
  proc sql noprint;
    select count(*) into :counter
    from sashelp.class
    where age = &age.;
  quit;
  %let result = &counter.; /* Ergebnis in Makrovariable
zurückschreiben */
%mend countAge;

data test;
  length age count 8;
  do age=10 to 15;
    count=countAge(age);
    output;
  end;
run;

proc print data=test;
run;
```

Das Ergebnis enthält nun für jede Zeile auch den Rückgabewert der Businesslogik-Funktion. Dies bestätigt, dass der PROC SQL Schritt innerhalb des DataSteps ausgeführt wurde.
Mit den bisherigen SAS Mitteln wäre dieses Verhalten auch nicht mit einem Aufruf der CALL EXECUTE Routine möglich gewesen, da der Quelltext, der CALL EXECUTE übergeben wird, erst nach dem Beenden des aufrufenden DataSteps ausgeführt wird und somit keine Rückgabewerte an den DataStep liefern kann.

Beob.	age	count
1	10	0
2	11	2
3	12	5
4	13	3
5	14	4
6	15	4

Es wird darauf hingewiesen, dass bei einem Makroaufruf per RUN_MACRO Funktion noch einige Feinheiten bezüglich der Parameterübergabe und der Sichtbarkeit der Variablen zu berücksichtigen sind. Diese sind jedoch hinreichend in der Dokumentation [3] beschrieben.
Besonders „tollkühne“ SAS Programmierer können unter SAS 9.3 den gleichen Effekt auch mit den experimentellen Funktionen **DOSUBL** und **DOSUB** erzielen[4]. Die beiden Funktionen können direkt ein Makro bzw. eine Quelltextdatei ausführen, ohne zuvor eine eigene FCMP Funktion/Subroutine implementieren zu müssen. Von einem produktiven Einsatz dieser Funktionen wird jedoch zunächst noch abgeraten!

2.3 Externe Bibliotheken verwenden & Funktionsformate erstellen

SAS ist bereits ein ausgereiftes und mächtiges Werkzeug. Aber leider fehlt für den Projekterfolg hier und da noch eine dringend benötigte Berechnungsfunktion oder ein kritisches Feature. Schnell werden Schätzungen aufgestellt wie lange es wohl dauern würde das fehlende Feature X zu implementieren. Nicht selten kommen dann astronomische Zahlen oder schlicht die Aussage: „*das geht mit SAS nicht*“ zutage.
Doch es geht auch anders: das Zauberwort hierfür heißt **Wiederverwendung**. Wie lange bräuchten Sie etwa, um mit SAS Mitteln eine Sprachausgabe zu realisieren, die ihnen wie ihr Navigationsgerät oder ihr Smartphone einen beliebigen Text vorlesen kann?
Bevor Sie nun mit dem Schätzen loslegen, sollten Sie sich erst einmal Gedanken machen, ob Sie in diesem Fall wirklich das Rad neu erfinden möchten, oder nicht viel lieber auf eine existierende Sprachbibliothek zurückgreifen wollen. Eine solche Sprachbibliothek ist beispielsweise auf jedem modernen Windows PC mit dem .Net Framework bereits vorinstalliert und kann kostenlos verwendet werden[5].

Um den Zugriff auf die Sprachausgabebibliothek zu vereinfachen, wurde eine C++/CLI Bibliothek erstellt, die nur eine einzige Methode mit dem Namen „speak“ enthält und als Parameter einen Textstring entgegennimmt:

```
#include "stdafx.h"
using namespace System;
using namespace System::Speech;
using namespace System::Speech::Synthesis;
```

```
void speak(char* mystring)
{
    String^ clistr = gcnew String(mystring);
    SpeechSynthesizer^ synth = gcnew SpeechSynthesizer;
    synth->SetOutputToDefaultAudioDevice();
    synth->Speak(clistr);
}
```

Mit **PROC PROTO** lassen sich externe Bibliotheken in die laufende SAS Sitzung importieren. Zunächst wird hierbei ein Package definiert, in das die externen Funktionsdefinitionen exportiert werden sollen:

```
proc proto package=work.proto_ds.examples;
  link 'C:\KSFE 2013\SpeakDemo\x64\Release\SapiWrapper.dll';
  void speak(char* x);
run;
```

Das **LINK** Statement erhält den Pfad zu der DLL Datei, die SAS importieren soll. Anschließend folgen die Definitionen der Methoden, die aus der externen Bibliothek importiert werden sollen. Man beachte, dass die Signatur der Methode in C Syntax angegeben werden und identisch mit der Signatur in C Quelltext sein muss.

Damit die Funktion nun auch aus PROC FCMP heraus aufrufen werden kann, muss man die Package-Bibliothek per **INLIB** Option importieren; ansonsten findet PROC FCMP die Funktionsdefinition nicht.

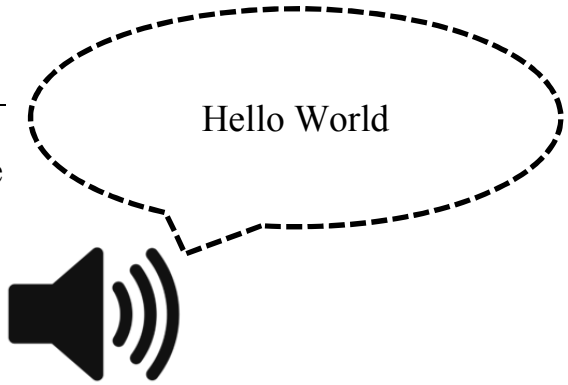
```
proc fcmp inlib=work.proto_ds outlib=work.fcmp_ds.sasfcns;
   subroutine sas_speak(text $);
      length temptext $100;   *** Platzhalter mit fester Breite,
sonst wird nach 32 Zeichen abgeschnitten;
      temptext = text;
      call speak(temptext);
      return;
   endsub;
quit;
```

Das war auch schon alles. Jetzt noch die Funktionsbibliotheken in der SAS Sitzung registrieren und schon kann die neue Funktion verwendet werden:

```
options cmplib=(work.proto_ds work.fcmp_ds);

%let text = Hello World;
%syscall sas_speak(text);
```

Bevor das SAS Log erscheint, begrüßt uns nun eine synthetische Computerstimme.



Als krönenden Abschluss für dieses Beispiel wird der Zugriff auf die Sprachausgabe durch ein benutzerdefiniertes Format vereinfacht. Hierzu wird zunächst die Subroutine in eine neue Funktion umgebaut, damit als Rückgabewert ein Text ausgegeben werden kann. Anschließend wird der Funktionsaufruf durch ein Format mit dem treffenden Namen $AudioFormat gekapselt:

```
proc fcmp inlib=work.proto_ds outlib=work.fcmp_ds.sasfcns;
   function sas_speak(text $) $;
      length temptext $100;   *** Platzhalter mit fester Breite,
sonst wird nach 32 Zeichen abgeschnitten;
      temptext = text;
      call speak(temptext);
      return (temptext);
   endsub;
quit;
options cmplib=(work.proto_ds work.fcmp_ds);
proc format;
   value $AudioFormat
      /* Platzhalter mit Breite $100, sonst wird nach 32 Zeichen
abgeschnitten */
      "_platzhalter_" =
"12345678901234567890123456789012345678901234567890123456789012345678
90123456789012345678901234567890"

      other=[sas_speak()];
run;

*** Format direkt mit put-Funktion verwenden ***;
data _null_;
  x = "Using formats is pretty easy.";
  y = put (x, $AudioFormat.);
run;

*** Format indirekt in ODS Ausgabe verwenden ***;
data test;
  set sashelp.class;
  format name $AudioFormat.;
run;

proc freq data=test;
  table name;
run;
```

Fertig ist das eigene Audioformat. Was als zunächst unmöglich erscheinende Aufgabenstellung begann, lies sich mit relativ geringem Aufwand realisieren.
Es warten nun tausende Klassen im .Net Framework, in OpenSource Projekten oder in firmeninternen Anwendungen darauf von Ihnen entdeckt und verwendet zu werden.

3 Ausblick

Obwohl in diesem Beitrag schon einige der wichtigsten und innovativsten Features von PROC FCMP vorgestellt wurden, war das noch längst nicht alles. Weitere vielversprechende Funktionalitäten (auf die aus Platzgründen nicht weiter eingegangen werden konnte) umfassen:

- Eine Vielzahl neuer Funktionen zur Matrizenrechnung innerhalb von PROC FCMP.
- Berechnung von impliziten Werten mit der **SOLVE** Funktion
- Von SAS bereitgestellte Excel-Funktionen in der Bibliothek **SASHELP.SLKWXL**. Diese SAS Funktionen haben den gleichen Namen, Signatur und Funktionalität wie die entsprechenden Funktionen in Excel.
- Verwaltungsfunktionen für ihre PROC FCMP Funktionsbibliotheken (**LIST** und **DELETE**)
- Die grafische Oberfläche **SAS FCmp Function Editor**, die über den Display Manager gestartet werden kann

Das Studium der SAS Dokumentation für PROC FCMP lohnt sich auf jeden Fall. Mit PROC FCMP stellt SAS nicht einfach nur eine neue Prozedur bereit, sondern führt auch still und leise eine komplett neue Programmierumgebung ein, in der sich viele Aufgaben wesentlich einfacher als bisher realisieren lassen können.

Literatur

[1] Eine Aufstellung aller FCMP spezifischen Funktionen ist unter http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#p0g744alplmu71n1caabv0r3pq7f.htm zu finden.

[2] Die Unterschiede werden unter http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n1aozmc89vjkpzn1q6a54nleh56o.htm beschrieben.

[3] http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n19ylmyrvhp7y6n1889zmwsv9nuh.htm

[4] http://support.sas.com/resources/papers/proceedings12/227-2012.pdf

[5] http://msdn.microsoft.com/en-us/library/ms586901.aspx