Prozessdokumentation und Auditing mit PROC SCAPROC

Dennis Voigt Timm Euler
viadee Unternehmensberatung GmbH
Anton-Bruchhausen-Str. 8
48147 Münster
dennis.voigt@viadee.de timm.euler@viadee.de

Zusammenfassung

In vielen Unternehmen bilden SAS-Programme wichtige regelmäßige Prozesse ab, zum Beispiel das Beladen eines Data Warehouses oder die Aktualisierung statistischer Modelle. Der Beitrag stellt eine SAS-Prozedur vor, mit der die Dokumentation jeder einzelnen Prozessausführung leicht automatisiert und damit die Nachvollziehbarkeit aller Datenflüsse gewährleistet werden kann. Dazu wird die Ausgabe der Prozedur noch mit SAS-Mitteln nachbearbeitet. In der Beispielanwendung wird bei jeder Ausführung eines SAS-Programms u. a. automatisch mitprotokolliert, welche SAS-Tabellen geschrieben und gelesen wurden, so dass Abhängigkeiten zwischen Programmen auswertbar werden. Es entstehen revisionssichere Metadaten, die für Auditing-Zwecke oder zur Dokumentation der Data Lineage geeignet sind.

Schlüsselwörter: Metadaten, Auditing, PROC SCAPROC, DS2, Regular Expression, ETL

1 Einleitung

In automatisierten Datenverarbeitungen erfüllen technische Metadaten vielfältigen Nutzen. Sie unterstützen etwa das Monitoring, die Wartung, die Analyse von Abhängigkeiten und Änderungsauswirkungen sowie die Dokumentation und ermöglichen so eine durchgehende Überwachung technischer Prozesse. Beispielsweise können Metadaten dazu dienen, eine Batchnacht bestmöglich und automatisch zu planen, Performance-Probleme in Programmen zu analysieren oder, im Fall von SAS-Programmen, zu erkennen, welche Makro-Variablen, Datasets oder Dataset-Variablen pro Prozess gelesen und geschrieben werden. Werden diese Metadaten systematisch erfasst und abgelegt, spricht man von einem Metadata Warehouse. Ein solches Metadata Warehouse unterstützt die o.g. Anforderungen zur Prozessüberwachung bestmöglich, jedoch stellt die Erfassung und strukturierte Ablage der Metadaten eine administrative und technische Herausforderung dar.

Für SAS-Programme bietet sich hier eine sehr nützliche Lösung an: PROC SCAPROC. Diese Prozedur läuft während der Ausführung eines SAS-Programms parallel dazu. Dabei wird dieses förmlich ausspioniert, denn die Prozedur protokolliert alle Interaktionen mit SAS-Daten mit. Am Ende wird eine sog. RECORD-Datei ausgegeben, in der alle

Schritte des SAS-Programms strukturiert wiedergegeben werden. Aus dieser Datei können die o.g. Metadaten systematisch extrahiert werden.
Da die Ausgabedatei je nach analysiertem SAS-Programm sehr groß werden kann, sollte man die Inhalte konsolidiert in Tabellen ablegen. Wie dieser Schritt am besten bzw. am übersichtlichsten zu bewerkstelligen ist, wie PROC SCAPROC genutzt wird und welche Schlussfolgerungen gezogen werden können, wird in diesem Dokument anhand eines durchgehenden Beispiels erläutert.

2 Allgemein

Die Prozedur SCAPROC ist seit der SAS Version 9.2 ein Bestandteil von SAS Base. Bei einem Aufruf dieser Prozedur werden alle nachfolgenden I/O Informationen eines Programmablaufs mitprotokolliert und in eine strukturierte Ausgabedatei geschrieben, die im `RECORD`-Aufruf festgelegt wird.

```
PROC SCAPROC;
  RECORD fileref <ATTR> <OPENTIMES> <INTCON> <EXPANDMACROS>;
RUN;
```

Durch die Angabe der Schalter `ATTR`, `OPENTIMES`, `INTCON` können die folgenden detaillierten Informationen hinzugefügt werden:

- `ATTR` – Die Variablen der Datasets werden ausgegeben
- `OPENTIMES` – Die Zugriffsinformationen eines Datasets werden ausgegeben
- `INTCON` – Die Integritätsbedingungen werden ausgegeben.

Durch den Schalter `EXPANDMACROS` werden Makros für die Protokollierung aufgelöst. Dabei ist zu beachten, dass der Prozeduraufruf außerhalb eines Makros ausgeführt wird.

Am Ende des Programmablaufs muss über einen weiteren Prozeduraufruf die Protokollierung beendet und die Datei geschrieben werden.

```
PROC SCAPROC;
  WRITE;
RUN;
```

Um bestimmte Probleme zu umgehen, ist es sinnvoll, den Prozeduraufruf innerhalb der Batch-Datei zu implementieren, die den SAS Prozess startet.

```
"[Pfad zur SAS.exe]" -initstmt   "[SCAPROC-Aufruf]"
                     -sysin      "[Pfad zum SAS-Programm]"
```

Bei einem Batch-Aufruf wird das `WRITE`-Statement am Ende des Programmablaufs automatisch ausgeführt.

Die Prozedur SCAPROC kann in der SAS Version 9.4 nicht mit inkludiertem Code umgehen. In der Ausgabedatei wird der inkludierte Code als ein einzelnes Task-Member identifiziert. Durch das Aktivieren von `EXPANDMACROS` wird die Ausgabe wohl in einzelne Task geschrieben, aber der ausgeführte Code kann nicht ermittelt werden. Um das zu umgehen, kann man den kompletten Code in einem Makro einschließen. In Kombination mit dem Schalter kann SCAPROC alle Informationen ausgeben.

3 Beispiel

In diesem Dokument wird der Analysezyklus ausgehend von dem ausgeführten SAS-Programm, über das Extrahieren der von PROC SCAPROC protokollierten Daten bis hin zum Erkennen von Zusammenhängen in diesen Daten anhand des folgenden Beispiels erklärt.

```
DATA KSFE.COPY_CARS;
  SET KSFE.CARS;
RUN;

PROC SORT DATA=KSFE.COPY_CARS
          OUT=KSFE.COPY_CARS_SORT;
  BY HORSEPOWER;
RUN;
```

Hier wird eine Kopie des Datasets CARS in der Bibliothek KSFE erstellt und dann eine weitere, nach der Pferdestärke des Kraftfahrzeugs sortierte, Kopie erstellt. Um das Beispiel übersichtlich zu halten, wurde das Dataset CARS für dieses Beispiel aus der Bibliothek SASHELP nach KSFE kopiert, da SASHELP mehrere Quellpfade besitzt und diese per CONCAT zusammengeführt werden müssen.

4 RECORD

Die Ausgabedatei von PROC SCAPROC, die durch die fileref-Angabe im RECORD-Statement (siehe Abschnitt 2) vorgegeben wird, stellt die Informationen taskorientiert dar. Dabei ist jede Ausführung einer SAS-Prozedur oder eines Data Steps ein Task. Zu jedem Task werden diverse I/O Informationen gesammelt.

In dem Beispiel sieht die RECORD-Datei wie in Abbildung 1 aus. Die Aufstellung aller JOBSPLIT-Variationen ist von SAS sehr gut dokumentiert [1].

Man kann die Ausgabe auch durch PUTLOG-Statements erweitern, um individuelle Informationen anzufügen. Ein mögliches Anwendungsbeispiel dafür wäre ein Step-Makro, welches die einzelnen Steps eines SAS-Prozesses abhängig von Bedingungen ausführt. Mit PUTLOG könnten dabei erweiterte Informationen zu den Bedingungen ausgegeben werden. Dabei ist zu beachten, dass die PUTLOG-Ausgaben durch einheitliche Schlüsselwörter klar erkennbar sind, damit diese Besonderheiten in der RECORD-Datei eindeutig identifiziert und ausgelesen werden können.

```
/* JOBSPLIT: JOBSTARTTIME 13DEC2016:08:52:24.31 */
/* JOBSPLIT: TASKSTARTTIME 13DEC2016:08:52:24.34 */
/* JOBSPLIT: DATASET INPUT SEQ KSFE.CARS.DATA */
/* JOBSPLIT: LIBNAME KSFE BASE 'D:\Temp\SEG3436\SAS Temporary Files\_TD8524_NBW7W448_\Prc2' */
/* JOBSPLIT: DATASET OUTPUT SEQ KSFE.COPY_CARS.DATA */
/* JOBSPLIT: LIBNAME KSFE BASE 'D:\Temp\SEG3436\SAS Temporary Files\_TD8524_NBW7W448_\Prc2' */
/* JOBSPLIT: ELAPSED 14  */
/* JOBSPLIT: SYSSCP WIN */
/* JOBSPLIT: PROCNAME DATASTEP */
/* JOBSPLIT: STEP SOURCE FOLLOWS */

DATA KSFE.COPY_CARS;
  SET KSFE.CARS;
RUN;


/* JOBSPLIT: TASKSTARTTIME 13DEC2016:08:52:24.35 */
/* JOBSPLIT: DATASET INPUT SEQ KSFE.COPY_CARS.DATA */
/* JOBSPLIT: LIBNAME KSFE BASE 'D:\Temp\SEG3436\SAS Temporary Files\_TD8524_NBW7W448_\Prc2' */
/* JOBSPLIT: DATASET OUTPUT SEQ KSFE.COPY_CARS_SORT.DATA */
/* JOBSPLIT: LIBNAME KSFE BASE 'D:\Temp\SEG3436\SAS Temporary Files\_TD8524_NBW7W448_\Prc2' */
/* JOBSPLIT: FILE OUTPUT C:\TEMP\RECORD.txt */
/* JOBSPLIT: SYMBOL GET SYSSORTDETAILS */
/* JOBSPLIT: SYMBOL GET SYSSORTTRACELEVEL */
/* JOBSPLIT: SYMBOL GET SYSSORTTRACE */
/* JOBSPLIT: ELAPSED 32  */
/* JOBSPLIT: PROCNAME SORT */
/* JOBSPLIT: STEP SOURCE FOLLOWS */
PROC SORT DATA=KSFE.COPY_CARS
          OUT=KSFE.COPY_CARS_SORT;
  BY HORSEPOWER;
RUN;


/* JOBSPLIT: JOBENDTIME 13DEC2016:08:52:24.38 */
/* JOBSPLIT: END */
```

Abbildung 1: Beispiel RECORD-Datei

5 ETL

Im nächsten Schritt kann die RECORD-Datei eingelesen und in konsolidierten Tabellen abgelegt werden. Wie die Informationen ausgelesen werden, ist jedem Entwickler selbst überlassen und kann über mehrere Wege erfolgen. Das folgende Beispiel zeigt einen einfachen Weg, der einige Funktionen, u. a. PRXPARSE im PROC DS2 nutzt, die eine SAS Version von 9.4 voraussetzen. Gute Gründe hierbei PROC DS2 zu nutzen sind u. a. die Übersichtlichkeit und einfache Wartbarkeit des Codes. Da keine Dateien im PROC DS2 eingelesen werden können, ist es notwendig, einen Data Step voranzustellen, der die von PROC SCAPROC erzeugte RECORD-Datei in ein SAS-Dataset einliest.

```
DATA WORK.SCA_OUT;
  LENGTH CODE $10000;
  INFILE SCAFILE;
  INPUT;
  CODE = UPCASE(_INFILE_);
RUN;
```

Jetzt kann die Ausgabetabelle WORK.SCA_OUT in PROC DS2 eingelesen werden.

```
PROC DS2;
  DATA _NULL_;

    DECLARE PACKAGE HASH h_DATASETS (.,'','','','','');
    DECLARE PACKAGE HASH h_LIBNAME  (.,'','','','','');

    DECLARE VARCHAR(200) DWH_LIBNAME_PATH;
    DECLARE VARCHAR(50)  DWH_DATASET_NAME;
    DECLARE VARCHAR(9)   DWH_DATASET_STREAM;
    DECLARE VARCHAR(8)   DWH_LIBNAME
                         DWH_LIBNAME_ENGINE
                         DWH_DATASET_LIB;
    DECLARE VARCHAR(6)   DWH_DATASET_ACCESS
                         DWH_DATASET_TYPE;
    DECLARE INT          DWH_TASK
                         TMP_SWITCH_TASK
                         TMP_IN_TASK;
    DECLARE DOUBLE       MARKE_DATASETS
                         MARKE_LIBNAME;

    RETAIN MARKE_DATASETS
           MARKE_LIBNAME
           DWH_TASK 1
           TMP_IN_TASK 1;

    FORWARD DEFINE_PRX_MARKE
            SET_TASK
            ADD_DATASETS
            ADD_LIBNAME;
```

Die Methoden `INIT`, `RUN` und `TERM` sind Standard-Methoden des PROC DS2. Sie bezeichnen den Ausführungszeitpunkt, zu dem sie ausgeführt werden. `INIT` wird immer zu Beginn der Verarbeitung ausgeführt und `TERM` am Ende. Die Methode `RUN` wird bei jeder Zeile des Eingabedatasets ausgeführt.

```
    METHOD INIT();
      DEFINE_HASH();
      DEFINE_PRX_MARKE();
    END;

    METHOD RUN();
      SET WORK.SCA_OUT;
      SET_TASK();
      ADD_DATASETS();
      ADD_LIBNAME();
    END;

    /* Schreiben der Hashobjekte in SAS-Datasets */
    METHOD TERM();
```

```
  h_DATASETS.OUTPUT('WORK.SCA_DATASETS');
  h_LIBNAME.OUTPUT('WORK.SCA_LIBNAME');
END;
```

In der Methode DEFINE_HASH werden die Hash-Objekte definiert. Dort wird angegeben, welche Variablen im Hash-Objekt aufgeführt werden und welche davon als Schlüsselvariable fungieren. Variablen, die bei den Schlüsselvariablen aufgeführt sind, werden nicht automatisch in das Dataset übernommen, sondern müssen ebenfalls als Variable unter Daten aufgeführt werden.

```
METHOD DEFINE_HASH();
  h_DATASETS.KEYS([DWH_TASK DWH_DATASET_STREAM
                   DWH_DATASET_ACCESS DWH_DATASET_LIB
                   DWH_DATASET_NAME DWH_DATASET_TYPE]);
  h_DATASETS.DATA([DWH_TASK DWH_DATASET_STREAM
                   DWH_DATASET_ACCESS DWH_DATASET_LIB
                   DWH_DATASET_NAME DWH_DATASET_TYPE]);
  h_DATASETS.DEFINEDONE();

  h_LIBNAME.KEYS([DWH_TASK DWH_LIBNAME DWH_LIBNAME_ENGINE
                  DWH_LIBNAME_PATH]);
  h_LIBNAME.DATA([DWH_TASK DWH_LIBNAME DWH_LIBNAME_ENGINE
                  DWH_LIBNAME_PATH]);
  h_LIBNAME.DEFINEDONE();
END;
```

Als nächstes werden die regulären Ausdrücke definiert, die zum Parsen der Record-Datei dienen. Durch die Funktion PRXPARSE werden die regulären Ausdrücke einmalig erstellt und in den Variablen mit dem Präfix MARKE gespeichert.

```
METHOD DEFINE_PRX_MARKE();
  MARKE_DATASETS = PRXPARSE('/\/\* JOBSPLIT: DATASET
                   (INPUT|OUTPUT|UPDATE) (?:(SEQ|MULTI) )?
                   ([^ .]*).([^ .]*).([^ .]*) \*\//');
  MARKE_LIBNAME = PRXPARSE('/\/\* JOBSPLIT: LIBNAME ([^ ]*)
                  ([^ (]*) ['''''"]*([^()'''''"]*)['''''"]*
                  \*\//');
END;
```

Um zu protokollieren, in welchem Task welche Informationen geschrieben wurden, gibt es hier die Methode SET_TASK. Diese Methode kontrolliert zeilenweise, ob ein „JOBSPLIT-Wechsel“ stattgefunden hat.

```
METHOD SET_TASK();
  IF SUBSTR(CODE,1,12) EQ '/* JOBSPLIT:' THEN DO;
    TMP_SWITCH_TASK = TMP_IN_TASK EQ 0;
    TMP_IN_TASK = 1;
  END;
  ELSE DO;
    TMP_IN_TASK = 0;
```

```
    END;
    IF TMP_SWITCH_TASK EQ 1 THEN DO;
      DWH_TASK + 1;
    END;
  END;
```

In den beiden nachfolgenden Methoden `ADD_DATASETS` und `ADD_LIBNAME` wird kontrolliert, ob der reguläre Ausdruck mit der Zeile übereinstimmt. Wenn er übereinstimmt, werden die Gruppen-Variablen der Zeile ausgelesen und in die passenden Variablen der Hash-Objekte gespeichert.

```
  METHOD ADD_DATASETS();
    IF PRXMATCH(MARKE_DATASETS,CODE) THEN DO;
      DWH_DATASET_STREAM = PRXPOSN(MARKE_DATASETS,1,CODE);
      DWH_DATASET_ACCESS = PRXPOSN(MARKE_DATASETS,2,CODE);
      DWH_DATASET_LIB    = PRXPOSN(MARKE_DATASETS,3,CODE);
      DWH_DATASET_NAME   = PRXPOSN(MARKE_DATASETS,4,CODE);
      DWH_DATASET_TYPE   = PRXPOSN(MARKE_DATASETS,5,CODE);
      h_DATASETS.ADD();
    END;
  END;

  METHOD ADD_LIBNAME();
    IF PRXMATCH(MARKE_LIBNAME,CODE) THEN DO;
      DWH_LIBNAME        = PRXPOSN(MARKE_LIBNAME,1,CODE);
      DWH_LIBNAME_ENGINE = PRXPOSN(MARKE_LIBNAME,2,CODE);
      DWH_LIBNAME_PATH   = PRXPOSN(MARKE_LIBNAME,3,CODE);
      h_LIBNAME.ADD();
    END;
  END;

RUN;
QUIT;
```

In dem Beispiel sehen am Ende der Extraktion die beiden Tabellen wie folgt aus:

	DWH_TASK	DWH_DATASET_STREAM	DWH_DATASET_LIB	DWH_DATASET_NAME	DWI
1	1	INPUT	KSFE	CARS	DATA
2	1	OUTPUT	KSFE	COPY_CARS	DATA
3	2	INPUT	KSFE	COPY_CARS	DATA
4	2	OUTPUT	KSFE	COPY_CARS_SORT	DATA

Abbildung 2: Beispiel Datasets

	DWH_TASK	DWH_LIBNAME	DWH_LIBNAME_ENGINE	
1	1	KSFE	BASE	D:\TEMP
2	2	KSFE	BASE	D:\TEMP

Abbildung 3: Beispiel Libnames

Hier sind In- und Output jedes Programmschritts strukturiert abgelegt und einer weiteren automatisierten Auswertung leicht zugänglich, wie der folgende Abschnitt demonstriert.

6 Auswertungsmöglichkeiten

Mit der so erstellten Datengrundlage können über eine einfache Selektion Abhängigkeiten zwischen Programmschritten und damit auch ganzen Programmen, ermittelt werden. Dafür werden je SAS-Programm erst einmal alle Ein- und Ausgabetabellen selektiert. Diese Daten werden dann miteinander verbunden. Schon werden die Zusammenhänge der Prozesse sichtbar und können beispielsweise genutzt werden, ganze Batchverarbeitungen effizient zu planen.

```
PROC SQL;
  CREATE TABLE BATCH_PRG AS
  SELECT MAX(DWH_DATASET_STREAM) AS DWH_PRG_STREAM
        ,DWH_DATASET_LIB         AS DWH_PRG_LIB
        ,DWH_DATASET_NAME        AS DWH_PRG_DATASET_NAME
  FROM SCA_DATASETS
  GROUP BY DWH_DATASET_LIB
          ,DWH_DATASET_NAME
  HAVING COUNT(DWH_DATASET_STREAM) EQ 1
     AND MAX(DWH_DATASET_STREAM) NE 'UPDATE';
QUIT;
```

	DWH_PRG_STREAM	DWH_PRG_LIB	DWH_PRG_DATASET_NAME
1	INPUT	KSFE	CARS
2	OUTPUT	KSFE	COPY_CARS_SORT

Abbildung 4: Beispiel Ein- und Ausgabetabellen auf Programmebene

7 Fazit

Die Prozedur SCAPROC erleichtert das Sammeln von technischen Metadaten zu Ausführungen von SAS-Programmen. Diese Metadaten sollten jedoch strukturiert abgelegt werden, um ihre Nutzung zu erleichtern. Dazu müssen sie aus der sequenziellen Ausgabedatei von PROC SCAPROC extrahiert werden. Ein Verfahren dazu wurde in diesem Dokument vorgestellt.
Anwendungsmöglichkeiten bieten sich insbesondere bei der automatisierten Prozessüberwachung (*process monitoring*), bei der Metadatenerfassung im Bereich ETL/Data Warehousing oder für die Dokumentation der (regelmäßigen) Ausführung von SAS-Programmen.

Literatur

[1] Base SAS 9.4 Procedures Guide, Sixth Edition SAS, PROC SCAPROC, unter: http://support.sas.com/documentation/cdl/en/proc/69850/HTML/default/viewer.htm#p0k5uaxpaz2uzin1qvbqmmafnqtl.htm (abgerufen am 20.02.2017).