Programminhalte extrahieren und modifizieren mit SAS

Katja Glaß
Katja Glass Consulting
12683 Berlin
katja.glass@glacon.eu

Zusammenfassung

Wenn Dokumentationen zu Programmen und Makros verfasst werden müssen, so ist dies für Programmierer doch eine eher langweilige Aufgabe. Letztlich muss doch schon im Programmheader und Quellcode alles Mögliche dokumentiert werden. Um sich die Dokumentation zu vereinfachen, können wesentliche Informationen aus dem Programm mit SAS extrahiert werden. Des Weiteren kann SAS auch verwendet werden um Informationen oder sogar Programmteile in die Programme und Makros automatisiert zu schreiben.
Dieser Beitrag zeigt Techniken anhand eines Fallbeispiels, wo das Extrahieren und Modifizieren von SAS Programmen und Makros eine wesentliche Arbeitserleichterung darstellt. Darüber hinaus werden weitere Anwendungsfälle und Möglichkeiten erläutert.

Schlüsselwörter: Data Step, INFILE, FILE

1 Vorwort

Dokumentationsautomatisierung können Programmierern einiges an lästiger Arbeit abnehmen. Leider werden gerade im behördlich reguliertem Life Science Bereich sehr viele Dokumentationen verlangt. Neben einer Programmspezifikation benötigen Nutzer User-Manuals und das Programm soll eigenständig selbsterklärend sein.
Dies führt dazu, dass die gleichen Informationen mehrfach abgelegt werden. Während solche Dokumentationen bei erstmaliger Erstellung nur lästig, aber doch recht schnell über Kopieren und Einfügen zusammengestellt werden, wird der Aufwand für jedwede weitere Änderung doch immer größer und größer, da nur bestimmte Abschnitte geändert werden.
In diesem Beitrag werden anhand von Beispielen die Möglichkeiten und Implementationen ergründet, die verwendet werden können, um Programminhalte zu extrahieren und Programme automatisiert anzupassen. Hierbei werden SAS Makros verwendet, welche von Scott Bass entwickelt wurden und über die „Unlicense“ Lizenz frei verfügbar sind [1].

2 Dateistrukturanalyse

Zunächst muss die Programmstruktur analysiert werden um zu ermitteln, welche Inhalte extrahiert werden sollen. Die Dateien sind fast alle gleich strukturiert und entsprechen der Beschreibung aus der Datei „@TEMPLATE.sas“. Es gibt einen Header, den Makro-

Teil und einem „Ende“-Tag. Der Header ist dabei für alle Dateien gleich und ist in folgende Bereiche unterteilt:

- Allgemeine Informationen
- Copyright & Lizenz
- Historie
- Anwendung
- Zusätzliche Hinweise

Zunächst wird auf die Extraktion der allgemeinen Informationen eingegangen. Diese sind folgendermaßen gespeichert:

```
/*===================================================================
Program Name             : optsave.sas
Purpose                  : Saves current options settings to a SAS
                           dataset
SAS Version              : SAS 9.1.3
Input Data               : None
Output Data              : SAS options dataset

Makros Called            : parmv

Originally Written by    : Scott Bass
Date                     : 14MAY2010
Program Version #        : 1.0
===================================================================
```

3 Informationsextraktion

3.1 Extrahieren der allgemeinen Header Informationen

Es gibt verschiedene Programmiersprachen, die für die Extraktion von Informationen aus Texten sehr gut geeignet sind. Dazu gehört unter anderem Perl. Wenn man sich jedoch mit SAS oder anderen Programmiersprache wie R auskennt, so kann man auch diese verwenden. Hier wird nur auf SAS eingegangen.

SAS bietet verschiedene Programmierstile und Möglichkeiten, mit Dateien zu arbeiten. Die Basis-Variante verwendet Data-Steps. Data-Steps sind sehr gebräuchlich und strukturiert. Dateien können dabei über INFILE eingelesen werden. Damit die einzelnen Zeilen verfügbar sind, ist ein INPUT notwendig. Da eine Programmdatei nicht aus Spalten besteht, empfiehlt es sich, die komplette Zeile in eine Text Variable für die Weiterverarbeitung zu speichern. Die komplette Zeile kann über die _INFILE_ Anweisung eingelesen werden.

```
%LET path = <path>;

DATA content;
     INFILE "&path/Macro/varlist.sas";
     INPUT;
     line = _INFILE_;
RUN;
```

Um jede Information in einer einzelnen Spalte zu speichern, werden entsprechende Variablen angelegt.

```
ATTRIB line           FORMAT = $255.  LABEL="File content line"
     name             FORMAT = $50.   LABEL="Macro Name"
     purpose          FORMAT = $500.  LABEL="Purpose"
     sas_version      FORMAT = $10.   LABEL="SAS Version"
     input_data       FORMAT = $50.   LABEL="Input Data"
     output_data      FORMAT = $50.   LABEL="Output Data"
     mac_called       FORMAT = $200.  LABEL="Macros Called"
     origin_by        FORMAT = $50.   LABEL="Originally Written By"
     date             FORMAT = $50.   LABEL="Date"
     mac_version      FORMAT = $10.   LABEL="Program Version";
```

Ein Data-Step bearbeitet immer eine Zeile bzw. Observation nach der anderen. Somit gibt es immer nur Zugriff auf die aktuelle Zeile der Datei. Um Informationen innerhalb einer Variablen zur nächsten Observation weiterzureichen, kann RETAIN verwendet werden.

```
RETAIN name purpose sas_version input_data output_data
       mac_called origin_by date;
```

Im so erzeugten Datensatz ist ersichtlich, dass alle Dateiinhalte in die „LINE“ Variable eingelesen wurden. Nun beginnt die Verarbeitung. Reguläre Ausdrücke könnten verwendet werden, um nach bestimmten Mustern zu suchen. Da der Header aber immer gleiche Indikatoren verwendet, kann nach diesen direkt gesucht werden. Hierfür bietet sich die INDEX Funktion an. Um den gewünschten Inhalt zu bekommen, wird der Text extrahiert, der nach dem ersten Doppelpunkt kommt. Dies wird mit der SUBSTR Funktion erreicht.

```
IF      INDEX(line,"Program Name") > 0
     THEN name = SUBSTR(line,INDEX(line,":"));
ELSE IF INDEX(line,"Purpose") > 0
     THEN purpose = SUBSTR(line,INDEX(line,":"));
ELSE IF INDEX(line,"SAS Version") > 0
     THEN sas_version = SUBSTR(line,INDEX(line,":"));
...
```

Wenn der Headerbereich beendet ist, werden die gesammelten Informationen in den Datensatz über OUTPUT ausgeben. Alle weiteren Zeilen werden danach ignoriert, indem der Data-Step über STOP beendet wird.

```
IF _N_ > 5 AND INDEX(line,"================") > 0
THEN DO;
     OUTPUT;
     STOP;
END;
```

Der Inhalt sieht schon sehr gut aus, jedoch werden Zeilenumbrüche noch ignoriert. So ist in dem Beispiel der „Purpose“ auf zwei Zeilen verteilt. Um die zweite Zeile richtig zuordnen zu können, muss sich die inhaltliche Gruppe gemerkt werden.

Tabelle 1: Transposed information of extracted text

Variable	Value
Macro Name	optsave.sas
Purpose	Saves current options settings to a SAS
SAS Version	SAS 9.1.3
Input Data	None
Output Data	SAS options dataset
Macros Called	parmv
Originally Written By	Scott Bass
Date	14MAY2010
Program Version	1.0

Um sich zu merken welches der letzte Parameter war, kann wieder ein RETAIN verwendet werden. In der Bedingung, wo zurzeit der Wert in die korrekte Variable geschrieben wird, soll nun zusätzlich die Variable „Parameter“ auf den entsprechenden Wert gesetzt werden.

```
DATA content;
     ...
     ATTRIB parameter FORMAT = $100. LABEL="Current Parameter";
     RETAIN parameter;

     IF      INDEX(line,"Program Name") > 0
     THEN DO;
          name = STRIP(SUBSTR(line,INDEX(line,":") + 1));
          parameter = "Program Name";
     END;
     ...
RUN;
```

Wenn im Programmablauf eine Zeile auftritt, wo kein Schlüsselwort verwendet wurde, so soll der Inhalt an den entsprechend letzten aktuellen Parameter anhängt werden. Hierbei könnte eine „IF“ Abfrage genutzt werden, jedoch bietet sich ein „SELECT-WHEN“ an, da auf mehrere Werte einer Variablen abgefragt wird.

```
ELSE DO;
      SELECT (STRIP(parameter));
            WHEN ("Program Name")   name = CATX(" ",name,line);
            WHEN ("Purpose")        purpose = CATX(" ",purpose,line);
            WHEN ("SAS Version")    sas_version = CATX(" ",sas_version,line);
            WHEN ("Input Data")     input_data = CATX(" ",input_data,line);
            WHEN ("Output Data")    output_data = CATX(" ",output_data,line);
            WHEN ("Macros Called")  mac_called = CATX(" ",mac_called,line);
            WHEN ("Originally Written by")
                                     origin_by = CATX(" ",origin_by,line);
            WHEN ("Date")             date = CATX(" ",date,line);
            WHEN ("Program Version")
                                  mac_version = CATX(" ",mac_version,line);
            OTHERWISE;
      END;
END;
```

Mit diesen und ähnlichen Methoden können verschiedenste Informationen aus einer Text-Datei extrahiert werden. Wichtige SAS Keywords sind hierbei:

- INFILE, _INFILE_, INPUT zum Einlesen
- INDEX, SUBSTR oder ähnliche Text-Vergleiche zum Erkennen des Elements
    - Reguläre Ausdrücke bieten sich in vielen Beispielen an, auf diese wird hier aber nicht näher eingegangen (RXMATCH, RXPARSE, …)
- STRIP, SUBSTR, SCAN, CATX als Extraktionsmöglichkeiten

3.2 Weitere Beispiele von Informationsextraktion

Neben den allgemeinen Headerinformationen können mit dieser Technik noch viele weitere Informationen aus Macros extrahiert werden.
Im Header befindet sich auch die Modifizierhistorie sowie Beispiele und zusätzliche Hinweise. Darüber hinaus kann das „Crawling“ genutzt werden, um sämtliche im Programm befindlichen Macro Definitionen oder Aufrufe zu ermitteln. Es könnte analysiert werden, welche Macro Variablennamen verwendet wurden und ob diese lokal initialisiert sind. Auch könnten Quellcode Kommentare extrahiert werden, die für die Dokumentation des Programmablaufes genutzt werden können.
Bei SAS Programmen könnten die Methoden verwendet werden, um zu überprüfen, ob und welche Pfade verwendet werden. Es kann analysiert werden, welche Datensätze eingelesen und erstellt wurden. Über spezielle Tags können gezielte Bereiche im Programm erzeugt werden, die für Dokumentationen zusammengesammelt werden können. Zum Beispiel benötigt man in klinischen Studien Informationen für „Analysis Results Metadata“, was Informationen aus den Programmen beinhaltet. Anstatt diese in extra Dokumenten einzutragen, können solche Informationen zusammen mit dem Quellcode gespeichert werden und am Ende extrahiert werden.
Besonders bei Dokumentationen von Macros und Programmen passiert es gelegentlich, dass diese an verschiedenen Stellen gepflegt werden, wodurch leicht Diskrepanzen auftreten.

3.3 Extrahieren aus vielen Dateien

Es ist nützlich und gut, die Informationen einer Datei zu extrahieren, aber viel praktischer ist es, Informationen aus mehreren Dateien zu extrahieren. SAS bietet die Möglichkeit über einen sogenannten PIPE Befehl, mehrere Dateien einzulesen. Noch einfacher geht es allerdings mit sogenannten „Wildcards“. So kann das Sternchen verwendet werden, um mehrere Dateien zu selektieren. In dem Beispiel sollen alle SAS Dateien eingelesen werden, also alle Dateien in der Form „*.sas“. Um unterscheiden zu können, welche Datei gerade eingelesen wird, sollte der Dateiname in eine Variable gespeichert werden.

```
DATA allContent;
     LENGTH filename fname $128;
     INFILE "&path/Macro/*.sas" FILENAME=fname;
     INPUT;
     line = _INFILE_;
     filename = fname;
RUN;
```

Da im Data Step mit RETAIN gearbeitet wird, müssen die Variableninhalte nach jeder gelesenen Datei geleert werden. Hierfür wird ein Indikator benötigt, der angibt, wenn eine neue Datei startet. Hierbei kann die INFILE Option „EOV“ verwendet werden, die auf 1 gesetzt wird, wenn eine neue Datei startet – wobei die erste Datei keine „1“ bewirkt. Da EOV seinen Wert auch behält, muss dieses manuell auf „0“ zurückgesetzt werden, um die nächste neue Datei zu ermitteln.
Als weitere Information soll das Ende des Einlesens erkannt werden, was über die Option END ermittelt wird. Da in der Programmlogik oben auch die aktuelle Zeile der Datei eine Rolle spielt, soll außerdem die aktuelle Dateizeile in einer Variablen zwischengespeichert werden, um das Ende des Headerbereichs weiterhin ermitteln zu können.

```
DATA allContent;
     LENGTH filename fname $128;
     RETAIN lineno 0;
     INFILE "&path/Macro/*.sas" FILENAME=fname EOV=eov END=end;
     INPUT;
     line = _INFILE_;
     filename = fname;
     lineno = lineno + 1;
     IF eov OR _N_=1 THEN DO;
          PUT "Read new file: " filename;
          eov=0;
          lineno = 1;
     END;
RUN;
```

Nun können die beiden Data Steps in einen zusammengefasst werden, um die Informationen aus allen Programmen zu extrahieren. Die Abfrage von _N_ muss lediglich auf die

aktuelle Zeilennummer der aktuellen Datei angepasst werden, also der Variablen „lineno“ geändert werden.
Darüber hinaus wird der „STOP“ Befehl entfernt, da die nächsten Dateien auch eingelesen werden soll. Stattdessen wird eine „ignore“ Variable hinzugefügt, deren Wert über RETAIN erhalten wird. Mit jeder neu eingelesenen Datei wird diese auf „0“ gesetzt und wo „Stop“ ausführen werden sollte, wird der Wert auf „1“ gesetzt. Die Headerinformationsabfrage wird dann in ein IF-Statement gefasst, so dass der Header nur extrahiert wird, wenn der Programmteil nicht ignoriert werden soll (ignore=0).
Um zu verhindern, dass durch RETAIN ggf. alte Informationen behalten werden, können diese einfach über CALL MISSING(<var1>, <var2>, …) geleert werden.
Am Ende können die gesammelten Informationen ausgeben werden um eine schöne Übersicht über alle Macros zu erhalten. Welche Macros gibt es mit welcher Beschreibung und zusätzlichen Informationen? Das Ganze kann dann auch einfach verwendet werden um Dokumentationen in HTML oder Word zu erstellen.

Tabelle 2: Extrahierte Informationen verschiedener SAS-Dateien

Macro Name	SAS Version	Input Data	Output Data	Macros Called	Originally Written By	Date	Program Version
age.sas	SAS 8.2	N/A	N/A	parmv	Scott Bass	30MAY2006	1.0
Purpose: Determines a person's age in <units> based on a reference date							
align_decimals.sas	SAS 9.1.3	N/A	N/A	parmv	Scott Bass	09OCT2010	1.0
Purpose: Aligns the decimal points of numeric or character variables, optionally adding additional bracketing characters (usually parentheses).							
attrib.sas	SAS 9.4	N/A	N/A	parmv	Scott Bass	31MAR2016	1.0
Purpose: Generate attrib statements from a template dataset							
batch_submit.sas	SAS 9.1.3	N/A	N/A	None	Scott Bass	08JUN2011	1.0
Purpose: Submits the current SAS DMS editor session in batch mode, routing the log and print files to the correct output locations.							
bench.sas	SAS 8.2	N/A	N/A	parmv	Scott Bass	24APR2006	1.0
Purpose: Measures elapsed time between successive invocations.							
check_if_empty.sas	SAS 9.4	N/A	N/A	parmv	Scott Bass	04MAR2016	1.0
Purpose: Checks if the source dataset is empty.							

4 Programmteile ersetzen – Header abändern

Neben dem Extrahieren von Informationen aus Programmen und Macros können auch Programmteile automatisiert hinzugefügt, bearbeitet oder gelöscht werden. Als konkretes Beispiel soll der allgemeine Header in den Beispielprogrammen ersetzt werden.

Es soll ein etwas kompakterer Header in folgender Form verwendet werden:

```
%*************************************************************;
%* Project       : Tools Macro Package
%* Program name  : <name>
%* Author        : <author>
%* Date created  : <date>
%* Purpose       : <purpose>
%* Origin        : https://github.com/scottbass/SAS
%*************************************************************;
```

Normalerweise würde man mehr Informationen in den Header aufnehmen, aber dieses Beispiel soll lediglich die Funktionsweise beschreiben.
Der INFILE Befehl wird wieder verwendet, um die Datei einzulesen. Darüber hinaus sollen jedoch auch Informationen aus einem Data Set hinzugefügt werden. Hierbei kann ein SET verwendet werden. Wenn jedoch eine Observation ausgelesen wird und ein INFILE dazukommt, so wird lediglich die erste Zeile der Datei gelesen.
Um eine komplette Datei zu lesen zusammen mit einer Zeile eines Data-Steps, kann eine DO UNTIL Schleife verwendet werden. Mit folgendem Code kann beispielsweise eine komplette Datei gelesen werden und die Variableninformationen aus dem Dataset bleiben erhalten. Im Beispiel beinhaltet „DATA.ALLCONTENT“ die Headerinformationen, die in Abschnitt 3 extrahiert wurden.

```
DATA content;
     SET data.allcontent
          (WHERE=(filename = "&path/Macro/age.sas"));
     ATTRIB line FORMAT = $255. LABEL="File content line";
     INFILE "&path/Macro/age.sas"
            FILENAME=fname EOV=eov END=end;
     DO UNTIL (eof);
          INPUT;
          line = _INFILE_;
          OUTPUT;
     END;
RUN;
```

Über den FILE Befehl in Kombination mit PUT kann eine neue Datei einfach erstellt und geschrieben werden. Jedoch fällt auf, dass führende Leerzeichen verschwinden. Ohne Einrückungsformatierungen sind die Programme schwerer zu lesen. Mit der Positionierung über „@(n)“ kann glücklicherweise eine Einrückung von n Leerzeichen hinzugefügt werden. Um die gewünschte Anzahl der Leerzeichen zu ermitteln kann die „NOTSPACE“ Funktion verwenden werden.

```
FILE "&path/Macro/Mod/age.sas";
PUT @(NOTSPACE(line)) line;
```

Nun wird über PUT Befehle der neuen Header am Anfang der Datei hinzugefügt, der alte Header wird ignoriert und nicht in die neue Datei geschrieben und der Rest wird der neuen Datei ebenso hinzugefügt.

```
DATA _NULL_;
     SET data.allcontent(WHERE=(filename = "&path/Macro/age.sas"));
     ATTRIB line FORMAT = $255. LABEL="File content line";
     INFILE "&path/Macro/age.sas" FILENAME=fname EOV=eov END=eof;
     FILE "&path/Macro/Mod/age.sas";
     * Put new header;
     PUT "%*************************************************;";
     PUT "%* Project       : Tools Macro Package";
     PUT "%* Program name  : " name;
```

```
      PUT "%* Author         : " origin_by;
      PUT "%* Date created   : " date;
      PUT "%* Purpose        : " purpose;
      PUT "%* Origin         : https://github.com/scottbass/SAS";
      PUT "%************************************************;";
      inHead = 1;
      DO UNTIL (eof);
            INPUT;
            line = _INFILE_;
            * skip the old header;
            IF inHead = 0
                  THEN PUT @(NOTSPACE(line)) line;
            ELSE DO;
                  IF INDEX(line,"=======") > 0 AND INDEX(line,"/*") = 0
                  THEN DO;
                        * include a new comment start;
                        inHead = 0;
                        PUT "/*==================================";
                  END;
            END;
      END;
RUN;
```

Da der Dateiinhalt über eine Variable im Data Step verfügbar ist, kann dieser an jeder Stelle modifiziert werden. Es können zusätzliche Zeilen in die Datei über weitere PUT Befehle hinzugefügt werden oder Zeilen werden einfach nicht mit PUT ausgegeben. Somit steht das breite Spektrum der Dateimanipulation zur Verfügung und bietet vielerlei Anwendungsmöglichkeiten.

5 Weitere Möglichkeiten

Das Programm zur Informationsextraktion und Modifizierung kann an vielen Stellen noch angepasst werden. So ist es zum Beispiel beim Update des Headers günstig, den „Purpose“ und ggf. andere Informationen auf verschiedene Zeilen zu teilen. Bei der Informationsextraktion wäre es ebenso günstig, auch Programme zu ermitteln, die den Standard-Header nicht verwenden – auch dies kommt vor.
Um den Header nicht nur in einem Programm zu ändern, sondern in allen, bietet sich an der Stelle auch eher die Macroprogrammierung an, statt diese Funktionsweise über Data Steps zu realisieren. Manchmal kann die Verwendung von Macros übersichtlicher sein, als sehr komplexe Data Steps zu programmieren, die beim Zusammenspiel von SET und INFILE nötig sind.
Sollen Inhalte vor einer bestimmten Zeile eingefügt werden, so kann dies zunächst schwierig anmuten, da SAS klassischerweise Zeile pro Zeile durch einen Datensatz geht. Hier kann sich eines Tricks beholfen werden, indem man einen Datensatz erstellt, die Zeilennummern speichert, den Datensatz mit PROC SORT dreht, die Zeile „davor“, was in dem Fall „dahinter“ ist zu markieren, den Datensatz wieder dreht und dann die Ausgabe entsprechend durchführt.

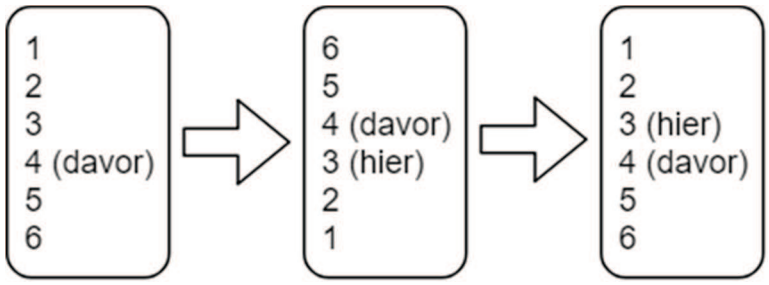

Abbildung 1: Schematische Darstellung, um Inhalte „vor“ etwas hinzuzufügen

6 Einsatzgebiete

Dateimanipulationen können auch mit SAS einfach programmiert werden. Natürlich bieten andere Programmiersprachen wie PERL bessere Optionen, wenn allerdings SAS Programmierer und Programmiererinnen die Manipulation programmieren bzw. verstehen und ändern sollen, so bietet sich SAS an. Dabei sind die Möglichkeiten und Einsatzgebiete sehr vielfältig.
Über das systematische Lesen von Dateien können verschiedene Informationen gesammelt werden:

- Informationen aus dem Header (sofern vorhanden)
  - Wer hat das Programm geschrieben?
  - Worum geht es im Programm?
  - Welche SAS Versionen werden unterstützt?
  - Wie ist der Validierungsstatus?

- Informationen aus den Inhalten
  - Welche Pfade werden im Programm genutzt?
  - Welche SAS Datensätze werden verwendet?
  - Welche Makros werden aufgerufen?
  - Wie viele Zeilen hat das Programm?
  - Befinden sich Kommentare im Programm?
  - Gibt es bestimmte vorgeschriebene Elemente in der Datei?
  - Finde bestimmte Kommentare, die für weitere Dokumentationen verwendet werden müssen (z.B. Analysis Results Metadata)

Diese Informationen könnten dann verwendet werden, um neue Dateien wie Dokumentationen zu erstellen. So könnte sich beispielsweise einen Überblick über Makroparameter in verschiedenen Makros verschafft werden, um diese im Nachhinein sprachlich zu standardisieren.
Es bieten sich eine Vielzahl von Situationen, wo automatisierte Änderungen von Dateien sehr hilfreich sind:

- Änderungen bestimmter Stellen im Programm, wenn von einer SAS Version zu einer anderen migriert wird
- Update von Standardmakroaufrufen mit umbenannten oder neuen Parameternamen

- Programmpfade automatisch ersetzen
- Automatische Header-Anpassung
    - wenn Programmtemplates in Entwicklungsbereiche kopiert werden
    - wenn Programme von A nach B kopiert werden
    - wenn ein Datum/eine Version ersetzt werden muss
- Code Formatierung (wobei sich hier eher andere Tools anbieten)

Darüber hinaus gibt es noch viele weitere Anwendungsgebiete, die den Programmierern manuelle Arbeit abnehmen können. Automatisierte Prozesse sind weniger fehleranfällig und verursachen häufig weniger Diskrepanzen bei Dokumentationen.

7 Fazit

Informationen aus Programmen und Makros zu extrahieren, ist auch mit SAS einfach möglich. Die Möglichkeit, Dateien wie Programme und Makros zu modifizieren, bietet ein breites Spektrum an Optionen. Anhand eines Fallbeispiels konnte gezeigt werden, wie der Programmheader umstrukturiert werden kann - voll automatisch, ohne Informationen zu kopieren und neu einzugeben.
Das Vorgehen und die Implementation für Informationsextraktion und Dateimanipulationen wurden gezeigt, so dass dieses Mittel nun verwendet werden kann, um diverse Aufgaben in dem Bereich automatisieren zu können. Die Beispiele helfen, um Anregungen zu bringen und Prozesserleichterungen zu realisieren.

Literatur

[1] SAS Makros von Scott Bass: https://github.com/scottbass/SAS (Stand: 20.12.2018)