Parallele Ausführung von SAS-Programmen und Logfile-Auswertung

Patrik Würzburger
Versicherungskammer Bayern
Maximilianstraße 53
80530 München
patrik.wuerzburger@vkb.de

Zusammenfassung

Normalerweise laufen SAS-Programme Schritt für Schritt und Prozedur für Prozedur sequentiell nacheinander ab. Bauen die Data Steps und Prozeduren nicht aufeinander auf, so wird hierbei die Zeit nicht optimal genutzt. Gerade in der Versicherungswirtschaft gibt es eine Trennung in Leben-, Kranken- und Kompositsparten. Möchte man aus diesen Sparten eine große, übergreifende Analysedatei erstellen, so sind diese Sparten unabhängig voneinander und können in einzelne Programme aufgeteilt und parallel ausgeführt werden. SAS bietet hierzu die Möglichkeit, diese Programme in mehreren Sessions laufen zu lassen. Dabei werden die Logfiles jedoch nicht zentral ausgegeben. Falls man bestehende Programme wegen der Vielzahl an Programmzeilen nicht noch einmal überarbeiten möchte, müsste man die vielen verstreuten Logfiles einzeln nach Fehlermeldungen durchsuchen. Der **Beitrag** zeigt neben der Parallelisierung eine Möglichkeit, wie Logfiles eingelesen und nach den entsprechenden Schlüsselwörtern durchsucht werden können.

Schlüsselwörter: Parallele Ausführung, Logfile, Pipe, Systask, Do-Schleife, SAS Enterprise Guide, Projekt-Log

1 Einleitung

SAS-Programme laufen normalerweise sequentiell ab, es folgt stets Data Step auf Data Step oder Prozedur auf Prozedur. Manche Abläufe lassen sich jedoch auch parallelisieren. Gerade in der Versicherungswirtschaft gibt es häufig eine Trennung in Leben-, Kranken- und Kompositsparten. Um aus diesen Datensilos eine gemeinsame Analysedatei zu erstellen, lassen sich hier die Verarbeitungsschritte in den einzelnen Sparten parallel ausführen.

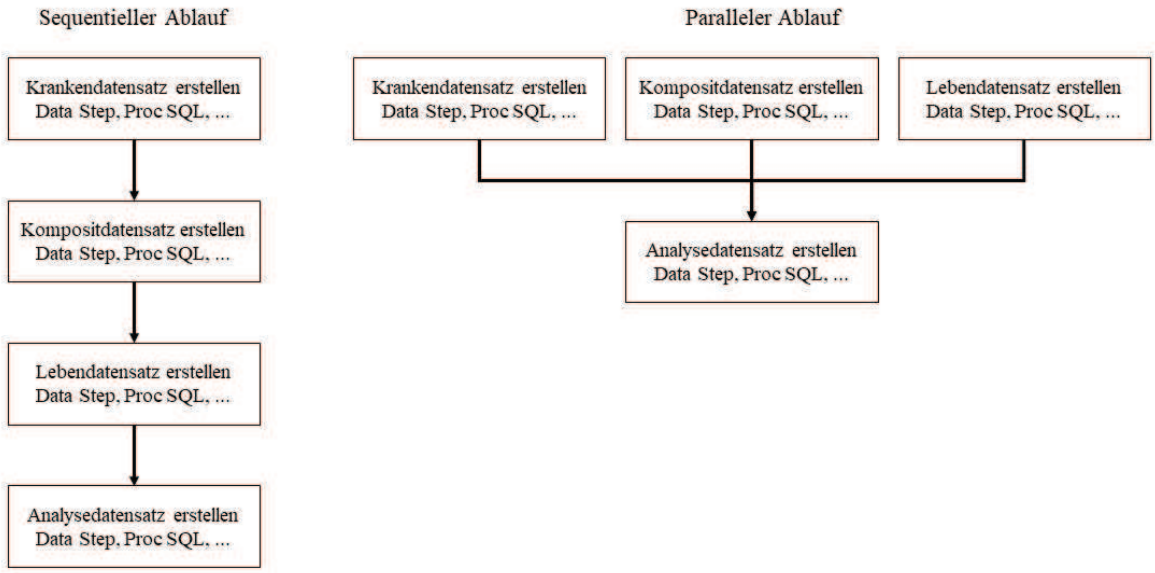

Abbildung 1: Sequentieller und paralleler Ablauf

2 Parallele Ausführung im Display Manager

Der Einfachheit halber gehen wird in der Folge davon ausgegangen, dass für jede Sparte ein eigenes Programm vorhanden ist, das die notwendigen Data Sets komplett erstellt:

- Das Programm „Krankenprogramm.sas“ führt die notwendigen Data Steps und Proc SQL für die Krankenverträge aus
- Für Komposit wird dies von „Kompositprogramm.sas“ ausgeführt
- Und „Lebenprogramm.sas“ erstellt die Data Sets für Leben
- Mit „Analysedatensatz.sas“ werden die Ergebnisse der einzelnen Programme zusammengetragen

Im sequentiellen Ablauf kann ein Ablauf am Ende wie folgt aussehen. In einer Session im SAS Display Manager werden mittels %include die einzelnen Spartenprogramme eingebunden. Sie sind so geschrieben, dass sie lediglich mit ein paar Makroparametern gesteuert werden müssen.

```
%include "C:\TEMP\Code\Krankenprogramm.sas";
%include "C:\TEMP\Code\Kompositprogramm.sas";
%include "C:\TEMP\Code\Lebenprogramm.sas";
%include "C:\TEMP\Code\Analysedatensatz.sas";

%erstelle_kranken (<Makroparameter>);
%erstelle_komposit (<Makroparameter>);
%erstelle_leben (<Makroparameter>);
%erstelle_analysedatensatz (<Makroparameter>);
```

Nacheinander laufen die einzelnen Programme ab. Erst wenn das Krankenprogramm beendet ist, startet das Kompositprogramm und erst nach dessen Lauf beginnt das Lebenprogramm. Sind alle drei Programme fertig, wird aus den Ergebnissen der Analysedatensatz erstellt.

Nun können die Programme für Kranken, Komposit und Leben auch parallel ablaufen. Damit lässt sich eine Zeitersparnis realisieren. Der parallele Ablauf lässt sich mit dem `systask`-Befehl starten.

```
systask command """%sysget(SASROOT)\sas.exe""
""C:\TEMP\Code\Krankenprogramm.sas""
-log ""C:\TEMP\Log\Krankenlog.log"""
taskname=task_kranken status=rc_kranken;

systask command """%sysget(SASROOT)\sas.exe""
""C:\TEMP\Code\Kompositprogramm.sas""
-log ""C:\TEMP\Log\Kompositlog.log"""
taskname=task_komposit status=rc_komposit;

systask command """%sysget(SASROOT)\sas.exe""
""C:\TEMP\Code\Lebenprogramm.sas""
-log ""C:\TEMP\Log\Lebenlog.log"""
taskname=task_leben status=rc_leben;

waitfor _all_ task_kranken task_komposit task_leben;

%erstelle_analysedatensatz (<Makroparameter>);
```

Was passiert nun genau? Es werden drei SAS Session gestartet. In jeder Session wird das entsprechende Programm ausgeführt.
Mit `-log ""C:\TEMP\Log\Krankenlog.log""` wird festgelegt, wohin das Log geschrieben wird. Mit `taskname` wird ein entsprechender Name vergeben. Der `status` enthält einen Returncode, der ausgewertet werden kann. Am Ende sorgt das `waitfor` Statement dafür, dass das Programm erst weiterläuft, wenn alle Data Sets aus Kranken, Komposit und Leben erstellt wurden. Erst dann startet `%erstelle_analysedatei`. Damit wird verhindert, dass es hier zu einem Fehler kommt.

Alternativ kann auch der Returncode im `status` ausgewertet werden. Beispielsweise kann mittels `if then else` geprüft werden, ob die Summe der Returncodes gleich 0 ist und dann erst das Programm zur Erstellung des Analysedatensatzes gestartet werden.

```
data _null_;
  if &rc_kranken + &rc_komposit + &rc_leben eq 0 then do;
    put 'NOTE: Ausführung ohne Fehler beendet';
    %erstelle_analysedatensatz (<Makroparameter>);
  end;
  else do;
    put 'ERROR: Fehler bei der Ausführung';
  end;
run;
```

Eine weitere Fehlerquelle ist das Work-Verzeichnis. Hier ist es besser, die Data Sets in den einzelnen Spartenprogrammen in feste Bibliotheken zu schreiben anstatt Work zu nutzen. Das Work-Verzeichnis in der ersten Session kann ein ganz anderes sein als in der zweiten oder dritten Session.

Gleiches gilt für Makrovariablen. Sollen die einzelnen Programme Makroparameter nutzen, so können diese nicht einfach von Session zu Session übergeben werden. Im Beispiel wäre das ein bestimmter Stichtag für eine `where`-Bedingung, die für alle Bestände, gleich ob Kranken, Komposit oder Leben, gilt. Oder ein Suffix für einen Dateinamen. Dies lässt sich lösen, indem vor dem Aufruf der parallelen Sessions per `systask` noch in einem `data _null_`-Step eine kleine Textdatei geschrieben wird, die die Makrovariablen enthält. Diese Textdatei kann zusätzlich die Pfade für Ausgabebibliotheken enthalten. In den später parallel laufenden Spartenprogrammen wird diese Textdatei mittels `%include` berücksichtigt. Die Textdatei kann entweder aus SAS heraus erstellt werden oder manuell verändert werden. Das ist der Vorteil, diese manuelle Änderbarkeit erlaubt eine Flexibilität.

```
* die notwendigen Makrovariablen ganz normal setzen ;
%let lib='C:\TEMP\SASDS';
%let refdat='08mar2019'd;
%let bestdat=190308;

* Textdatei schreiben ;
data _null_;
  file 'C:\TEMP\Parameter.txt';
  z1=cat('%let lib="' &lib., '";');
  z2= cat('%let refdat=' &refdat., ';');
  z3= cat('%let bestdat="' &bestdat., '";');
  put z1;
  put z2;
  put z3;
run;
```

Die notwenigen Makrovariablen stehen nun in der Textdatei. Jedes Programm, das parallel ausgeführt werden soll und diese Makrovariablen benötigt, braucht dann zu Codebeginn ein `%include`, mit dem die Makrovariablen geladen werden. Alternativ lässt sich dafür auch ein SAS Data Set erstellen, das in den weiteren Programmen mit `set` gelesen wird und mit `call symputx` die Makrovariablen erstellt. Der Autor findet die Lösung mit der Textdatei in der Praxis relevanter, denn der Vorteil liegt darin, dass die Werte auch direkt in der Textdatei geändert werden können. Nachfolgend ist ein Beispiel dafür zu sehen.

```
%include 'C:\TEMP\Parameter.txt';

libname ausgabe &lib.;

data ausgabe.teststocks&bestdat.;
  set sashelp.stocks;
  where date eq &refdat.;
run;
```

3 Auswertung der Logfiles

Es folgt nun der zweite Teil. Die Programme sind parallel durchgelaufen. Jedes hat viele Zeilen im Log erzeugt. Und es können viele Logdateien sein, je nachdem wie die einzelnen Programme aufgebaut sind. Diese Logdateien sollen in der Folge in SAS eingelesen, nach Schlüsselwörtern durchsucht und eine Zusammenfassung ausgegeben werden.

Andere Programmiersprachen können direkt ein Verzeichnis ansprechen und durchlaufen. Zum Beispiel in VBA mit `DIR$`.

```
Dim strData As String

' strDir wird beim Aufruf übergeben.
' Logdateien als Suchziel festlegen.
strData = Dir$(strDir & "\*.log")

' Schleife durch alle Textdateien im Verzeichnis.
Do While strData <> ""
    ' hier der Code...
    ' und die Variable wieder initialisieren.
    strData = Dir$
Loop
```

In SAS geht das nicht direkt, hier muss mittels Pipe das Verzeichnis ausgelesen und in ein Data Set geschrieben werden. Dann kann mittels Proc SQL die Anzahl der Logdateien im Verzeichnis gezählt und in eine Makrovariable geschrieben werden. Gleiches gilt für die Dateinamen. Auch sie werden in eine Makrovariable geschrieben.

```
%macro analysiere_log (verzeichnis=);
/* alle Dateien mit der Endung *.log werden im filename aufgenommen
*/
filename aus pipe "dir ""&verzeichnis""\*.log /B/A:-D";

/* diese Dateien werden dann in ein SAS Dataset geschrieben */
data inhalt;
  length datei $200. ;
  infile aus length=linelen; ;
  input datei $varying500. linelen;
run;

/*  die  Namen  der  Log-Dateien  werden  in  die  Makrovariable
"dateiliste" geschrieben */
/*  die  Anzahl  der  Log-Dateien  wird  in  die  Makrovariable
"dateianzahl" geschrieben */
proc sql noprint;
  select scan(datei, 1) into: dateiliste separated by ' '
  from inhalt;
  select count(datei) into: dateianzahl
```

```
  from inhalt;
quit;
```

Als nächstes werden in einer Schleife die Dateien in der Makrovariable durchlaufen. Mit `infile` und `input` werden sie zeilenweise gelesen. Falls die gewünschten Schlüsselwörter vorkommen, wird die Zeile in das Data Set geschrieben. Dazu wird die Index-Funktion genutzt, die nach einer Zeichenkette sucht und die Stelle zurückgibt, an der die Zeichenkette das erste Mal auftritt.

```
/* mit einer Schleife werden alle Log-Dateien durchlaufen und der
Inhalt, d. h. der Logtext, in eigene SAS Datasets geschrieben */
%do i=1 %to &dateianzahl;
%let saslog=%scan(&dateiliste, &i);

/* der entsprechende filename wird erzeugt */
filename ein "&verzeichnis\&saslog..log";

/*  der  Text  des  i-ten  Logs  wird  in  ein  eigenes  SAS  Dataset
geschrieben und mit index durchsucht */
data log&i.;
  infile ein end=eof length=len;
  length LOGDATEI $50 TYP $15;
  input @1 MELDUNG & $varying200. len;
  logdatei="&saslog..log";

/* von den Notes werden die ausgegeben, die die Anzahl Beobachtungen
und Variablen beinhalten */
  if index(upcase(meldung), 'NOTE:') and
     index(upcase(meldung), 'OBSERVATION') and
     index(upcase(meldung), 'VARIABLE') then do;
    typ='NOTE';
    output;
  end;

/* wenn es statt eines Data Steps ein Proc Sql war */
  if index(upcase(meldung), 'NOTE:') and
     index(upcase(meldung), 'TABLE') and
     index(upcase(meldung), 'CREATED') then do;
    typ='NOTE';
    output;
  end;

/* weitere Notes: Variable is uninitialisized, Missing values were
generated, MERGE statement has more than one data set with repeats
of BY values. */
  if index(upcase(meldung), 'NOTE:') and
     index(upcase(meldung), 'UNINITIALIZED') or
     index(upcase(meldung), 'MISSING VALUES') or
     index(upcase(meldung),  'MERGE  STATEMENT  HAS  MORE  THAN  ONE')
then do;
    typ='NOTE';
```

```
    output;
  end;

/* Warnings werden ausgegeben, Ausnahmen Multiple lengths were
specified, Unable to copy SASUSER und No ouput destination */
  if index(upcase(meldung), 'WARNING:') and not
     index(upcase(meldung), 'WARNING: MULTIPLE LENGTHS') and
     not index(upcase(meldung), 'WARNING: UNABLE') and not
     index(upcase(meldung), 'WARNING: NO OUTPUT') then do;
    typ='WARNING';
    output;
  end;

/* alle Errors werden ausgegeben, Ausnahme ist die Errors Printed-
Zeile */
  if index(upcase(meldung), 'ERROR:') and not
     index(upcase(meldung), 'ERRORS PRINTED') then do;
    typ='ERROR';
    output;
  end;
run;

filename ein clear;
%end;
```

Als Ergebnis entsteht für jede Logdatei ein Data Set mit den relevanten Meldungen. Diese einzelnen Data Sets werden zu einem Data Set zusammengefasst. Das ist die Ergebnisdatei. Mittels Proc Datasets werden alle nicht mehr benötigten Dateien gelöscht.

```
/* die einzelnen SAS Datasets mit den Meldungen werden in einem SAS
Dataset zusammengetragen */
data ergebnis;
  set log:;
run;

proc sort data=ergebnis;
  by logdatei;
run;

/* alle nicht mehr benötigten Dateien werden gelöscht */
proc datasets lib=work nolist;
  delete log: inhalt;
run; quit;
```

Dieses Vorgehen eignet sich nicht nur zum Einlesen der Logdateien. Falls es Anwendungen oder andere Systeme gibt, die Daten in gleichbleibenden Textdateien ausgeben, so lassen sie sich ebenfalls mit der vorgestellten Methode in SAS einlesen.

Nun wird eine Zusammenfassung erstellt. Der Anwender, der die Analysedatei erstellt hat, soll auf einen Blick zum einen sehen, wie viele Logdateien aus welchem Verzeich-

nis ausgewertet wurden. Zum anderen soll eine Übersicht entstehen, wie viele relevante Meldungen in den einzelnen Logdateien gefunden wurden. Dazu wird eine kleine Textdatei erstellt. Des Weiteren wird mit der Libname-Engine eine Excelmappe erstellt. Sie enthält drei Arbeitsblätter. Im ersten Arbeitsblatt sind alle relevanten Meldungen enthalten, im zweiten Arbeitsblatt die Errors und im dritten Arbeitsblatt die Warnings. Alle Arbeitsblätter sind identisch aufgebaut. Die Spalte A enthält den Dateinamen der untersuchten Logdatei, die Spalte B den Typ und die Spalte C die konkrete Meldung. Damit lässt sich filtern, in welchem Programmteil eine relevante Meldung von SAS geschrieben wurde.

```
/* die Ergebnisse werden nach Excel in das Verzeichnis geschrieben,
in dem auch die Log-Dateien liegen. Doppelter Unterstrich am Anfang
des Dateinamens sortiert die Datei nach oben */
libname ausgabe excel "&verzeichnis\__Logauswertung.xlsx";

data ausgabe.Logauswertung ausgabe.Errors ausgabe.Warnings;
  set ergebnis;
  if typ eq 'ERROR' then output ausgabe.errors;
  if typ eq 'WARNING' then output ausgabe.warnings;
  if   typ   in   ('NOTE',   'ERROR',   'WARNING')   then   output
ausgabe.logauswertung;
run;

libname ausgabe clear;

/* eine Zusammenfassung wird erstellt */
proc sql noprint;
   select count(*) into: anznotes
   from ergebnis
   where typ eq 'NOTE';
   select count(*) into: anzwarnings
   from ergebnis
   where typ eq 'WARNING';
   select count(*) into: anzerrors
   from ergebnis
   where typ eq 'ERROR';
quit;

data _null_;
  format d deudfwkx. t time8.;
  file "&verzeichnis\__Auswertungsprotokoll.txt";
  d=date();
  t=time();
  put "Logfileauswertung vom " d " um " t;
  put ;
  put "Verzeichnis: &verzeichnis.";
  put "Es wurden %trim(&dateianzahl.) Logdateien untersucht";
  put ;
  put "Anzahl ERRORS: %trim(&anzerrors.)";
  put "Anzahl WARNINGS: %trim(&anzwarnings.)";
```

```
  put "Anzahl NOTES: %trim(&anznotes.)";
  put ;
  put  "Die  Details  werden  in  &verzeichnis.\__Logauswertung.xlsx
ausgeben";
run;

%mend analysiere_log;
```

4 Alternative: SAS Enterprise Guide

Das beschriebene Vorgehen wurde im SAS Display Manager ausgeführt. Der SAS Enterprise Guide bietet gleichfalls die Möglichkeit, Programme parallel ausführen zu lassen. Über die grafische Oberfläche ist dies zugleich schön anzuschauen. Dazu findet sich unter den Eigenschaften (zu finden unter dem Menü „Datei“) das folgende Formular. Dort einfach den Haken setzen. Für den Fehlerfall stehen drei Aktionen zur Auswahl. Idealerweise wird „Alle Elemente stoppen“ ausgewählt.

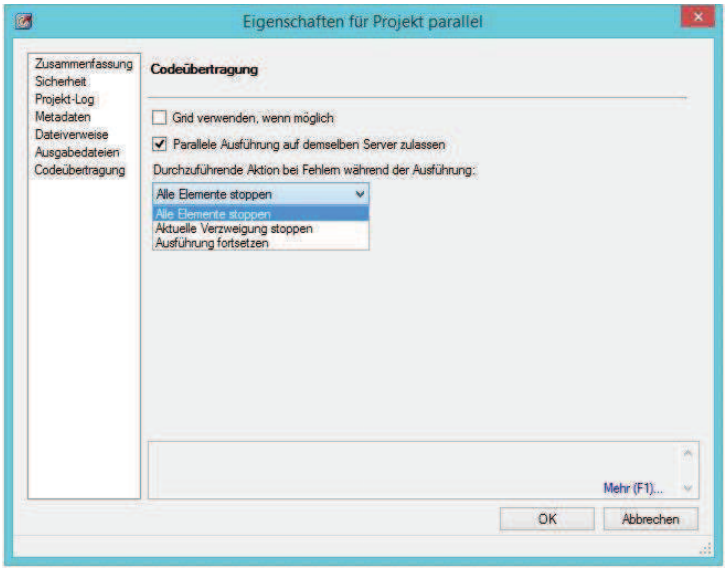

Abbildung 2: Parallele Ausführung einrichten

Die gerade laufenden Codeknoten sind grün hinterlegt, die wartenden stehen auf gelb, im Anwendungsroutinenstatus ist eine Liste zu sehen, wo der Ablauf gerade steht. Falls es zu einer Warnung oder zu einem Fehler kommt, hat der Codeknoten rechts oben ein kleines, entsprechendes Symbol, so dass ersichtlich ist, wo es zu einem Problem kam.

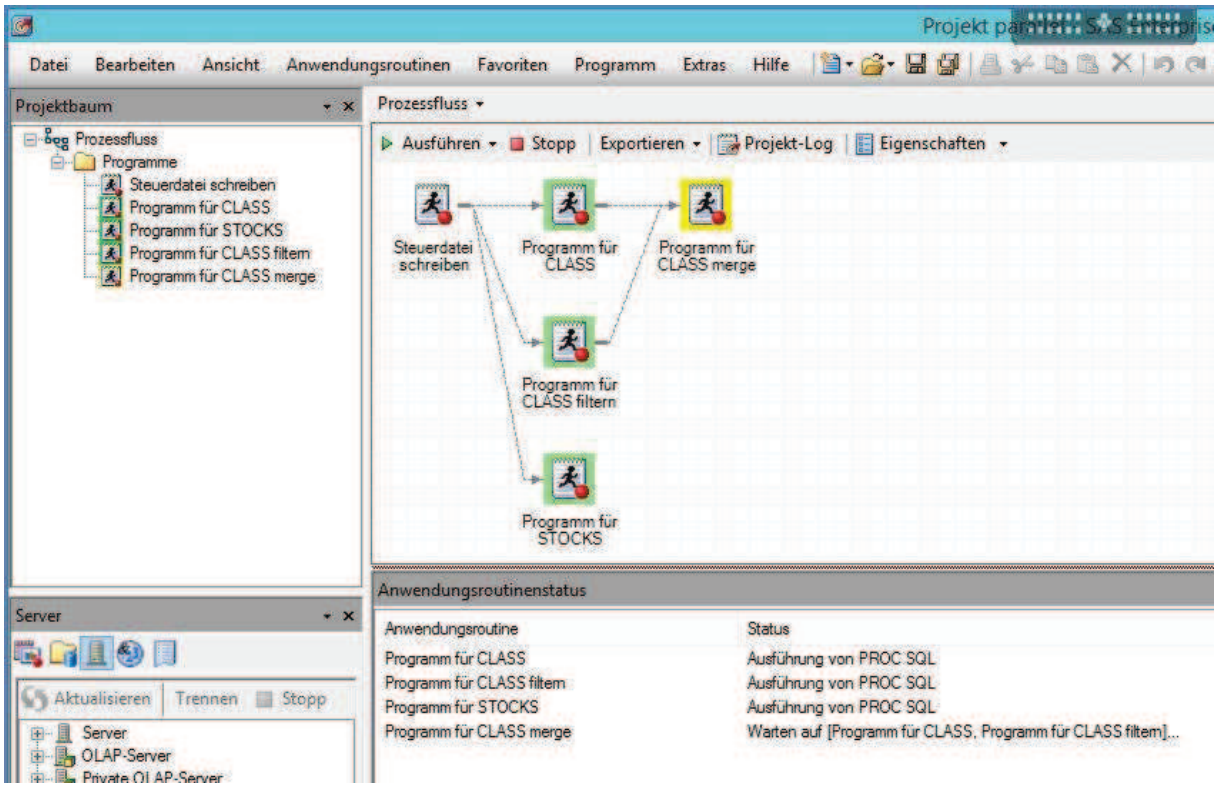

Abbildung 3: Prozessfluss im SAS Enterprise Guide

Für die Weitergabe von Makrovariablen und die Nutzung der Bibliotheken gelten die gleichen Einschränkungen wie oben beschrieben. Im SAS Enterprise Guide lässt sich die Textdatei zur Weitergabe der Makros genauso nutzen. Jedoch hat der SAS Enterprise Guide die Möglichkeit, Eingabeaufforderungen einzubauen. Dies ermöglicht dem Nutzer, über eine grafische Oberfläche Werte einzugeben oder auszuwählen, z. B. ein Verzeichnis oder ein Datum aus einem Kalendersteuerelement.

Des Weiteren lässt sich eine Logzusammenfassung anfordern. Darin ist eine Übersicht enthalten, wie viele Errors, Warnings und Notes im gesamten Prozessfluss enthalten sind. Mit ein paar Klicks lassen sich auf diese Weise die Logfiles durchsuchen. Dazu oben auf „Projekt-Log“ klicken. Im Projekt-Log lässt sich die Log-Zusammenfassung aufrufen. Sie ist in die Reiter „Fehler“, „Warnungen“ und „Hinweise“ gegliedert. Zunächst sind alle Meldungen angezeigt. Sind nur die Errors von Interesse, so lassen sich diese mit einem Klick auf „Fehler“ filtern. Es wird sogar ausgegeben, an welcher Stelle und in welcher Zeile der Error vorgekommen ist. Dies ermöglicht ein schnelles Auffinden im entsprechenden Programm.

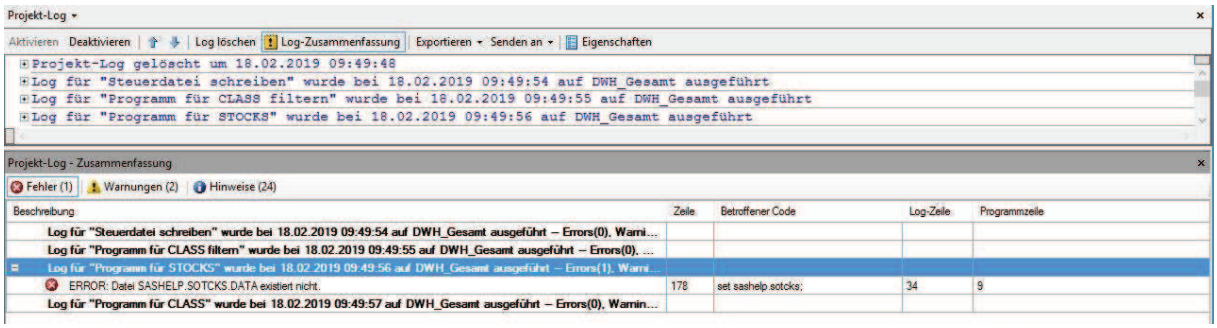

Abbildung 4: Projekt-Log mit Log-Zusammenfassung

Im Unterschied zum oben gezeigten Vorgehen im SAS Display Manager ist die Logzusammenfassung standardisiert, es wird jede Meldung aufgenommen. Im Analysemakro besteht die Möglichkeit, die Meldungen auf die eigenen Bedürfnisse zu filtern.

Literatur

[1] Troy Martin Hughes: Parallel Processing Your Way to Faster Software and a Big Fat Bonus: Demonstrations in Base SAS. https://support.sas.com/resources/papers/proceedings17/0870-2017.pdf

[2] Benjamin First, Steven First: Improving Efficiency in SAS Enterprise Guide: Parallel Processing and Other Hidden Gems. https://support.sas.com/resources/papers/proceedings17/0927-2017.pdf

[3] Michael A. Walega: Search Your LOG Files for Problems the Easy Way: Use LOGCHK.SAS. https://www.lexjansen.com/nesug/nesug97/coders/walega.pdf