# Beyond the Macro – Call Procedures in the Datastep with PROC FCMP

Dr. Christoph Frank
HMS Analytical Software GmbH
Grüne Meile 29
69115 Heidelberg
Christoph.frank@analytical-software.de

**Zusammenfassung**

Mit Hilfe von PROC FCMP kann man benutzerdefinierte Funktionen erstellen, die beispielsweise im SAS Datastep aufgerufen werden können und einen Rückgabewert liefern. Es ist möglich, dass dieser Rückgabewert durch ein von der Funktion aufgerufenes Makro erzeugt wird. Innerhalb einer Datastep-Iteration kann also ein Makro bzw. eine darin verschachtelte Prozedur ausgeführt werden, wobei der entsprechende Prozedur-Rückgabewert noch innerhalb der Datastep-Iteration weiterverarbeitet werden kann.
Im Rahmen dieses Beitrages soll demonstriert werden, wie man dieses Konstrukt nutzen kann, um die Gleichheit von SAS-Datensätzen unterschiedlicher Quellen automatisiert zu überprüfen und eine übersichtliche Darstellung bzw. Zusammenfassung der Ergebnisse zu erzeugen.

**Schlüsselwörter:** SAS Base, SAS MAKRO, PROC FCMP, PROC COMPARE, Return Codes

## 1 Einführung in PROC FCMP

Die SAS Base Prozedur PROC FCMP (Function Compiler) ermöglicht es, benutzerdefinierte Funktionen zu erstellen, die im Datastep oder im PROC SQL Statement aufgerufen werden können. Auf diese Art können beispielsweise spezielle Berechnungen, für die es keine SAS-Standardprozedur gibt, einmalig definiert und dann beliebig oft eingesetzt werden.

```
PROC FCMP OUTLIB=libref.table.package;

    FUNCTION function-name(arguments) <$> <length>;

    . . . programming statements . . .

    RETURN(expression);

    ENDSUB;
RUN;

OPTIONS CMPLIB=libref.table;
```

**Abbildung 1:** Allgemeine Syntax von PROC FCMP

Die allgemeine Syntax ist in Abbildung 1 dargestellt. Die Informationen zur Definition einer benutzerdefinierten Funktion, insbesondere die Berechnungsvorschrift, wird in einem SAS Dataset gespeichert, welcher über die Prozedur-Option *OUTLIB* festgelegt wird. Über die globale Option *CMPLIB* wird der SAS-Dataset angeben, in dem beim Aufruf von benutzerdefinierten Funktionen nach deren Definition gesucht werden soll.
Innerhalb von PROC FCMP beginnt mit dem Statement *FUNCTION* die Definition einer neuen Funktion, welche mit *ENDSUB* abgeschlossen wird. In einem Aufruf von PROC FCMP können mehrere Funktionen definiert werden.
Eine Funktion kann mit numerischen und alphanumerischen Argumenten als Eingabeparameter definiert werden, und kann ihrerseits entweder einen numerischen oder alphanumerischen Rückgabewert liefern. Die Berechnungsvorschrift einer Funktion (*programming statements* in Abbildung 1) wird mittels Datastep-Syntax ausgedrückt.
Für eine ausführlichere Beschreibung von PROC FCMP sei an dieser Stelle auf die SAS-Online-Dokumentation[1] hingewiesen. Daraus sind auch die Beispiele der folgenden Abschnitte 1.1 und 1.2 entnommen.

## 1.1 Erstes einfaches Beispiel

Die Abbildung 2 zeigt die Definition und den Aufruf einer benutzerdefinierten Funktion sowie das entsprechende Ergebnis in einem einfachen Beispiel. Die dabei definierte Funktion *test* hat einen alphanumerischen Rückgabewert der Länge 12 und hängt von einem einzelnen alphanumerischen Eingabewert *x* ab. Hat dieser den Wert *yes,* so lautet der Rückgabewert *si si si*, für alle anderen Eingabewerte wird *no* zurückgegeben. Die Funktionsdefinition ist im Dataset *funcs* in der Bibliothek *work* gespeichert.
Das Ergebnis des Aufrufs *test('yes')* ist im LOG-Ausschnitt rechts in Abbildung 2 hervorgehoben.

[1] Base SAS 9.4 Procedures Guide, Seventh Edition, https://documentation.sas.com/?docsetId=proc&docsetTarget=p10b4qouzgi6sqn154ipglazix2q.htm&docsetVersion=9.4&locale=en

```
proc fcmp
outlib=work.funcs.testfunction;
   function test(x $) $ 12;
   if x='yes' then
      return('si si si');
      else
      return('no');
   endsub;
run;

options cmplib=work.funcs;

data _null_;
   spanish=test('yes');
   put spanish=;
run;
```

LOG:

```
33         proc fcmp outlib=work.funcs.testfunction;
34            function test(x $) $ 12;
35            if x='yes' then
36               return('si si si');
37               else
38               return('no');
39            endsub;
40         run;

NOTE: Function test saved to work.funcs.testfunction.
NOTE:  Verwendet wurde: PROZEDUR FCMP - (Gesamtverarbeitungszeit):
      real time           0.13 seconds
      cpu time            0.04 seconds


41
42         options cmplib=work.funcs;
43
44         data _null_;
45            spanish=test('yes');
46            put spanish=;
47         run;

spanish=si si si
NOTE:  Verwendet wurde: DATA statement - (Gesamtverarbeitungszeit):
      real time           0.10 seconds
      cpu time            0.01 seconds
```

**Abbildung 2:** Definition und Aufruf einer benutzerdefinierten Funktion

## 1.2 Benutzerdefinierte Funktion und Makros

Das folgende Beispiel in Abbildung 3 zeigt, wie eine benutzerdefinierte Funktion ein SAS Makro aufrufen und ausführen kann. Hierbei wird die numerische Funktion *substract_macro* mit den numerischen Parametern *a* und *b* definiert. Innerhalb der Berechnungsvorschrift sind weitere numerische Funktionsvariablen *rc* und *p* implizit definiert.
PROC FCMP bietet über das Statement *run_macro* die Möglichkeit, ein parametrisiertes SAS Makro aufzurufen. Im Beispiel in Abbildung 3 ruft die Funktion *substract_macro* das Makro *testmacro* auf. Beim Anwenden der Funktion wird das Makro dann direkt in einer eigenen SAS-Sitzung im Hintergrund ausgeführt. Dabei werden die Funktionsvariablen *a*, *b* und *p* zu Makrovariablen innerhalb der Makro-Ausführung. Ändern sich deren Werte während der Makroausführung, so werden auch die entsprechenden Funktionsvariablen für die weiteren Berechnungsschritte der Funktion überschrieben.
Der Befehl *run_macro* gibt den Wert 0 zurück, wenn das Makro aufgerufen wurde.

```
/* Create a macro called %TESTMACRO. */
%macro testmacro;
   %let p=%sysevalf(&a - &b);
%mend testmacro;

/* Use %TESTMACRO within a function in PROC FCMP to subtract two numbers. */
proc fcmp outlib=sasuser.ds.functions;
   function subtract_macro(a, b);
      rc=run_macro('testmacro', a, b, p);
      if rc eq 0 then return(p);
      else return(.);
   endsub;
run;

/* Make a call from the DATA step. */
option cmplib=(sasuser.ds);
data _null_;
   a=5.3;
   b=0.7;
   p=.;
   p=subtract_macro(a, b);
   put p=;
run;
```

**Abbildung 3:** Aufruf eines Makros aus einer Funktion heraus

Abbildung 4 zeigt das Ergebnis bzw. den Auszug des Logs vom Programm aus Abbildung 3. Der hier skizzierte einfache Fall mit einem Makro, in welchen lediglich der Wert einer Makrovariable durch eine arithmetische Operation überschrieben wurde, soll im nächsten Abschnitt um eine weitere Komplexitätsstufe erweitert werden und damit das nötige Werkzeug bereitstellen, das anschließend anhand eines Fallbeispiels demonstriert werden soll.

LOG:

```
45
46          /* Make a call from the DATA step. */
47        option cmplib=(sasuser.ds);
48
49        data _null_;
50           a=5.3;
51           b=0.7;
52           p=.;
53           p=subtract_macro(a, b);
54           put p=;
55        run;

p=4.6
NOTE:  Verwendet wurde: DATA statement - (Gesamtverarbeitungszeit):
      real time           0.08 seconds
      cpu time            0.01 seconds
```

**Abbildung 4:** Ergebnis nach Ausführung des Programmes aus Abbildung 3

## 1.3 Ausführung einer Prozedur aus einer Funktion heraus

Die im vorherigen Abschnitt skizzierte Möglichkeit, aus einer benutzerdefinierten Funktion heraus ein Makro auszuführen, kann unter anderem dafür eingesetzt werden, um während einer Datastep-Iteration eine SAS Prozedur auszuführen und deren Ergebnisse noch innerhalb derselben Datastep-Iteration weiterzuverarbeiten. Dieses Konzept ist in Abbildung 5 illustriert.

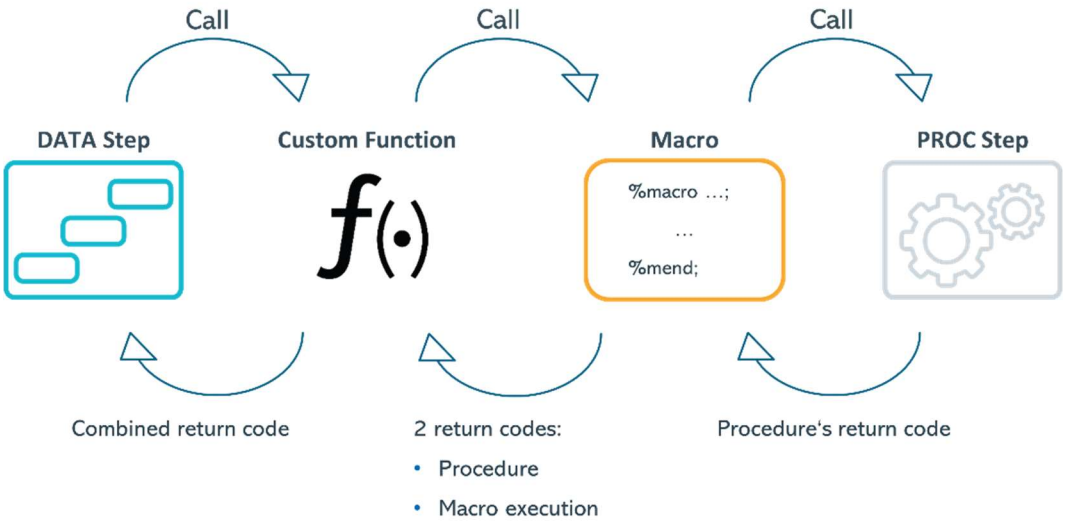


**Abbildung 5:** Verschachtelter Aufruf von Prozeduren im Datastep mit Hilfe von PROC FCMP

Die Ergebnisse der innerhalb des Makros ausgeführten Prozedur oder deren Returncode (Wert der Makrovariable *&SYSINFO* nach der Ausführung einer Prozedur, siehe Abschnitt 2.2) können Makrovariablen zugewiesen werden, die wiederum Funktionsvariablen der benutzerdefinierten Funktion sein können und somit als Rückgabewerte der Funktion noch innerhalb der Data-Iteration zur Verfügung gestellt werden können. Darin liegt der Vorteil bzw. der Mehrwert dieses Konzeptes verglichen mit dem Einsatz von CALL EXECUTE[2], bei dem die Makro-Ausführung im Datastep am Ende einer Iteration stattfindet.

# 2 Anwendungsbeispiel

Das im vorherigen Abschnitt 1.3 vorgestellte Konzept zum Einsatz von PROC FCMP soll nun in einem Anwendungsbeispiel demonstriert werden. Dabei soll für zwei Verzeichnisse überprüft werden, ob die darin gespeicherten SAS-Datasets mit gleichem Namen auch tatsächlich identisch sind. Zum einen soll für alle SAS-Datasets, die in beiden Verzeichnissen enthalten sind, eine übersichtliche Zusammenfassung der etwa-

[2] SAS 9.4 Macro Language: Reference, Fifth Edition, https://documentation.sas.com/?docsetId=mcrolref&docsetTarget=n1q1527d51eivsn1ob5hnz0yd1hx.htm&docsetVersion=9.4&locale=en

igen Unterschiede erstellt werden. Darüber hinaus sollen für jeden paarweisen Vergleich ausführliche Berichte für eine detaillierte Auswertung erzeugt werden.

## 2.1 Generierung von Beispieldaten

Basierend auf der SASHELP Bibliothek, die üblicherweise bei jeder SAS-Installation zur Verfügung steht, wurden zwei SAS-Bibliotheken *BASE_LIB* und *COMP_LIB* angelegt. Diese enthalten zwar gleichnamige Datasets, die Datasets in *COMP_LIB* wurden aber teilweise modifiziert, um bestimmte Unterschiede zu simulieren.

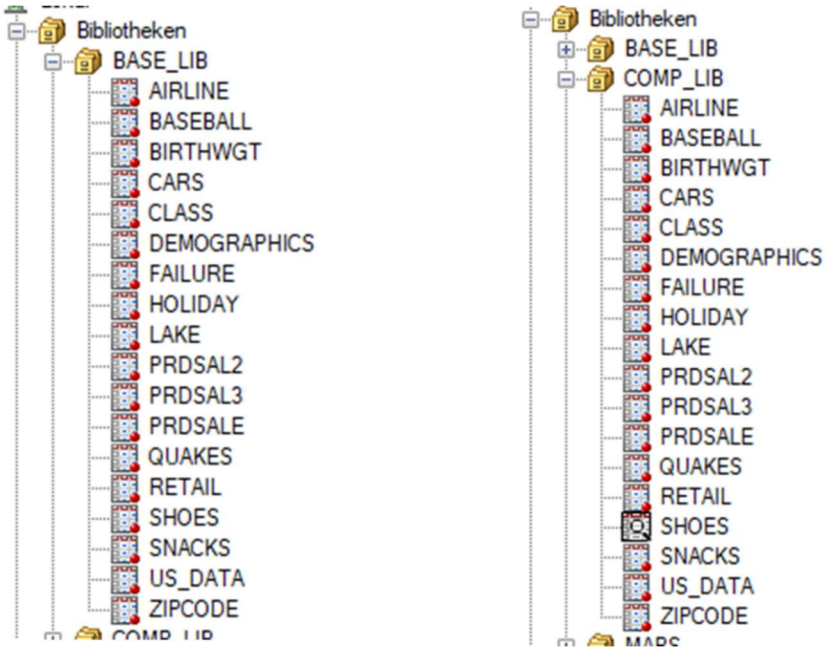


**Abbildung 6:** Zwei SAS-Bibliotheken mit gleichnamigen Datasets, die auf Unterschiede untersucht werden sollen.

Die Datasets *BASE_LIB* sind identische Kopien der SASHELP-Datensätze und die Änderungen an den *COMP_LIB*-Datensätzen wurden durch das folgende SAS-Programm erzeugt:

```
/******* Assign SAS Libs for sample dataset directories **********/

libname base_lib 'U:\KSFE 2020\ds_samples1';
libname comp_lib 'U:\KSFE 2020\ds_samples2';

proc datasets library=base_lib kill;
run;
quit;

proc datasets library=comp_lib kill;
run;
quit;

* create base directory samples ;
proc copy in=SASHELP out=base_lib memtype=data;
```

```
    select  airline baseball birthwgt cars class demographics failure
            holiday lake prdsale prdsal2 prdsal3 quakes retail shoes
            snacks us_data zipcode;
run;

* create Compare directory samples to by modified ;
proc copy in=SASHELP out=comp_lib memtype=data;
    select  airline baseball birthwgt cars class demographics failure
            holiday lake prdsale prdsal2 prdsal3 quakes retail snacks
            us_data zipcode;
run;

/************* Alter datasets: create deviations ***************/

* modify attributes ;
proc datasets library=comp_lib nolist;
     modify baseball (label='New data set label');
     modify prdsal2;
          informat MONYR 8.;
     modify cars;
          format Invoice msrp 15.;
     modify prdsale;
          label actual='New variable label';
quit;

* change variable type;
data comp_lib.airline;
     set comp_lib.airline;
     air_char = strip(put(air,6.));
     drop air;
     rename air_char=air;
run;

* create different type of data set;
proc sql noprint;
     create view comp_lib.shoes
     as select *
     from sashelp.shoes
     ;
quit;

* change variable length;
data comp_lib.prdsal3;
     set comp_lib.prdsal3;
     length year 6.;
run;

* exclude observations ;
data comp_lib.birthwgt;
     set comp_lib.birthwgt;
     if _n_ <= 100;
run;
```

```
* add observation ;
data comp_lib.holiday;
     set comp_lib.holiday end=last;
     output;
     if last then do;
          output;
     end;
run;

* exclude variable;
data comp_lib.class;
     set comp_lib.class;
     drop Age;
run;

* add variable;
* index will get losed;
data comp_lib.zipcode;
     set comp_lib.zipcode;
     new_var = _N_;
run;

* alter variable values ;
data comp_lib.failure;
     set comp_lib.failure;
     if 10 <= _N_ <= 20 then count = count + 1;
run;
```

Die Datasets DEMOGRAPHICS, LAKE, QUAKES, RETAIL, SNACKS und US_DATA sind in beiden Bibliotheken identisch, wohingegen sich die Datasets AIRLINE, BASEBALL, SHOES, FAILURE, BIRTHWGT, CLASS, HOLIDAY, ZIPCODE, PRDSAL3, CARS, PRDSAL2 und PRDSALE auf verschiedene Weise unterscheiden.

## 2.1 Dataset-Vergleiche mit Hilfe von PROC COMPARE

Die Prozedur PROC COMPARE vergleicht zwei SAS-Datasets auf Gleichheit bzgl. ihrer Dateninhalte und ihrer Attribute. Die Ergebnisse des Vergleichs können entsprechend des benutzerdefinierten Detaillierungsgrades in einem SAS-Report ausgegeben werden. Für eine ausführliche Beschreibung der Prozedur wird auch hier auf die SAS-Online-Dokumentation verwiesen[3]. Dort ist unter anderem beschrieben, wie sich die numerischen Rückgabewerte von PROC FCMP gemäß folgender Tabelle 1 zusammensetzen.

[3] Base SAS 9.4 Procedures Guide, Seventh Edition, https://documentation.sas.com/?docsetId=proc&docsetTarget=n0c1y14wyd3u7yn1dmfcpaejllsn.htm&docsetVersion=9.4&locale=en

**Tabelle 1:** Returncodes von PROC COMPARE

| Code | Description |
|---|---|
| 1 | Data set labels differ |
| 2 | Data set types differ |
| 4 | Variable has different informat |
| 8 | Variable has different format |
| 16 | Variable has different length |
| 32 | Variable has different label |
| 64 | Base data set has observation not in comparison |
| 128 | Comparison data set has observation not in base |
| 256 | Base data set has BY group not in comparison |
| 512 | Comparison data set has BY group not in base |
| 1024 | Base data set has variable not in comparison |
| 2048 | Comparison data set has variable not in base |
| 4096 | A value comparison was unequal |
| 8192 | Conflicting variable types |
| 16384 | BY variables do not match |
| 32768 | Fatal error: comparison not done |

Durch die Abstufung der Codes in Zweierpotenzen können diese zu eindeutigen Werten addiert werden, falls verschiedene Arten von Abweichungen bei der Auswertung der Datasets festgestellt werden. Unterscheiden sich die Datasets beispielweise darin, dass für eine Variable unterschiedliche Ausgabeformate vorliegen und dass eine andere Variable in dem einen Dataset als Text und im anderen Dataset als numerischer Variable gefunden wird, so lautet der Rückgabewert von PROC COMPARE: 8+8192=8200. Nach Ausführung von PROC FCMP wird dieser Returncode der automatischen Makrovariable SYSINFO zugewiesen.

## 2.2 SAS-Programm zum automatisierten Vergleich

Das folgende Programm bildet das eingangs beschriebene Anwendungsbeispiel ab. Die Nachvollziehbarkeit der verschachtelten Aufrufe von Funktion, Makro und Prozedur wird durch einen Blick auf Abbildung 5 erleichtert.
Zu Beginn des Programms werden die gleichnamigen Datasets zweier Verzeichnisse bestimmt:

```
*************************************************************
** Assign SAS Libs for both directories and select common datasets *
*************************************************************;

libname base_lib 'U:\KSFE 2020\ds_samples1' access=readonly;
libname comp_lib 'U:\KSFE 2020\ds_samples2' access=readonly;
```

```
proc sql noprint;
     create table base_lib_content as
     select memname from dictionary.tables
     where lowcase(libname)="base_lib"
     order by memname
     ;
     create table comp_lib_content as
     select memname from dictionary.tables
     where lowcase(libname)="comp_lib"
     order by memname
     ;
     create table common_datasets as
     select a.*
     from base_lib_content as a
     inner join comp_lib_content as b
     on a.memname=b.memname
     ;
quit;
```

Die übereinstimmenden Datasets aus *common_datasets* sollen mit Hilfe des Makros *compare()* verglichen werden. Vom Makro wird jeweils ein PROC COMPARE ausgeführt und die Ergebnisse werden in eigene Textdateien umgeleitet:

```
*****************************************************************
** Define function which calls a macro that executes PROC COMPARE **
*****************************************************************;

%macro compare();

     %let compare_ds = %SYSFUNC(DEQUOTE(&in_ds));
     %let rc_compare = %SYSFUNC(DEQUOTE(&rc_compare));

     ***** redirect results *****;
     proc printto print="U:\KSFE
2020\Results\compare_&compare_ds._%sysfunc(date(),date9.)_%sysfunc(t
ime(),B8601TM6.).txt";
     run;

     proc compare base=base_lib.&compare_ds.
                  compare=comp_lib.&compare_ds. maxprint=100;
          title "Comparison of &compare_ds.";
     run;

     ***** store the proc compare return code *****;
     %let rc_compare = &sysinfo.;

     ***** set default log and result output ******;
     proc printto;
     run;

%mend compare;
```

Das Makro *compare()* selbst wird wiederum von der alphanumerischen Funktion *compare_func* aufgerufen. Dabei wird das Makro mit *in_ds* und *rc_compare* parametrisiert. Die Funktionsvariable *in_ds* ist der Eingabewert der Funktion *compare_func* und entspricht dem Namen des Datasets, welcher untersucht werden soll. Dagegen wird die Funktionsvariable *rc_compare* während der Makro-Ausführung mit dem Returncode des PROC COMPARE überschrieben. Letztlich setzt sich der Rückgabewert von *compare_func* als String aus dem Returncode von PROC COMPARE und dem Returncode *rc_macro* des Makroaufrufs zusammen:

```
proc fcmp outlib=work.function.compare_datasets;
     FUNCTION compare_func(in_ds $) $ ;

          ***** variable to store proc compare return code *****;
          rc_compare = .;

          ***** call macro by passing on function variables *****;
          rc_macro = run_macro("compare",in_ds,rc_compare);

          ***** function return value for further processing *****;
          RETURN(cats(rc_compare,',',rc_macro));

     ENDSUB;
run;

options cmplib=work.function;
```

Schließlich wird *compare_func* innerhalb eines Datasteps auf die Datasets aus *common_datasets* angewendet. Das heißt, dass in jeder Datastep-Iteration ein Dataset, der sowohl in *base_lib* als auch in *comp_lib* enthalten ist, mit Hilfe von PROC COMPARE verglichen wird. Die entsprechenden Ergebnisse werden in einzelne Textdateien geschrieben.
Aus dem Rückgabewert der Funktion wird der Returncode von PROC COMPARE extrahiert, in einen numerischen Wert konvertiert und in seine Zweierpotenzen zerlegt. Diese werden gemäß Tabelle 1 übersetzt und in der Variable *result_proc_compare* zusammengefasst:

```
data compare_results;
     length result_proc_compare $200;
     set common_datasets;

     ***** call compare function *****;
     return_codes = compare_func(memname);

     ***** separate the return code *****;
     rc_proc_compare = input(scan(return_codes,1,','),5.);
     rc_macro_call = input(scan(return_codes,2,','),5.);
```

```
     array compare_return_codes(16) _temporary_ (1
                                                 2
                                                 4
                                                 8
                                                 16
                                                 32
                                                 64
                                                 128
                                                 256
                                                 512
                                                 1024
                                                 2048
                                                 4096
                                                 8192
                                                 16384
                                                 32768     );

     array compare_return_code_dc(16) $50 _temporary_ (
               'Data set labels differ.'
          'Data set types differ.'
          'Variable has different informat.'
          'Variable has different format.'
          'Variable has different length.'
          'Variable has different label.'
          'Base data set has observation not in comparison.'
          'Comparison data set has observation not in base.'
          'Base data set has BY group not in comparison.'
          'Comparison data set has BY group not in base.'
          'Base data set has variable not in comparison.'
          'Comparison data set has variable not in base.'
          'A value comparison was unequal.'
          'Conflicting variable types.'
          'BY variables do not match.'
          'Fatal error: comparison not done.');

     if rc_proc_compare = 0 then do;
          result_proc_compare = 'Data sets are equal.';
     end;
     else do;
          ***** decompose the PROC COMPARE return code *****;
          rc_component = rc_proc_compare;
          do i=dim(compare_return_codes) to 1 by -1;
               if rc_component >= compare_return_codes[i] then do;
                    result_proc_compare = catx('
',compare_return_code_dc[i],result_proc_compare);
                    rc_component = rc_component -
compare_return_codes[i];
               end;
          end;
     end;
     drop rc_component i;
run;
```

Der so erstellte Dataset *compare_results* enthält nun eine Zusammenfassung der PROC COMPARE Ergebnisse für alle gleichnamigen Datasets aus den Bibliotheken *base_lib* und *comp_lib* und kann als Übersicht in eine Excel-Datei exportiert werden:

```
************ Export test results overview ***********;

ods excel file="U:\KSFE
2020\Results\compare_sas_datasets_overview_%sysfunc(date(),date9.)_%
sysfunc(time(),B8601TM6.).xlsx";
proc print data=Compare_results noobs label ;
     var memname result_proc_compare rc_proc_compare;
     label memname='Test data set';
     label result_proc_compare='Proc compare result';
     label rc_proc_compare='PROC COMPARE return code';
run;
ods excel close;
```

## 2.3 Ergebnisse

Das SAS-Programm aus Abschnitt 2.3 erstellt für die Beispieldaten aus Abschnitt 2.1 sowohl detaillierte Berichte für jeden einzelnen Vergleich gleichnamiger Datasets als Textdatei (siehe Beispiel in Abbildung 7) als auch eine Zusammenfassung aller Ergebnisse der Vergleichstests (siehe Tabelle 2).
Als Hinweis zu den Ergebnissen in Tabelle 2 sei angemerkt, dass der häufig gefundene Unterschied *Data set labels differ.* darauf zurückzuführen ist, dass einige Modifikationsschritte bei der Generierung der Beispieldaten indirekt auch zu Änderung des Dataset-Labels geführt haben.

Zusammenfassung für Wertevergleich

Anzahl der verglichenen Variablen, bei denen alle Werte gleich sind: 3.
Anzahl der verglichenen Variablen, bei denen einige Werte ungleich sind: 1.
Gesamtanzahl der ungleichen Werte: 11.
Maximale Differenz: 1.

Variablen mit ungleichen Werten

| Variable | Typ | Länge | Ndif | MaxDif |
|---|---|---|---|---|
| Count | NUM | 8 | 11 | 1.000 |

Comparison of FAILURE

Die Prozedur COMPARE
Vergleich von BASE_LIB.FAILURE mit COMP_LIB.FAILURE
(Methode=EXACT)

Ergebnisse des Wertevergleichs für Variablen

| Beob. | Basis Count | Vergleich Count | Diff. | % Diff |
|---|---|---|---|---|
| 10 | 1.0000 | 2.0000 | 1.0000 | 100.0000 |
| 11 | 3.0000 | 4.0000 | 1.0000 | 33.3333 |
| 12 | 1.0000 | 2.0000 | 1.0000 | 100.0000 |
| 13 | 9.0000 | 10.0000 | 1.0000 | 11.1111 |
| 14 | 2.0000 | 3.0000 | 1.0000 | 50.0000 |
| 15 | 20.0000 | 21.0000 | 1.0000 | 5.0000 |
| 16 | 1.0000 | 2.0000 | 1.0000 | 100.0000 |
| 17 | 1.0000 | 2.0000 | 1.0000 | 100.0000 |
| 18 | 0 | 1.0000 | 1.0000 | . |
| 19 | 3.0000 | 4.0000 | 1.0000 | 33.3333 |
| 20 | 7.0000 | 8.0000 | 1.0000 | 14.2857 |

**Abbildung 7:** Auszug aus dem PROC COMPARE Bericht zum Vergleich des Datasets FAILURE

**Tabelle 2:** Zusammenfassung der paarweisen Dataset-Vergleiche

| **Test data set** | **Proc compare result** | **PROC COMPARE return code** |
|---|---|---|
| AIRLINE | Conflicting variable types. | 8192 |
| BASEBALL | Data set labels differ. | 1 |
| BIRTHWGT | Data set labels differ. Base data set has observation not in comparison. | 65 |
| CARS | Variable has different format. | 8 |
| CLASS | Data set labels differ. Base data set has variable not in comparison. | 1025 |
| DEMOGRAPHICS | Data sets are equal. | 0 |
| FAILURE | Data set labels differ. A value comparison was unequal. | 4097 |
| HOLIDAY | Data set labels differ. Comparison data set has observation not in base. | 129 |
| LAKE | Data sets are equal. | 0 |
| PRDSAL2 | Variable has different informat. | 4 |
| PRDSAL3 | Data set labels differ. Variable has different length. | 17 |
| PRDSALE | Variable has different label. | 32 |
| QUAKES | Data sets are equal. | 0 |
| RETAIL | Data sets are equal. | 0 |
| SHOES | Data set labels differ. | 1 |
| SNACKS | Data sets are equal. | 0 |
| US_DATA | Data sets are equal. | 0 |
| ZIPCODE | Data set labels differ. Comparison data set has variable not in base. | 2049 |