Echt groovy – Neue Sprachen nutzen mit SAS

Sebastian Reimann
viadee Unternehmensberatung GmbH
Anton-Bruchausen-Straße 8
48147 Münster
sebastian.reimann@viadee.de

Zusammenfassung

Mit jedem SAS Release wird der Funktionsumfang der SAS Software um neue Komponenten und Prozeduren erweitert. Doch was kann man mit diesen neuen Funktionen alles machen? Mit dem Data Step Component Object kann man schon seit längerem aus einem Data Step Java Funktionen aufrufen. Diese müssen dafür jedoch als kompilierte JAR-Archive in der Umgebung verfügbar sein.
Mit SAS 9.3 wurde die neue Groovy Prozedur aufgenommen. Damit können Java Programme in der Sprache Groovy zur Laufzeit erstellt, kompiliert und ausgeführt werden. Es soll ein kurzer Einblick in die Nutzung und die damit verbundenen Möglichkeiten gegeben werden. Mit SAS 9.4 wurde mit der Lua Prozedur eine weitere Programmiersprache in SAS integriert. Auch diese Prozedur wird kurz vorgestellt. Weiterhin wird auf die im Vergleich zu Groovy noch weitergehenden Nutzungsmöglichkeiten eingegangen.

Schlüsselwörter: PROC GROOVY, PROC LUA, DataStep Component Interface

1 Motivation

Mit jedem neuen Release erweitert sich der Funktionsumfang der Software SAS. Häufig wird auf diese Neuerungen lediglich in der Aufstellung „What's new" referenziert. Sofern man nicht speziell danach sucht, bemerkt der typische SAS Entwickler gar nicht, dass schon wieder neue Funktionen zur Verfügung stehen.

Aus diesem Grund sollen an dieser Stelle zwei eher technisch geprägte Erweiterungen vorgestellt werden. Die Prozedur PROC GROOVY nutzt die javabasierte Skriptsprache Groovy und stellt hierfür eine Ausführungsumgebung innerhalb der SAS Session zur Verfügung. Die Prozedur PROC LUA integriert die imperative Programmiersprache LUA in die SAS Umgebung und stellt hierfür eine Ausführungsumgebung bereit.

Auf die Unterschiede beider Ansätze wird im Weiteren eingegangen.

2 Groovy

2.1 Was ist Groovy?

Die Programmiersprache Groovy steht als Open Source Produkt unter der Apache Software Foundation zur Verfügung. Erstmals veröffentlicht wurde die Sprache im Jahr 2003; aktuell ist Version 2.4.8 (Januar 2017) verfügbar (vgl. [1]). Maßgeblich beeinflusst wurde Groovy durch die Entwicklungen der Sprachen Python, Ruby und Java. Es wurde das Ziel verfolgt, die Konzepte von Ruby und Python auf die Programmiersprache Java zu übertragen und so eine an Java angelehnte Skriptsprache zu erhalten.

Die Sprache wird in der Java Virtual Machine (JVM) ausgeführt und steht damit auf allen gängigen Plattformen zur Verfügung. Für den Einstieg in diese Programmiersprache stehen verschiedene Online-Entwicklungsumgebungen[1] zur Verfügung, in denen die Sprache einfach getestet werden kann.

Das viel zitierte, einfache „Hello World“ Beispiel kann in Groovy als alleinstehende Codezeile einfach ausgeführt werden:

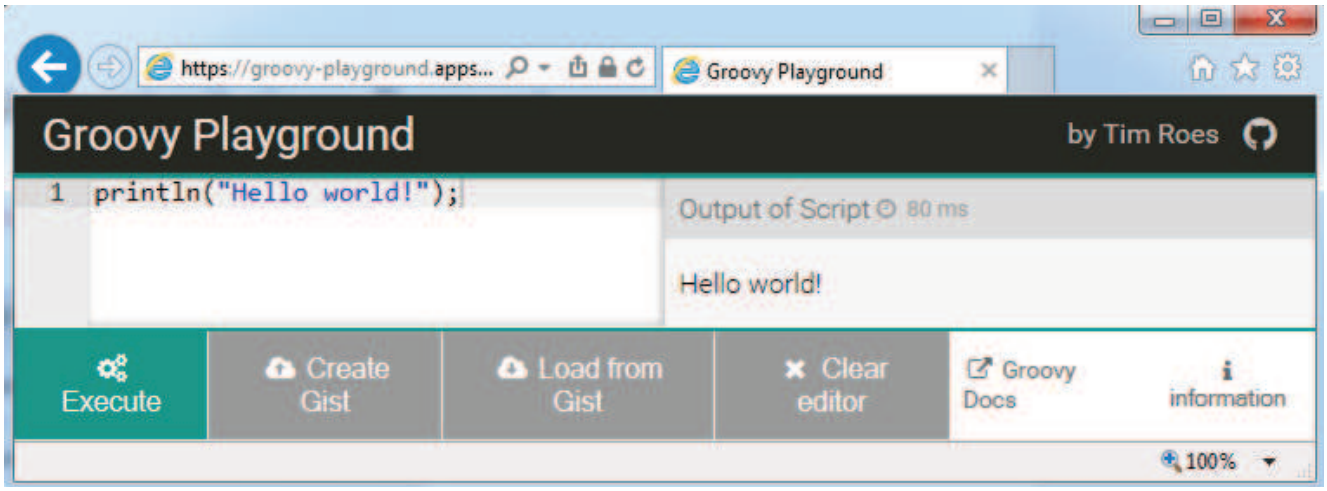

Abbildung 1: Groovy Playground: Hello World Beispiel

2.2 Integration in SAS?

Die Prozedur Groovy wurde mit SAS 9.3 in die SAS Software integriert. Darauf hingewiesen wurde im What’s New Dokument mit folgendem erklärenden Hinweis: „The new GROOVY procedure can run GROOVY statements in your SAS code.“ (vgl. [2]).

Die Ausführung der Groovy Statements erfolgt in der Java Virtual Machin (JVM), welche durch die übergeordnete SAS Session verwaltet wird. Die JVM wird unter dem Userkonto der aufrufenden SAS Session gestartet und hat die gleichen Zugriffsrechte auf das Dateisystem und die Netzwerkumgebung. Die Nutzung ist nur möglich, wenn

[1] z.B. https://groovy-playground.appspot.com/

sie über das Deaktivieren der standardmäßigen Systemoption NOXCMD ermöglicht wurde.

Mit der Groovy Prozedur kann der Classpath der laufenden SAS Session erweitert werden und so zusätzlicher Funktionsumfang für folgende Groovy Aufrufe oder das JAVA DataStep Objekt verfügbar gemacht werden.

Das obige Hello World Beispiel übertragen auf die SAS Integration erfordert folgenden SAS Code:

```
proc groovy;
     submit;
          println("Hello world!");
     endsubmit;
quit;
```

Innerhalb der Prozedur wird ein Submit-Block erstellt, der über die println-Funktion den String „Hello world!“ ausgibt. Dies führt zu folgender Ausgabe:

```
1    proc
1  !      groovy;
2         submit;
3             println("Hello world!");
4         endsubmit;
Hello world!
NOTE: The SUBMIT command completed.
5    quit;

NOTE: PROZEDUR GROOVY used (Total process time):
      real time           0.50 seconds
      cpu time            0.01 seconds
```

Zum Datenaustausch mit der SAS Session gibt es in der Prozedur Groovy vier spezielle Variablen. Über die Variable BINDING können Variablenwerte von einer Groovy Prozedur zu einer anderen übertragen werden. Es werden automatisch alle global erstellten Variablen in die BINDING Map übertragen, so dass die Werte später wieder abgerufen werden können. Alternativ können auch Elemente dieser Map manuell belegt und ausgelesen werden.

Mit Hilfe der ARGS Variable können Argumente übergeben werden. Eine Besonderheit ist, dass das Auflösen von Makrovariablen innerhalb eines Groovy Submit-Blocks nicht stattfindet.

```
%let Nachricht=World;
proc groovy;
     submit;
          println("Hello &Nachricht.!");
     endsubmit;
quit;
```

In diesem Beispiel soll die Makrovariable in der Printausgabe verwendet werden. Obwohl der String in doppelten Anführungszeichen steht, wird die Variable nicht aufgelöst. Stattdessen wird

```
Hello &Nachricht.!
```

ausgegeben. Um dies zu verhindern können dem Submit-Block Argumente übergeben werden, auf die innerhalb des Programms später zugegriffen werden kann.

```
%let Nachricht=World;
proc groovy;
     submit "&Nachricht.";
          println("Hello " + args[0] + "!");
     endsubmit;
quit;
```

In diesem Fall wird die Makrovariable korrekt aufgelöst und innerhalb des Groovy Programms ausgegeben.

```
Hello World!
```

Um Werte aus dem Groovy Programm an die SAS Session zurückzugeben kann die EXPORTS Variable genutzt werden. Hierbei handelt es sich um eine Map mit Schlüssel-Werte-Paaren, die beim Beenden des Groovy Programms als Makrovariablen in die SAS Session übernommen werden können.

2.3 Anwendungsbeispiel

Auf der KSFE 2012 in Dresden wurde vom Autor bereits das DataStep Component Interface vorgestellt. Der dort dargestellte Anwendungsfall zeigte, wie mit reinen SAS Base Mitteln auf relationale Datenbanken zugegriffen werden kann.

Die Basis bildet hierbei das Java Objekt im DataStep Component Interface. Hierüber hat man die Möglichkeit, eigene Java Klassen im DataStep zu nutzen und dort Methoden aufzurufen, die entsprechende String- oder Double Werte aus der Datenbank zurückliefern.

Bei dem damals vorgestellten Ansatz war es erforderlich, in einer separaten Entwicklungsumgebung das entsprechende Java Zugriffsmodul zu erstellen und dieses dann

über den Classpath in der SAS Session verfügbar zu machen. Durch die Groovy Prozedur ist es nun möglich, aus dem SAS Programm heraus die Zugriffsklasse zu generieren und zu nutzen. Die Source Verwaltung für eine weitere Java Entwicklung entfällt komplett; lediglich der JDBC Java Connector zur Datenbank wird im Classpath benötigt.

```
filename jdbc "D:\KSFE\mysql-connector-java-5.1.40-bin.jar";

proc groovy;
 add path=jdbc;
 submit parseonly;

  import java.sql.Connection;
  import java.sql.PreparedStatement;
  import java.sql.ResultSet;
  import java.sql.DriverManager;
  import java.sql.SQLException;
  class MYSQL {

   boolean setStatusForProgram(String program, double status,
                               String host, String database,
                               String user, String password) {
     Connection connect = null;
     PreparedStatement preparedStatement = null;
     ResultSet resultSet = null;
     try {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      connect = DriverManager.getConnection("jdbc:mysql://"
               + host + "/" + database + "?user=" + user
               + "&password=" + password + "&useSSL=false");
      preparedStatement = connect.prepareStatement(
             "SELECT status FROM log WHERE program = ?;");
      preparedStatement.setString(1, program);
      resultSet = preparedStatement.executeQuery();
      if (resultSet.first()) {
        int oldstatus = resultSet.getInt("status");
        resultSet.close();
        preparedStatement = connect.prepareStatement(
              "UPDATE log SET status = ? WHERE program = ?;");
        preparedStatement.setInt(1, new Double(status).intValue());
        preparedStatement.setString(2, program);
        preparedStatement.executeUpdate();
        connect.close();
        return true;
      }
      preparedStatement = connect.prepareStatement(
          "INSERT INTO log (program, status) VALUES (?, ?);");
      preparedStatement.setString(1, program);
      preparedStatement.setInt(2, new Double(status).intValue());
      preparedStatement.executeUpdate();
      connect.close();
      return true;
```

```
      } catch (Exception e) {
        System.out.println(e.toString());
      }
     return false;
    }
 }
 endsubmit;
quit;

data _null_;
  declare javaobj j('MYSQL');
  j.callBooleanMethod("setStatusForProgram", "Prozess1", 4,
           "localhost", "tabelle", "user", "password", ergebnis);
  PUT ergebnis=;
run;
```

Im ersten Schritt wird der Groovy Prozedur der JDBC Treiber für den Zugriff auf die MySQL Datenbank bekannt gemacht. Anschließend wird über den Submit Block die Java Klasse für den Datenbankzugriff kompiliert. Der Zusatz PARSEONLY stellt sicher, dass an der Stelle nur kompiliert wird und das Skript noch nicht ausgeführt wird. Die Java Klasse wurde aus dem Beispiel aus dem Jahre 2012 1:1 übernommen. Im Prinzip wird eine Klasse mit einer Methode bereitgestellt, die einen Boolean Wert zurückliefert und in der Datenbank den Staus für ein Programm aktualisiert. Der Aufruf der Methode erfolgt im DataStep über das Java Objekt. Dort wird die CallBoolean-Methode aufgerufen, die in dem Objekt die Methode setStatusForProgramm aufruft.

Dieses Beispiel ist sicher eine recht spezielle Variante der Nutzung der Groovy Prozedur. Sie zeigt aber wie einfach es ist, individuelle Java-Funktionen zu erstellen und diese dann in der SAS Session weiter zu verwenden.

Ein weiteres, im Internet häufig angeführtes Einsatzgebiet von Groovy ist das Einlesen von JSON Objekten. Mit der Prozedur PROC JSON lassen sich mit SAS leicht JSON Objekte erstellen und anderen Anwendungen bereitstellen, aber das Einlesen von JSON Schnittstellen anderer Programme erweist sich als deutlich komplexer. Hier hilft die Groovy Prozedur, da für Groovy verschiedene Implementierungen für einen JSON Parser verfügbar sind, mit dessen Hilfe sich die JSON Objekte in XML Dokumente verwandeln lassen, so dass diese mit der XML Libname Engine gelesen werden können. Mit dem SAS 9.4 Maintainance Update 4 soll der SAS Sprachumfang jedoch um eine JSON Libname Engine (vgl. [3]) erweitert werden, so dass dann ein nativer Zugriff auf diesen Datentyp besteht.

Außerdem findet man häufig den Anwendungsfall beschrieben, in dem man mit der Groovy Prozedur den Inhalt von ZIP Archiven ermittelt, um selektiv Daten aus dem Archiv zu entpacken und weiterzuverwenden (vgl. [4]). In diesem Fall ist es sehr hilfreich, dass Groovy problemlos Methoden rekursiv aufrufen kann, um Inhalte von Ordnern und Unterordnern zu ermitteln, was mit reinen SAS Funktionen immer schwierig umzusetzen ist.

3 Lua

3.1 Was ist Lua?

Lua ist eine unter der MIT Lizenz verfügbare freie Programmiersprache, die bereits im Jahr 1993 erschienen ist. Besondere Eigenschaft von Lua ist, dass sie sich leicht in andere Programme integrieren lässt. Lua ist plattformunabhängig und besitzt mit LuaJIT einen JIT Compiler, der zur Laufzeit die Skripte in Bytecode übersetzt. Dieser zeichnet sich durch seine geringe Größe aus (vgl. [5]).

Auch für die Sprache Lua steht eine Online-Laufzeitumgebung[2] zur Verfügung, um die Sprache auszuprobieren. Das klassische Hello World Programm kann in Lua mit folgender Code Zeile erstellt werden:

```
print("Hello World!")
```

3.2 Integration in SAS?

Die Prozedur Lua wurde mit SAS 9.4M3 in die SAS Software integriert. Darauf hingewiesen wurde im What's New Dokument mit folgendem erklärenden Hinweis: „Using the LUA procedure, you can run LUA code within a SAS session. The LUA procedure also enables you to call SAS functions from within blocks of LUA code." (vgl. [6]).

Das obige Hello World Beispiel übertragen auf die SAS Integration erfordert folgenden SAS Code:

```
proc lua;
  submit;
    print("Hello World!")
  endsubmit;
quit;
```

Die Ausführung des Codes führt zu folgendem, zu erwartendem Ergebnis:

```
1    proc lua;
2      submit;
3        print("Hello World!")
4      endsubmit;
5    quit;

NOTE: Lua initialized.
Hello world!
NOTE: PROZEDUR LUA used (Total process time):
      real time           0.18 seconds
      cpu time            0.04 seconds
```

[2] z.B. https://www.lua.org/cgi-bin/demo

Auch wenn der Prozeduraufruf vollständig vergleichbar zum Groovy Aufruf ist, stellt sich die dahinter liegende Technik deutlich unterschiedlich dar. Während bei Groovy die Ausführung in der JVM stattfindet, für die eine zur SAS Session korrespondierende Laufzeitumgebung gestartet wird, wird der Lua Code in der ursprünglichen SAS Session ausgeführt. Dies ist möglich, da sowohl SAS als auch Lua in der Programmiersprache C implementiert sind und dies die Integration von Lua in SAS deutlich vereinfacht.

Die von der Prozedur Lua auszuführenden Skripte können wahlweise in eigenen Dateien (mit der Endung .lua bzw. .luc) oder inline im SAS Code abgelegt sein. Um den Ablageort für Lua-Skripte zu definieren muss die Fileref LUAPATH definiert sein, die auf den bzw. die Ordner mit den Lua-Skripten verweist.

Wie auch bei der Groovy Prozedur bleiben Änderungen an der Laufzeitumgebung über mehrere Proc Lua Aufrufe erhalten. Dies bedeutet auch, dass für eine Veränderung des LUAPATH die Laufzeitumgebung neu initialisiert werden muss, damit der geänderte Fileref Wirkung zeigt. Hierzu kann über die Proc Lua Reset Anweisung oder die Proc Lua Terminate Anweisung die Laufzeitumgebung zu Beginn oder zum Ende der Prozedur zurückgesetzt und neu initialisiert werden.

Während bei der Groovy Prozedur der Fokus darauf lag, Groovy Code in der SAS Umgebung ausführbar zu machen, steht bei der Lua Prozedur die Integration in die SAS Umgebung deutlich stärker im Mittelpunkt. Neben der Ausführung von Lua Code können aus Lua heraus die meisten SAS Funktionen und auch in PROC FCMP geschriebene, eigene Funktionen aufgerufen werden. Weiterhin ist es möglich, SAS Code aus der Lua Umgebung heraus aufzurufen und auf in SAS gespeicherte Daten zuzugreifen.

Der Inhalt von Makrovariablen kann mit Hilfe von Übergabevariablen an den Submit Block an die Lua Umgebung übergeben und dort weiterverwendet werden.

```
%let Nachricht=World;
proc lua;
     submit "Nachricht = '&Nachricht.'";
          print("Hello " .. Nachricht .. + "!");
     endsubmit;
quit;
```

Besonders hervorzuheben an der Lua Prozedur ist, dass aus Lua heraus direkt auf den SAS Sprachumfang zugegriffen werden kann. So kann beispielsweise die RANDOM Funktion aufgerufen werden, um eine Zufallszahl zu generieren oder die Datumsfunktion genutzt werden, um das aktuelle SAS Datum zuermitteln:

```
proc lua;
     submit;
          local zufall = sas.rand("uniform");
          local datum = sas.date();
          print("Zufallszahl vom " .. datum .. ": " .. zufall .. "!")
     endsubmit;
quit;
```

Auch das Ausführen von SAS Code aus der Lua Prozedur heraus ist möglich. Hierzu wird die Funktion SAS.SUBMIT genutzt.

```
proc lua;
     submit;
          local eingabe = "sashelp.class";
          sas.submit([[
               data @ergebnis@;
                    set @eingabe@;
               run;
          ]], {ergebnis="work.ausgabe"});
     endsubmit;
quit;
```

Das Beispiel zeigt einen trivialen DataStep. Dennoch wird die ausgefeilte Integration von Lua und SAS deutlich. Zum einen können lokale Lua-Variablen automatisch an den auszuführenden SAS Code übergeben werden, zum anderen können Variablen explizit als Map dem SAS Aufruf übergeben werden. Innerhalb des SAS Codes werden zu ersetzende Variablen durch @-Zeichen markiert. Als Ausgabe wird folgender SAS Code ausgeführt.

```
data work.ausgabe;
   set sashelp.class;
run;
```

Weitere Feinheiten ergeben sich aus der Funktion SAS.SUBMIT_, mit der SAS Code inkrementell erstellt werden kann, ohne dass dieser sofort ausgeführt wird. Die Ausführung geschieht erst, wenn die Funktion SAS.SUMBIT aufgerufen wird.

3.3 Anwendungsbeispiel

Durch den oben beschriebenen Funktionsumfang und die enge Integration mit SAS Code bietet sich Lua als Alternative zur SAS Makrosprache an (vgl. [7]). Gerade wenn SAS Makro für die Steuerung des Ablaufs der SAS Prozeduren genutzt wird, kann die Nutzung von Lua Vorteile bringen. Hier erweist sich die Möglichkeit, in Lua auch auf Datenstrukturen und Tabellen zuzugreifen und direkt Werte aus SAS Tabellen auszulesen als Vorteil gegenüber der SAS Makrosprache.

Als Anwendungsbeispiel wird ein SAS Programm betrachtet, welches den Inhalt einer SAS Tabelle, gruppiert nach einem Feldinhalt, in individuellen Report ausgibt. Dies lässt sich mit einem SAS Makro wie folgt abbilden:

```
%macro genReport;
     %let quelle = sashelp.cars;
     proc sql noprint;
          create table marken as
               select make
                      ,count(*) as anzahl
```

```
                from &quelle.
                group by make;
     quit;
     %let anz=0;
     data _null_;
          set marken end=eof;
          call symput(compress("M_" !! put(_n_, 8.)), strip(make));
          call symput(compress("A_" !! put(_n_, 8.)),
                                strip(put(anzahl, 8.)));
          if eof then call symput("anz", strip(put(_n_, 8.)));
     run;
     %do i=1 %to &anz.;
          title "Ausgabe der Marke &&M_&i.. (&&A_&i.. Datensätze)";
          proc print data=&quelle.(where=(make="&&M_&i.."));
          run;
     %end;
%mend;
%genReport;
```

In einem ersten Schritt werden die Marken der Tabelle SASHELP.CARS ausgelesen und gruppiert. Danach werden alle Marken temporär in Makrovariablen gespeichert. Um die Kontrollstruktur abzubilden, wird die Gesamtzahl der Marken separat abgespeichert, jede Makrovariable ist um einen Laufindex erweitert, so dass über alle Marken iteriert werden kann. Zum Abschluss wird ein PROC PRINT der jeweiligen Marke erstellt.

Gleiches lässt sich mit der Sprache Lua auch abbilden.

```
local quelle="sashelp.cars";
sas.submit([[
     proc sql;
          create table marken as
                select make
                       ,count(*) as anzahl
                from @quelle@
                group by make;
     quit;
]]);

local dsid = sas.open("marken");
for row in sas.rows(dsid) do;
     sas.submit([[
          title "Ausgabe der Marke @make@ (@anzahl@ Datensätze)";
          proc print data=@quelle@(where=(Make="@make@"));
          run;
     ]], {make = row.make, anzahl = row.anzahl});
end;
sas.close(dsid);
```

Im Gegensatz zur Makro-Implementierung lässt sich in der Lua-Implementierung direkt über den Inhalt der Tabelle Marken iterieren. Es müssen keine temporären Makrovariablen gebildet werden. Auch die Lesbarkeit wird erleichtert, da auf die doppelte Makroauflösung (z.B. `&&M_&i..`) verzichtet werden kann.

Wird das Lua Skript als separate Datei gespeichert, kann dies sogar direkt als SYSIN für SAS genutzt werden.

```
sas.exe -sysin genReport.lua
        -log genReport.log
        -print genReport.lst
```

Bei der Ausführung wird durch SAS der Lua Code automatisch in die Lua Prozedur eingesetzt und dort ausgeführt. Auf diese Weise kann der Lua Code auch über einen für Lua optimierten Editor inkl. bspw. Syntax Highlighting entwickelt werden. Der ausgeführte SAS Code sieht wie folgt aus:

```
1   /* Generated PROC LUA stream */
2   proc lua;
3    submit;
4      package.path = "d:\\ksfe\\genReport.lua;"
                      .. package.path
5      require "genReport"
6      package.loaded["genReport"] = nil
7    endsubmit;
8   run;
```

6 Fazit

Mit den Prozeduren Groovy und Lua hat SAS zwei neue Programmiersprachen in die SAS Umgebung integriert. Auch wenn beide Prozeduren vom Aufruf her sehr ähnlich sind, ist deren Umsetzung dennoch grundlegend verschieden.

Mit Groovy liegt der Fokus darauf, den Funktionsumfang durch Java Module zu erweitern und so die Anwendung offener zu gestalten. Es können fertige Java Archive genutzt werden oder es können mit Hilfe des Groovy Compilers Java Klassen zur Laufzeit kompiliert und für den weiteren Programmablauf verfügbar gemacht werden.

Bei der Lua Prozedur liegt der Fokus auf der Optimierung der Programm-Ablaufsteuerung. Die SAS Makro Sprache ist zwar sehr universell und weit verbreitet, bringt jedoch lediglich den Datentyp Text mit sich. Datenstrukturen lassen sich damit nicht abbilden. Dieses wird durch die Lua Integration aufgehoben. Dort können auch komplexere Datenstrukturen in einzelnen Variablen abgebildet werden. Diese Datenstrukturen können dann wiederum zur Steuerung des Programmablaufs genutzt werden. Durch die einfa-

che Programm-Syntax steigt die Lesbarkeit gerade bei komplexeren Kontrollstrukturen erheblich an.

Beide Prozeduren verdienen somit eine detaillierte Betrachtung und können je nach Situation den Umgang mit der SAS Software deutlich vereinfachen.

Literatur

[1] Groovy auf Wikipedia, https://de.wikipedia.org/wiki/Groovy (abgerufen am 20.02.2017)

[2] SAS Institute Inc. – What's New in SAS 9.3, https://support.sas.com/documentation/cdl/en/whatsnew/64209/PDF/default/whatsnew.pdf (abgerufen am 20.02.2017)

[3] Reading Data with SAS JSON libname engine, http://blogs.sas.com/content/sasdummy/2016/12/02/json-libname-engine-sas/ (abgerufen am 20.02.2017)

[4] Paper 493-2013 – Writing a Useful Groovy Programm, http://support.sas.com/resources/papers/proceedings13/493-2013.pdf (abgerufen am 20.02.2017)

[5] Lua auf Wikipedia, https://de.wikipedia.org/wiki/Lua, (abgerufen am 20.02.2017)

[6] SAS Institute Inc. – What's New in SAS 9.4, http://support.sas.com/documentation/cdl/en/whatsnew/64788/PDF/default/whatsnew.pdf (abgerufen am 20.02.2017)

[7] SAS Institute Inc., Paul Tomas – Driving SAS with Lua, Paper SAS 1561-2015 - http://support.sas.com/resources/papers/proceedings15/SAS1561-2015.pdf (abgerufen am 20.02.2017)