Benutzerdefinierte Programm-Meldungen mit Log4SAS im SAS-Log und darüber hinaus

Alexander Zakharenko
HMS Analytical Software GmbH
Rohrbacher Straße 26
69115 Heidelberg
Alexander.Zakharenko@analytical-software.de

Zusammenfassung

Kaum ein SAS-Programm kommt ohne Meldungen an den Benutzer aus. Hierzu gibt es schon viele gute und etablierte Lösungen. Warum also etwas ändern? „Never touch a running system“. Durch die vorgestellte SAS-Technologie Log4SAS kann man verschiedene Ziele mit den SAS-Meldungen erreichen, nicht nur das SAS-Log. Auch externe Fehlerlogs können angesteuert werden. Ebenso das Windows Event Log und andere Monitoring Tools wie Xymon (früher: Hobbit) oder Tivoli. Müssen dazu die Programme angepasst werden? Ja. Aber nur einmal! Danach ist es nur eine Frage der Konfiguration in welchem Umfang und wohin die Meldungen aus den SAS-Programmen verteilt werden.

Schlüsselwörter: SAS Log, SLF, Log4SAS, Windows Event Log

1 Überblick

Wenn dem Benutzer etwas mitzuteilen ist, muss mein Programm eine Meldung herausgeben. Das ist jedem Entwickler klar. Doch mit wachsender Komplexität der Projekte und der Verteilung der Rollen innerhalb der Teams werden die Meldungen dem Einen zu viel und einem Anderen zu wenig. Ein Administrator will nur dann etwas wissen, wenn kritische Fehler passieren, die in seinen Zuständigkeitsbereich fallen, bei der Fehlersuche will der Entwickler möglichst viele Informationen haben, der Fachanwender interessiert sich nur für Meldungen, die ablaufrelevant sind.
Die erfreuliche Nachricht ist: wir können nun Allen gerecht werden und das simultan. Dank der seit SAS 9.2 verfügbaren SAS Logging Facility (weiter “SLF”) können die Meldungen unterschiedliche Ziele ansteuern und dabei je nach Kategorie und Inhalt gefiltert werden.
SLF ist in der Lage, die Meldungen nach SAS Log, Textdateien, Datenbanktabellen (SAS Datasets und Drittanbieter), Systemlogs (Windows, Unix, z/OS), Message Queues und selbstdefinierten Java-Klassen zu schreiben. Kontexte, von denen aus SLF verwendet werden kann, sind DATA Step, Makro und SCL.
SAS Logging Facility ist eine Anlage, die eine mächtige Infrastruktur bereitstellt. Alles was mit Logging zu tun hat kann über SLF mit wenig Aufwand und hoher Effizienz ab-

gebildet werden. Wir können den Prozess der Message-Ausgabe in einem dreistufigen Ablauf darstellen:

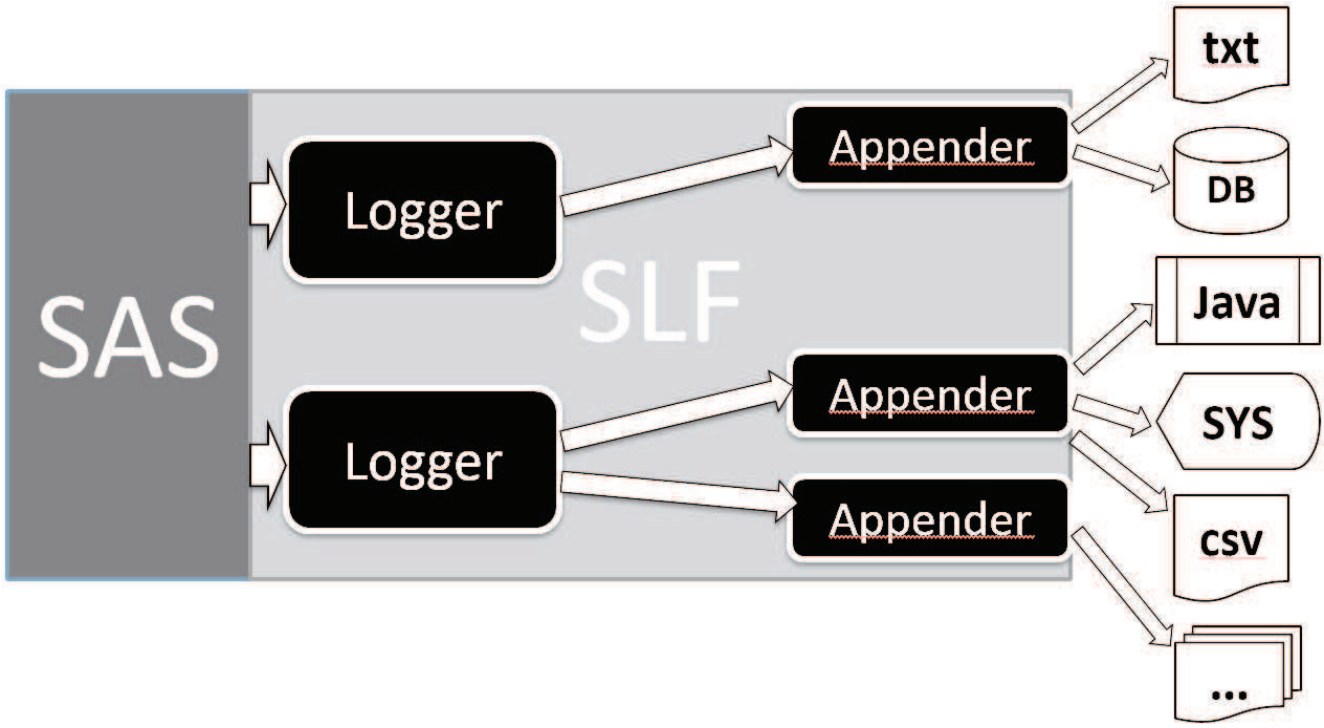


Abbildung 1: SAS Logging Facility – Überblick.

- SAS gibt die Meldungen an die Logger.
- Logger nehmen die Meldungen entgegen, filtern sie nach Dringlichkeitsstufe und geben nur die relevanten Meldungen weiter an die Appender.
- In einem Appender wird an die Message ein Layout angewendet und diese wird je nach definierter Destination herausgeschrieben. Außerdem gibt es an der Stelle die Möglichkeit, die Meldungen nach dem Textinhalt zu filtern.

Ab SAS 9.2 wird SLF mitgeliefert. Das heißt, wenn Sie mit der besagten oder höheren Version arbeiten, ist die ganze Infrastruktur bereits vorhanden. Alles was Sie brauchen ist diese zu konfigurieren und schon können Ihre Programme die SLF verwenden.

Konfiguriert wird entweder direkt in Programmen oder mittels einer XML-Datei. Wir empfehlen die XML-Version, da die ganze Konfiguration an einer Stelle untergebracht wird; man muss sie nicht zusammensuchen und kann die Konfigurationsdatei je nach Projekt beim SAS-Aufruf austauschen.

Intern verwendet SAS die SLF ebenso. Insbesondere bei dem Plattformbetrieb gibt es viele vordefinierte Logger und Appender, die Sie auch mitbenutzen oder ganz ändern können. Wenn die Einstiegshürde überwunden ist, lässt sich SLF sehr bequem und effizient benutzen.

2 Bausteine und Zusammenspiel

2.1 Die Hauptelemente

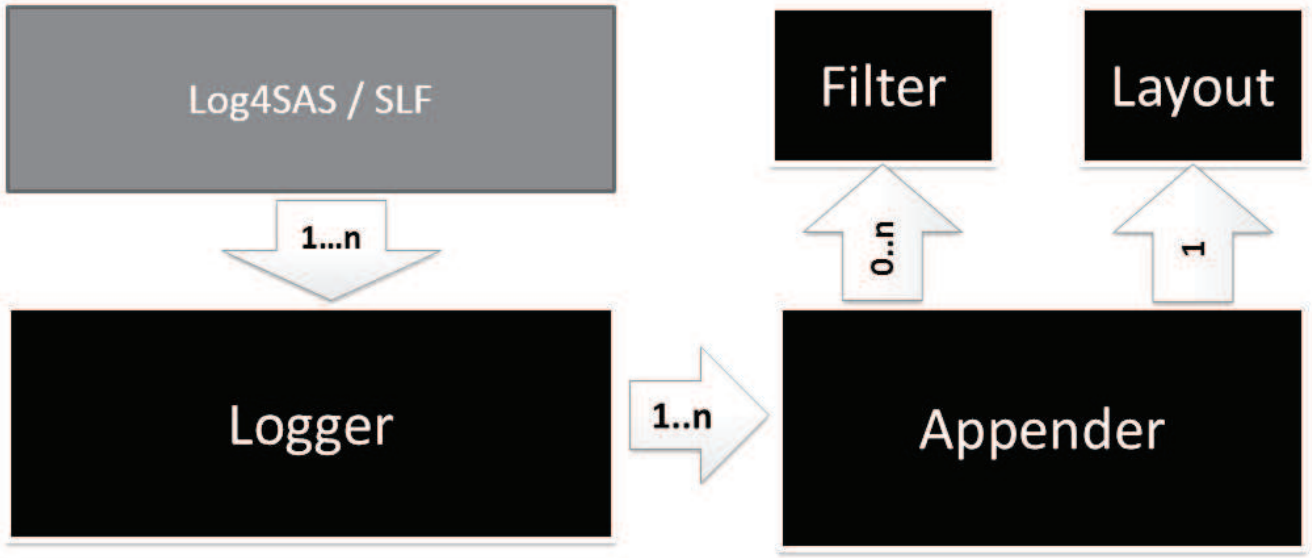


Abbildung 2: Vier Bausteine von SAS Logging Facility

Logger
Ein Logger ist ein benannter Ausgabekanal, der Nachrichten aus SAS entgegennimmt. Durch die Verwendung verschiedener Loggernamen lassen sich die Nachrichten kategorisieren. Zusätzlich lassen sich auch Hierarchien erstellen. Der sogenannte root-Logger ist in der Hierarchie an oberster Stelle und alle anderen Logger erben seine Einstellungen.

Appender
Appender sind für das Schreiben der Nachrichten zuständig. Nachrichten können in Textdateien, auf die Konsole, in Betriebssystemlogs, Datenbanken, Message Queues und eigene Java Klassen geschrieben werden.

Layout
Durch die Verwendung von Layouts ist es möglich die Nachricht zu formatieren und mit weiteren Informationen zu ergänzen. So können zusätzlich u.a. statischer Text, Zeitstempel und Systeminformationen ausgegeben werden.

Filter
Filter ermöglichen eine zusätzliche Filterung bei der Nachrichtenbehandlung. So können Nachrichten aufgrund des Log-Levels oder enthaltenem Text gefiltert werden.

2.2 Zusammenspiel der Elemente

SLF kann beliebig viel Logger ansteuern. Jeder Logger kann wiederum beliebig viele Appender befüllen. Jeder Appender kann (kein Muss!) die eingehenden Meldungen durch mehrere Filter durchgehen (oder eben durchfallen) lassen. Außerdem muss jeder

Appender ein Layout haben, damit klar ist, wie die Meldung auszusehen hat (bzw. welche Informationen auszugeben sind).

Schauen wir uns das anhand einer Beispiel-Hierarchie an.

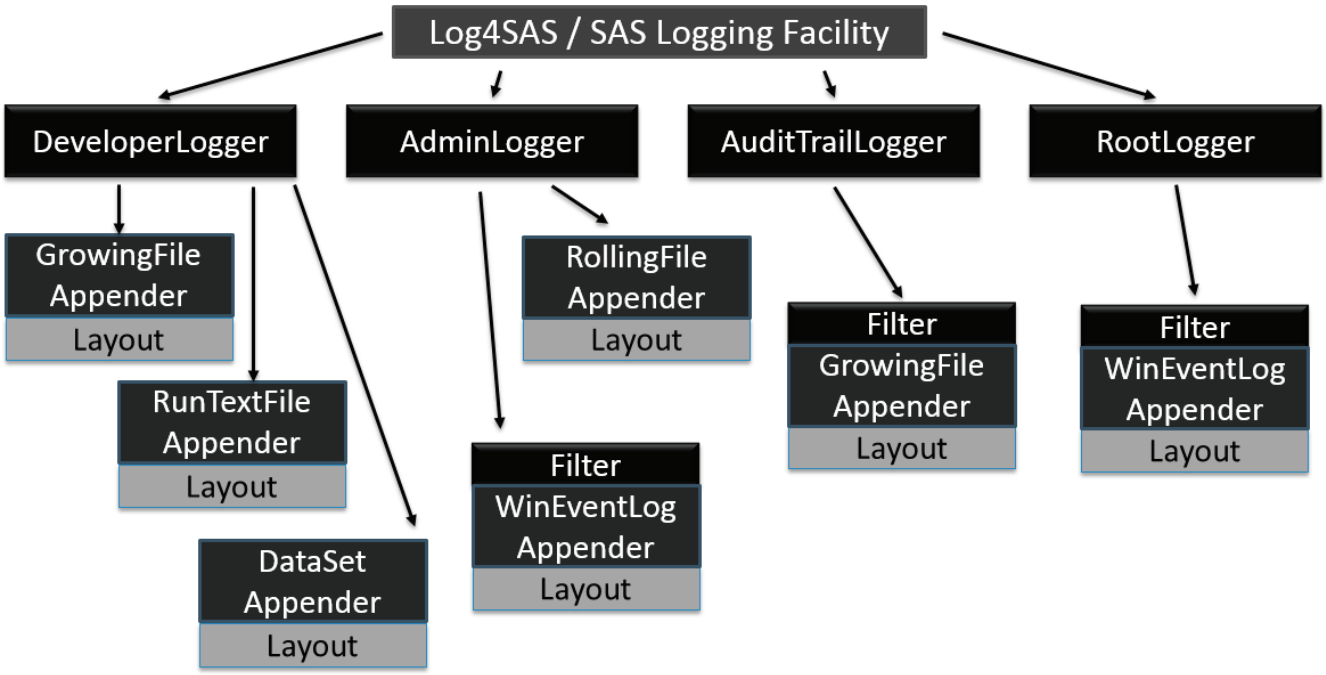


Abbildung 3: Beispiel einer Logger- / Appender-Hierarchie.

In diesem Beispiel befüllt SLF vier Logger. Root- und AuditTrailLogger geben die Meldungen an jeweils einen Appender und beide filtern sie. Der AdminLogger befüllt einen Logger ungefiltert und schreibt nur eine Untermenge der Meldungen ins Windows Event Log. Schließlich gibt es einen DeveloperLogger, der drei Appender ungefiltert befüllt.

Ist es aufgefallen, dass es unterschiedliche Appender-Klassen gibt? Folgende Klassen sind in SLF verfügbar:

- **FileAppender** – schreibt Meldungen in eine Textdatei. Dieser kann sowohl eine Datei pro Session erstellen als auch ans Ende der referenzierten Datei einfügen.
- **RollingFileAppender** – funktioniert wie FileAppender, ist aber in der Lage je nach Konfiguration eine neue Datei basierend auf Datum oder Indexes zu erstellen, z.B. eine Datei je Stunde, Tag, Woche, Monat etc.
- **FilteringAppender** – wie FileAppender, zusätzlich gibt es die Möglichkeit, die Meldungen nach Stufe und Inhalt zu filtern.
- **ConsoleAppender** – schreibt Meldungen in die Windows- oder Unix-Konsole.
- **DBAppender** – die Meldungen landen in einem SAS-Dataset oder eine Tabelle von externen Datenbank, die Tabelle / Data Set muss zum Startpunkt bereits

existieren. Leider ist diese Art der Appender nur im Serverkontext vorhanden, doch es gibt eine Methode mit der man die CSV-Dateien als Vorstufe der Tabellenform herausschreiben kann.

- **JavaAppender** – übergibt die Meldungen an eine benutzerdefinierte Java-Class.
- **JMSAppender** – schickt die Meldungen an Java Message Service (JMS).
- Betriebssystemabhängige Logger:
  - **WindowsEventAppender**
  - **UNXFacilityAppender**
  - **ZOSFacilityAppender**
  - **ZOSWtoAppender**
- Appender im Server/Plattformbetrieb:
  - **IOMServerAppender**
  - **ARMAppender**

3 Konfiguration

3.1 SAS-Aufruf

Die Konfiguration des Loggingsystems kann entweder zur Laufzeit innerhalb von SAS vorgenommen werden oder mittels einer externen XML-Datei. Der Pfad der XML-Konfiguration wird entweder innerhalb der SASV9.CFG-Datei angegeben:

```
-LOGCONFIGLOC="C:\Data\Konfig\SAS\log4sas.xml"
```

oder unter Windows in der Programmverknüpfung bzw. CMD/BAT/Shell-Datei ergänzt:

```
"C:\Program Files\SASHome\x86\SASFoundation\9.3\sas.exe" -CONFIG
"C:\Program Files\SASHome\x86\SASFoundation\9.3\nls\en\sasv9.cfg"
 -LOGCONFIGLOC="C:\Data\Konfig\SAS\log4sas.xml"
```

Es empfiehlt sich die Konfiguration mit einer externen XML-Datei vorzunehmen, da unter anderem bei der Konfiguration innerhalb von SAS nur der FileAppender unterstützt wird. Außerdem ist die externe Datei wesentlich besser in Punkto Übersichtlichkeit – man muss die Informationen über mehrere Code-Module nicht zusammensuchen, denn alles was Loggen betrifft ist in der Datei aufgehoben.

3.2 Konfiguration mit einer XML –Datei

3.2.1 Einstiegsbeispiel

Beginnen wir mit einem kleinen übersichtlichen Beispiel:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE logging [
   <!ENTITY logPath "C:\Demo\logs">
]>
<logging:configuration
xmlns:logging="http://www.sas.com/xml/logging/1.0/">

   <appender name="rootFile" class="FileAppender">
      <param name="File"
value="&logPath;\rootFile__root.log"/>
      <param name="Append"         value="true"/>
      <param name="ImmediateFlush" value="true"/>
      <layout>
         <param name="ConversionPattern" value="%d{ISO8601} %r [%t]
%-5p %c %u - %m"/>
      </layout>
   </appender>
   <root>
      <appender-ref ref="rootFile" />
      <level value="error" />
   </root>
</logging:configuration>
```

Hier wird ein Root-Logger definiert, der einen Appender befüllt. Unter Entity wird die Variable gesetzt, die den Pfad zur Logdatei festlegt (`C:\Demo\logs`). Wir erstellen einen FileAppender namens rootFile und legen fest wo die Zieldatei, also unser Text-Log sich befindet. Der Parameter Append besagt ob die Datei überschrieben (false) oder angefügt (true) wird. Der Parameter `ImmediateFlush` besagt ob die Meldungen gepuffert (false) oder sofort in die Datei geschrieben (true) werden sollen. Wir empfehlen diesen Parameter immer auf true zu setzen, da im Fall der Pufferung die Gefahr besteht, dass die letzten Meldungen am Programmschluss nicht in den Log geschrieben werden.

Im Abschnitt Layout legen wir fest wie die Meldung auszusehen hat. In unserem Beispiel:

%d{ISO8601} – Datum yyyy-MM-dd HH:mm:ss,SSS
%r – Millisekunden seit Programmstart
[%t] – Thread-ID in der der Aufruf stattfand
%-5p – Log-Level der Nachricht (mind. 5 Zeichen lang)
%c – Name des Loggers
%u – Benutzer
%m – Eigentliche Nachricht

Anschließend wird ein Logger definiert. Wir haben einen Root-Logger (dieser muss in der Konfigurationsdatei immer vorhanden sein, auch wenn er leer ist, sonst lässt sich SAS nicht starten). Wir teilen dem Logger mit, welchen Appender er zu beliefern hat und ab welchem Level die Meldungen durchgereicht werden sollen.

Welche Levels gibt es?

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- FATAL

Die Dringlichkeit steigt von oben nach unten. Beim Setzen einer Stufe in Logger-Konfiguration legen wir diese Stufe für alle Meldungen in dieser und in allen höheren Stufen fest. Das bedeutet, wenn wir TRACE wählen, bekommen wir absolut alle Meldungen; wenn wir uns wie im Beispiel oben für ERROR entscheiden, bekommen wir nur ERROR und FATAL.

3.2.2 Verteilung der Meldungen: Jedem nach Bedarf

Nun schauen wir uns ein Beispiel an in dem die Vorteile der Technologie zum Vorschein kommen. Wir legen drei Appender (also Ausgabekanäle) an. RootFile vom vorherigen Beispiel lassen wir unverändert und erstellen parallel dazu zwei Weitere. Ein dailyFile und ein Appender für Betriebssystem-Log:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE logging>
<logging:configuration
xmlns:logging="http://www.sas.com/xml/logging/1.0/">
   <appender name="rootFile" class="FileAppender">
      <param name="File" value="C:\Demo\rootFile__root.log"/>
      <param name="Append" value="true"/>
      <param name="ImmediateFlush" value="true"/>
      <layout>
         <param name="ConversionPattern" value="%d{ISO8601} %r [%t]
%-5p %c %u - %m"/>
      </layout>
   </appender>

   <!-- Schreibt für jeden Tag eine neue Logdatei -->
   <appender name="dailyFile" class="RollingFileAppender">
      <param name="Append"         value="true"/>
      <param name="ImmediateFlush" value="true"/>
      <rollingPolicy class="TimeBasedRollingPolicy">
         <param name="FileNamePattern" value="&logPath;\%d.log"/>
      </rollingPolicy>
```

```
      <layout>
         <param name="ConversionPattern" value="%d{ISO8601} %r [%t]
%-5p %c %u - %m"/>
      </layout>
   </appender>

   <!-- Betriebsystem-Log -->
   <appender name="sysLog" class="WindowsEventAppender">
      <layout>
         <param name="ConversionPattern" value="%d: %m"/>
      </layout>
   </appender>

   <!-- Root-Logger. Siehe log4sas-1.xml -->
   <root>
      <appender-ref ref="rootFile" />
      <level value="trace" />
   </root>

   <!-- Neuer Logger für unseren DalyFile-Appender. -->
   <logger name="myOwnLogger">
         <level value="warn" />
         <appender-ref ref="dailyFile" />
   </logger>

   <logger name="sysLogger">
      <level value="fatal" />
      <appender-ref ref="sysLog" />
   </logger>
</logging:configuration>
```

Was ist neu an diesem Beispiel? Wir erhalten zwei neue Appender-Klassen: RollingFileAppender und WindowsEventAppender.

RollingFileAppender erstellt je nach Vorgabe (Element rollingPolicy) zeitgesteuert eine neue Datei. Aus dem Parameter FileNamePattern ergibt sich der Name der Log-Datei. Wenn im Pattern so wie im Beispiel oben ein Platzhalter für Tag auftaucht wird täglich eine neue Log-Datei erstellt. Ein Logger der Klasse WindowsEventAppender fügt die Meldungen zu Windows Ereignissen ein.

An dieser Stelle muss verraten werden, wie man die Meldungen produziert (genauer wird es im nächsten Kapitel behandelt):

```
%LOG4SAS;
%LOG4SAS_FATAL(sysLogger, Houston wir haben ein Problem. So schrei-
ben Sie die Meldungen in Windows Eventlog.);
```

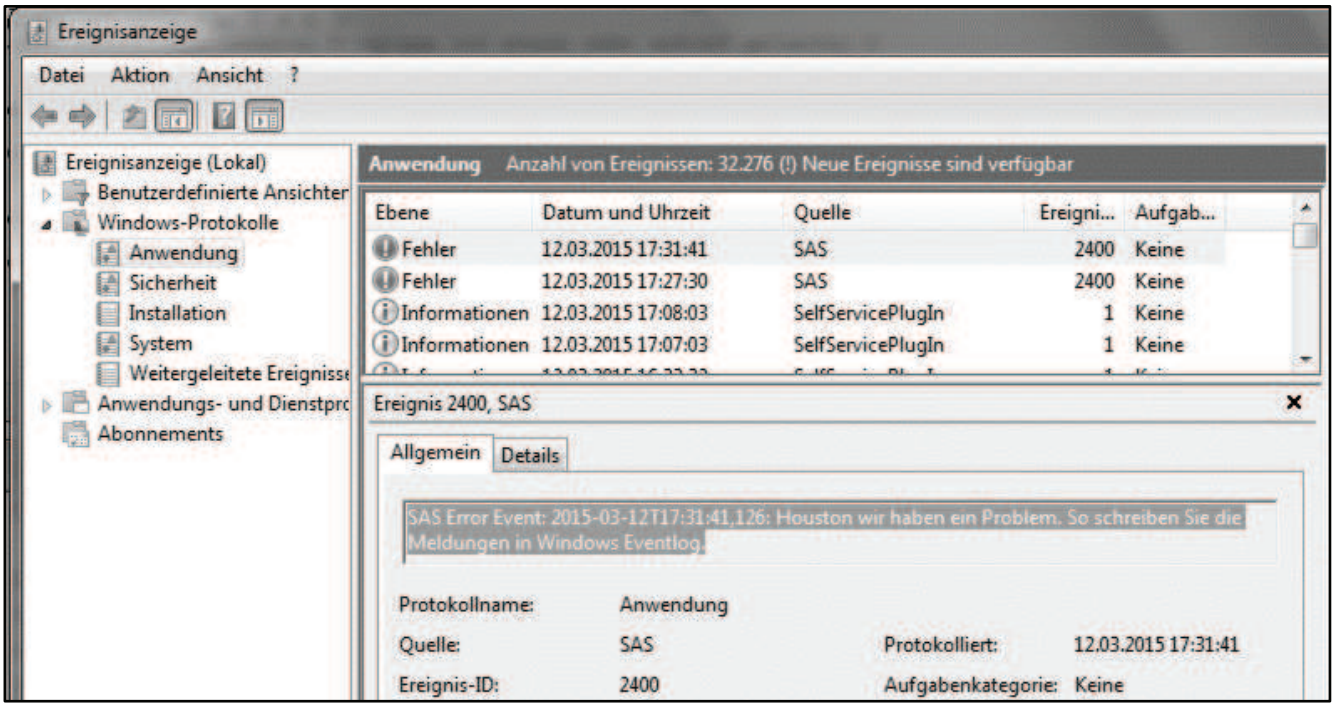


Abbildung 4: Meldungen aus SAS Logging Facility im Windows Event Log

3.2.3 Ausgabe in ein Data Set

Leider steht die Möglichkeit die Meldungen in einen Data Set zu schreiben nur im Serverkontext und nur in bestimmten Logger-Hierarchien zur Verfügung (z.B. Perf.ARM, App, Audit). Doch es gibt eine Möglichkeit, die Meldungen in eine tabellarische Form zu bringen: wir können einen FileAppender als eine CSV-Datei definieren, und im Layout die CSV-Struktur definieren:

```
<appender name="csvFile" class="FileAppender">
   <param name="File"           value="C:\Demo\Logs\csvLog.csv" />
   <param name="Append"         value="false"/>
   <param name="ImmediateFlush" value="true"/>
   <layout>
      <param name="ConversionPattern"
             value="%d{ISO8601};%r;%t;%-5p;%c;%u;%m"/>
   </layout>
</appender>
```

Diese CSV lässt sich dann im Handumdrehen als ein Data Set einlesen. Wenn in den Programmeldungen Semikolons vorkommen sollten, kann man ein anderes Trennsymbol aussuchen, zum Beispiel „|“ (Pipe).

3.2.4 Filterung

Mit dem noch nicht genug – man kann die Meldungen nach Bedarf auch noch filtern. Nicht nur nach dem Level, sondern auch nach einer bestimmten Zeichenkette, außerdem

kann man doppelte Meldungen unterdrücken. Die Filterangaben werden in Appenderdefinitionen untergebracht:

```
<appender name="sysLog" class="WindowsEventAppender">
     <layout>
        <param name="ConversionPattern" value="%d: %m"/>
     </layout>
     <filter class="StringMatchFilter">
          <param name="StringToMatch" value="Abbruch"/>
          <param name="AcceptOnMatch" value="true"/>
     </filter>
     <filter class="DenyAllFilter">
     </filter>
</appender>
```

Filter werden nacheinander angewendet und damit mit logischem „oder“ verknüpft. Damit die Meldungen, die von keinem Filter akzeptiert wurden nicht durchkommen, müssen diese am Schluss mit DenyAllFilter explizit abgewiesen werden.

Die Filterung der Meldungen erfolgt über textlichen Inhalt (siehe Beispiel oben), Meldungsstufe oder -bereich. Außerdem gibt es einen speziellen Filter, der die doppelten Meldungen unterdrückt. Der DenyAllFilter wurde bereits erwähnt. Es gibt auch einen AndFilter, der die andere Filter mit logischem „und“ verknüpft.

4 Meldungen produzieren

4.1 Aufrufe in den SAS-Programmen

Nun ist alles perfekt konfiguriert und Sie wollen unbedingt sehen wie die Meldungen von SAS aus in allen definierten Destinations landen. Das ist im Vergleich zur Konfiguration eine leichte Übung. Die Meldungen lassen sich aus Makros, DATA Step und SCL ausgeben. SLF ist per Default aktiv und jederzeit in Programmen einsatzbereit (MAUTOSOURCE-Option muss aktiv sein).

Außerdem: Je nach Definition und Hierarchie können auch die SAS-eigenen Nachrichten in Ihrer Ausgabe landen. Das ist der Fall, wenn Sie zum Beispiel den Root-Logger modifizieren.

4.2 Aufrufe aus Macro Facility

Bevor man die Meldungen aus Macro Facility ausgibt, muss SLF initialisiert werden. Das wird mit dem Aufruf des Autocall-Macros %LOG4SAS gemacht. Damit der Aufruf nicht vergessen wird, empfehlen wir diesen in der Autoexec unterzubringen.

Beispiel von Aufrufen aus der Makro-Welt:

```
%log4sas;

%log4sas_trace(myOwnLogger, Hallo mein ganz eigener Logger);
%log4sas_error("", Jeder Aufruf an unbekannte Logger wird an den
RootLogger weitergeben);
%log4sas_warn (root, Natürlich können Sie den RootLogger auch direkt
ansprechen);
```

War's das? Ja! Mehr braucht man nicht zu tun. Wenn myOwnLogger und RootLogger korrekt konfiguriert sind, sind die Meldungen nach diesen Aufrufen bereits in den definierten Destinations.

4.3 Aufruf aus DATA Step

```
DATA _null_;
  rc = LOG4SAS_LOGEVENT("MyOwnLogger", "Info", "Wie gewohnt, beim
  Logging innerhalb eines DATA-Steps sind Anführungszeichen
nötig.");

  %LET notiz = Macro-Variablen werden berücksichtigt.;
  rc = LOG4SAS_LOGEVENT("Root", "Info", "Gut zu Wissen: &notiz.");

  Hinweis = "DATA Step Variablen angeblich nicht";
  rc = LOG4SAS_LOGEVENT("MyOwnLogger",  "Error", Hinweis);
  rc = LOG4SAS_LOGEVENT("MyOwnLogger",  "Error", "Gemeint sind aber
  nur die Variablen innerhalb von Quotes: Hinweis");
RUN;
```

Nach der Ausführung landen folgende Zeilen in der Zieldatei von myOwnLogger:
[Zeit] Info - Wie gewohnt, beim Logging innerhalb eines DATA-Steps sind Anführungszeichen nötig.
[Zeit] Error - DATA Step-Variablen angeblich nicht
[Zeit] Error - Gemeint sind aber nur die Variablen innerhalb von Quotes: Hinweis

Achten sie darauf, dass die Variablen in den Quotes nicht aufgelöst werden (Variable Hinweis wird als Text „Hinweis" ausgegeben). Bauen sie ggf. vorab die Meldung zusammen und geben Sie diese dann „am Stück" heraus.

4.4 Aufrufe aus SCL:

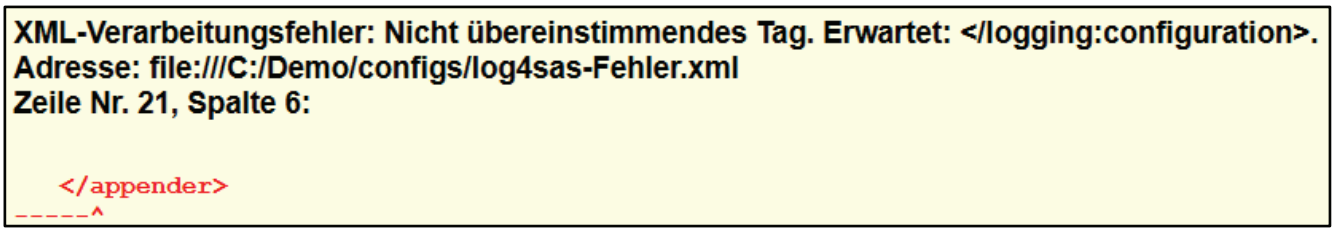
```
DATA _null_;
   if _n_ = 1 then do;
      declare logger logobj("Root");
   end;
   logobj.error("Auch SCL kann LOG4SAS bedienen. Ist das nicht
toll?");
RUN;
```

5 Probleme?

Dann wollen wir uns ein paar der häufigsten Fehler ansehen. Die XML-Datei muss als UTF-8 gespeichert werden. Um festzustellen welche Kodierung die Datei hat, öffnen Sie diese mit Windows-Editor und wählen Sie Datei->Speichern unter. Unten rechts sehen Sie die aktuelle Kodierung:

Ändern Sie diese ggf. auf UTF-8 und speichern Sie die Datei.

Die XML-Datei muss well-formed sein. Ob Ihre XML well-formed ist kann Ihnen jeder beliebige Browser sagen. Wenn die Datei fehlerhaft ist, lässt sie sich nicht öffnen. Beispiel mit Firefox:



6 Fazit

Auch wenn sie bereits ein eigenes Framework geschrieben haben, ist die Umstellung auf SLF empfehlenswert. Die Technologie basiert auf der Philosophie von Open-Source-Framework log4j, das sich über die Jahre zu einem De-facto-Standard für das Loggen schlechthin entwickelt. Wie bei jedem Framework gilt hier das Prinzip „Code less do more“.

Benutzung des SLF bringt viele Vorteile mit sich, darunter:

- Zukunftssicherheit
- flexible Ansteuerung der neuen Logpositionen
- Steuermöglichkeit über die Loglevels und Filter
- Trennung zwischen Protokollierung und Anzeige
- Einfachere Bewältigung der Audit Trail Vorgaben
- Zugriff auf Windows Event Log / Tivoli / Xymon
- Mehr Ergebnisse durch weniger Aufwand

Nun haben sie den Grundstock an SLF-Wissen erworben und können loslegen. Probieren sie es aus!

Literatur

[1] SAS(R) 9.3 Logging: Configuration and Programming Reference