<?xml version="1.0" encoding="utf-8"?> - Über die Tücken eines XML-Datenstroms

Christian Kothenschulte
LBS Westdeutsche Landesbausparkasse
Himmelreichallee 40
48149 Münster
christian.kothenschulte@lbswest.de

Zusammenfassung

In einem konkreten Anwendungsfall soll XML (eXtensible Markup Language) aus einer DB2-Datenbank (unter z/OS) in ein SAS-Dataset überführt und fachlich lesbar werden. Dabei liegt der XML-Strom in einem Tabellenfeld vor. Mittels Connect-To-DB2 kann der XML-Strom in ein Feld eines SAS-Datasets eingelesen werden. Die Informationen liegen dann jedoch weiterhin als XML-Strom vor und sind nicht interpretierbar. Daher müssen die Daten von dort über den Umweg einer (temporären) Datei mit der XML-Engine eingelesen werden. SAS unterscheidet beim Einlesen von XML zwischen einfachen und komplexen Strukturen. Beide Varianten (mit und ohne XMLMap) werden vorgestellt. Der Vortrag zeigt außerdem Lösungsansätze für die Aufgabenstellung auf und geht insbesondere auf die Fallstricke ein, die einem (wahrscheinlich nicht nur bei z/OS) über den Weg laufen. Bei einer möglichen Lösung wurde der SAS XML Mapper zur Erzeugung der XMLMaps genutzt. Dieses Programm wird ebenso vorgestellt, wie weitere Tipps&Tricks.

Schlüsselwörter: XML, XML-Engine, XMLMap, XML Mapper

1 Anwendungsfall

Die LBS Westdeutsche Landesbausparkasse (LBS West) ist seit dem Start von Wohn-Riester im Jahr 2008 Anbieter von Riester-Bausparverträgen.
Die Abwicklung verschiedener Geschäftsvorfälle (z.B.: Zulagenbeantragung, Anbieterwechsel) erfordert eine Kommunikation mit der Zentralen Zulagenstelle für Altersvermögen (ZfA).
Diese Kommunikation erfolgt über verschiedene Meldungen mit unterschiedlichen Meldungstypen, die in XML (eXtensible Markup Language) zwischen der LBS West und der ZfA ausgetauscht werden.

In der LBS West existiert eine DB2-Tabelle auf dem Großrechner unter z/OS mit einem Feld, das einen XML-Strom beinhaltet. Das Feld kann bis zu 30.000 alphanumerische Zeichen aufnehmen. Die DB2-Tabelle wird von einem Cobol-Programm beschrieben.
Ein möglicher XML-Strom sieht so aus:

```
<Daten><Header erstDat="2011-02-25T09:00:26-000" mmMeld="1" anbie-
ter="1234567890" vtNr="0987654321" zfNr="654321"
meGd="XY89"/><RueckzSchaedlEnt refNr="Z1111111111123" stundung="0"
```

```
datumSchaedLVerw="2010-12-11" lfd_ber_nref="9" datumAntrag="2011-01-
12"><Anleger nachname="Kothenschulte" vorname="Christian"
zuNr="10000000K789"><GebDat gebDat="1978-01-
01"/></Anleger><RueckzZulage
waehrung="EUR">123,00</RueckzZulage><RueckzStErm
waehrung="EUR">20,02</RueckzStErm></RueckzSchaedlEnt></Daten>waehrun
g="EUR">20,02</RueckzStErm></RueckzSchaedlEnt></Daten>
```

Der Nachteil der Speicherung der Informationen in einer DB2-Tabelle liegt in der grundsätzlichen Flüchtigkeit (Überschreiben/Löschen) der Zeilen.
Um die ein- und ausgehenden Meldungen mit Stand zum Zeitpunkt der Verarbeitung nachweisen zu können, sollen sie in einer Liste (sequentielle Datei) ausgegeben und archiviert werden.
Das Berichtswesen in der LBS West wird grundsätzlich mit SAS abgewickelt. Daher soll in einer Vorstudie geprüft werden, wie eine Umsetzung der Anforderung mit der vorhandenen Infrastruktur erfolgen kann.
Um diese nutzen können, ist die Breite einer Liste fest vorgegeben und kann nicht beliebig variiert werden.
Bei der Umsetzung soll ein vertretbares Maß an Interpretierbarkeit der Daten ermöglicht werden. Mit Interpretierbarkeit ist das fachliche Verständnis technischer Bezeichnungen (Beispiel aus dem obigen XML-Strom: *</RueckzStErm>* => Rückzahlung Steuerermäßigung) gemeint.
Außerdem sollen die Einträge nach Kerninformationen (z.B.: Vertragsnummer, Meldungstyp) klassifiziert und sortiert werden.

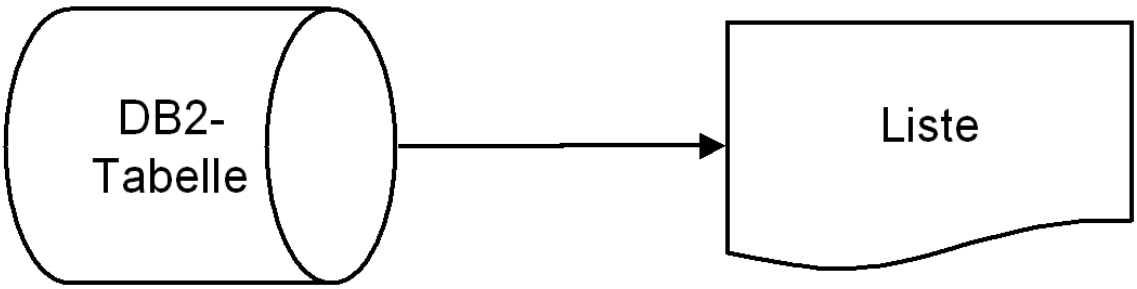

Abbildung 1: Anwendungsfall

2 XML (eXtensible Markup Language)

Übersetzt bedeutet XML „erweiterbare Auszeichnungssprache“ und dient der Beschreibung von Daten in Textform.
Es gibt 3 Komponenten von XML, die sich sehr gut auf verschiedene Rollen verteilen lassen:

- **Schema**: In einem Schema wird vorgegeben, wie eine XML-Datei aufgebaut sein muss.
- **Stylesheet**: Ein Stylesheet gibt vor, wie eine XML-Datei in der Ausgabe angezeigt wird.

- **Inhalt**: In der eigentlichen XML-Datei stehen die konkreten Inhalte.

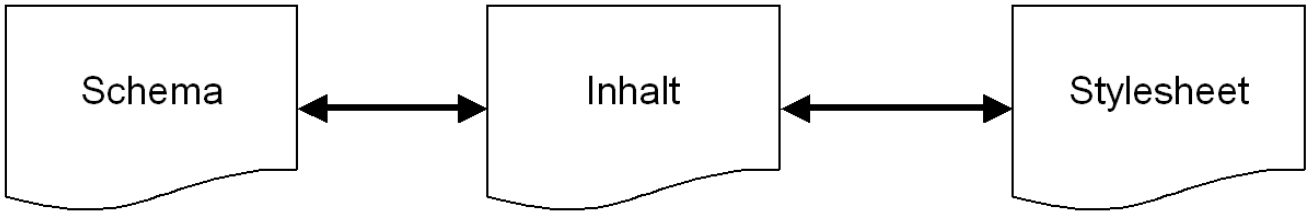

Abbildung 2: Komponenten von XML

Eine Rolle kann den grundsätzlichen Aufbau der Daten vorgeben (Schema).
Das Schema macht die Daten grundsätzlich interpretierbar.
Nicht immer sind die gewählten Tag-Bezeichnungen sprechend (Beispiel: meGd = Meldegrund).
Im Schema können unter anderem Datentypen, Wertemengen und Kommentare zu einzelnen Elementen abgelegt werden.
Eine weitere Rolle kann eine oder mehrere Layout-Vorgaben für unterschiedliche Zielgruppen machen (Stylesheet).
Die dritte Rolle füllt die vorgegebene XML-Struktur mit Daten (Inhalt).
Dabei interessiert es den Inhalt nicht, wie er später (für welche Zielgruppe) angezeigt wird. Er ist also layoutunabhängig.

Außerdem wird XML zum universellen Austausch von Daten zwischen verschiedenen Systemen oder Plattformen verwendet.

XML besteht wie HTML (Hypertext Markup Language) aus verschiedenen Tags.

<Beispiel-Tag></Beispiel-Tag>

Im Unterschied zu HTML gibt es in XML keine festen Namensräume für Tags, da diese bei XML frei wählbar sind.
XML und HTML lassen sich über verschiedene Programme anzeigen. Die gängigsten Programme sind Browser.
Während Browser in der Regel auch fehlerhaftes HTML anzeigen, ist dies bei fehlerhaftem XML nicht der Fall.

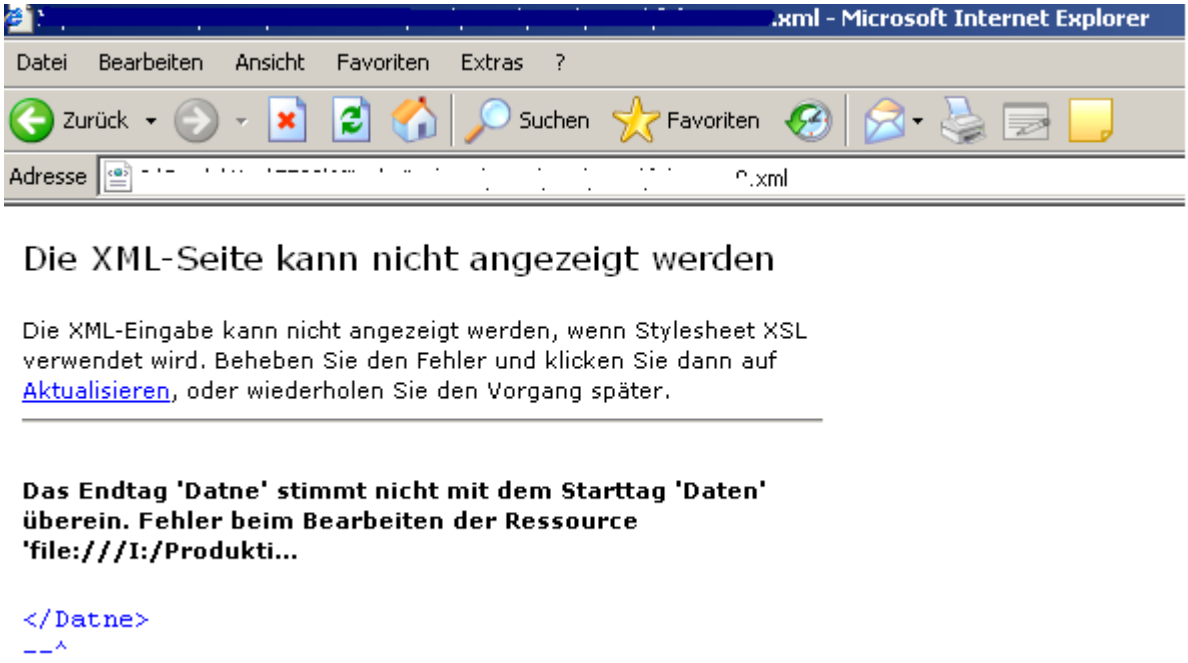

Abbildung 3: Fehlerhaftes XML im Internet Explorer

Gerade weil XML eine hohe Flexibilität aufweist, müssen über 100 Regeln eingehalten werden, z.B.:

- Jedes öffnende Tag benötigt ein schließendes Tag.
- Es darf nur ein Wurzel-Element geben.
- Tags dürfen verschachtelt, jedoch nicht überlappend sein.

Eine einfache XML könnte so aussehen:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- KSFE 2011 in Heidelberg -->
<vortrag>
    <titel>
        Ueber die Tuecken eines XML-Stroms
    </titel>
    <referent id="5">
        Christian Kothenschulte
    </referent>
</vortrag>
```

Die erste Zeile enthält Informationen über die XML-Version und die Codierung. Darauf folgt eine Zeile mit Kommentar.
Das Tag <vortrag> ist das Wurzelelement.
Bei <titel></titel> handelt es sich um ein Element, „Ueber die Tuecken eines XML-Stroms“ ist der Elementeninhalt.
Tags können weitere Informationen hinzugefügt werden. Als Beispiel für ein solches Attribut dient id=“5“. (Attributwerte müssen in Hochkomma angegeben werden.)

XML ist menschenlesbar. Es enthält keine Binärdaten und kann in gewöhnlichen (Text-) Editoren geöffnet werden.

Die universellen Einsatzbereiche und die hohe Flexibilität von XML sind gleichzeitig Vor- und Nachteil. Während man in einer XML-Datei ganze Datenbank-Inhalte ablegen kann, wird der Transfer zu zwei-dimensionalen SAS-Tabellen mit höherer Komplexität der XML-Datei immer aufwendiger.

SAS unterscheidet zwischen einfachem und komplexem XML:

2.1 XML ohne XMLMap

SAS liest XML nur aus Dateien, nicht aus Tabellen oder Feldern.
Zur Nutzung von XML wird die XML-Engine benötigt.
In einem Libname-Statement wird eine logische Bibliothek allokiert, die physikalisch eine (XML-)Datei ist:

```
libname EINXML xml 'D:\EINXML.XML';
```

Eine einfache XML-Datei erzeugt man bequem in einem Data-Step. Als Beispieldaten dient der Inhalt der Tabelle CLASS aus der Bibliothek SASHELP.

VIEWTABLE: Sashelp.Class

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98

Abbildung 4: SAS-Tabelle SASHELP.CLASS

Der Data-Step

```
data EINXML.CLASS;
     set SASHELP.CLASS (obs=3);
run;
```

erzeugt die folgende XML-Datei

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <CLASS>
      <Name> Alfred </Name>
      <Sex> M </Sex>
      <Age> 14 </Age>
      <Height> 69 </Height>
      <Weight> 112.5 </Weight>
   </CLASS>
   <CLASS>
      <Name> Alice </Name>
      <Sex> F </Sex>
      <Age> 13 </Age>
      <Height> 56.5 </Height>
      <Weight> 84 </Weight>
   </CLASS>
```

```
   <CLASS>
      <Name> Barbara </Name>
      <Sex> F </Sex>
      <Age> 13 </Age>
      <Height> 65.3 </Height>
      <Weight> 98 </Weight>
   </CLASS>
</TABLE>
```

Das Einlesen der XML-Datei erfolgt analog:

```
data WORK.CLASS;
     set EINXML.CLASS;
run;
```

2.2 XML mit XMLMap

Die folgende XML-Datei enthält komplexere Strukturen:

```
<Daten>
    <Header erstDat="2011-02-25T09:00:26-000" mmMeld="1"
anbieter="1234567890" vtNr="0987654321" zfNr="654321" meGd="XY89"/>
    <RueckzSchaedlEnt refNr="Z1111111111123" stundung="0"
datumSchaedLVerw="2010-12-11" lfd_ber_nref="9" datumAntrag="2011-01-
12">
        <Anleger nachname="Kothenschulte" vorname="Christian"
zuNr="10000000K789">
            <GebDat gebDat="1978-01-01"/>
        </Anleger>
        <RueckzZulage waehrung="EUR">
            123,00
        </RueckzZulage>
        <RueckzStErm waehrung="EUR">
            20,02
        </RueckzStErm>
    </RueckzSchaedlEnt>
</Daten>
```

Wird diese Datei mit einfachen SAS-Mitteln eingelesen, bricht SAS mit einem Fehler ab:

```
ERROR: XML describe error: XML data is not in a format supported
natively by the XML libname engine. Files of this type usually
require an XMLMap to be input properly.
```

Aus der Fehlermeldung geben zwei Worte Hinweise auf das Problem:

- natively: Das XML aus der Eingabe-Datei ist zu komplex, um es mit dem Standard der XML-Engine einzulesen.
- XMLMap: Für das Einlesen komplexerer XML-Strukturen wird eine XMLMap benötigt.

Eine XMLMap funktioniert wie ein Wörterbuch für SAS und gibt die Struktur vor, in der die XML-Datei aufgebaut ist. Eine XMLMap kann man mit einem Schema (siehe oben) vergleichen.
Eine XMLMap zu der Beispiel-XML-Datei sieht (auszugsweise) so aus:

```
<?xml version="1.0" encoding="windows-1252"?>
<SXLEMAP name="XY89" version="1.2">

 <TABLE name="Header"> [...]</TABLE>

 <TABLE name="RueckzSchaedlEnt">
     <TABLE-PATH syntax="XPath">/Daten/RueckzSchaedlEnt</TABLE-PATH>

     <COLUMN name="refNr">
         <PATH syntax="XPath">/Daten/RueckzSchaedlEnt/@refNr</PATH>
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>14</LENGTH>
     </COLUMN>

     <COLUMN name="stundung">[…] </COLUMN>

     <COLUMN name="datumSchaedLVerw">[…] </COLUMN>

     <COLUMN name="lfd_ber_nref">[…] </COLUMN>

      <COLUMN name="datumAntrag">[…] </COLUMN>
 </TABLE>

 <TABLE name="Anleger"> [...]</TABLE>
 <TABLE name="RueckzZulage"> [...]</TABLE>
</SXLEMAP>
```

Für jedes Element unterhalb des Wurzel-Elements „Daten“ wird ein eigenes Table-Tag erstellt.
Im Unterschied zum einfacheren XML wird die XML-Datei nicht direkt im Libname-Statement allokiert, sondern zunächst in einem Filename-Statement.
Zusätzlich wird die XMLMap in einem Filename-Statement allokiert.
In dem anschließenden Libname-Statement werden beide Dateien verknüpft:

```
filename  XY89 '...\XY89.xml';
filename  SXLEMAP '...\XY89.map';
libname   XY89 xml xmlmap=SXLEMAP;
```

Nun kann jede Tabelle (Table-Tag) separat angesprochen werden:

```
data WORK.Header; set XY89.Header; run;
data WORK.RueckzSchaedlEnt; set XY89.RueckzSchaedlEnt; run;
data WORK.Anleger; set XY89.Anleger; run;
data WORK.RueckzZulage; set XY89.RueckzZulage; run;
data WORK.RueckzStErm; set XY89.RueckzStErm; run;
```

In dem Beispiel gibt es in jeder Tabelle genau einen Satz. Daher können die Tabellen per 1:1-Merge in einem Data-Step zu einer Tabelle zusammengefügt werden:

```
data WORK.XY89;
     merge     WORK.ANLEGER
               WORK.HEADER
               WORK.RUECKZSCHAEDLENT
               WORK.RUECKZZULAGE
               WORK.RUECKZSTERM;
run;
```

3 XML Mapper

Der XML Mapper ist ein in Java geschriebenes Programm, das im SAS Software-Depot enthalten ist.
Das Programm unterstützt bei der Erstellung von XMLMaps.
Der XML Mapper liest XML-Daten (*.XML), XML-Schema (*.XSD) oder XMLMaps (*.MAP) ein.
Mit Drag&Drop-Funktionalität können XMLMaps erzeugt werden. Der vorgegebene Standard-Name „SXLEMAP“ sollte geändert werden, da ein sprechender Name die spätere Nutzung vereinfacht. Außerdem wird so eine Fehlermeldung vermieden, die aber keine Auswirkung hat.

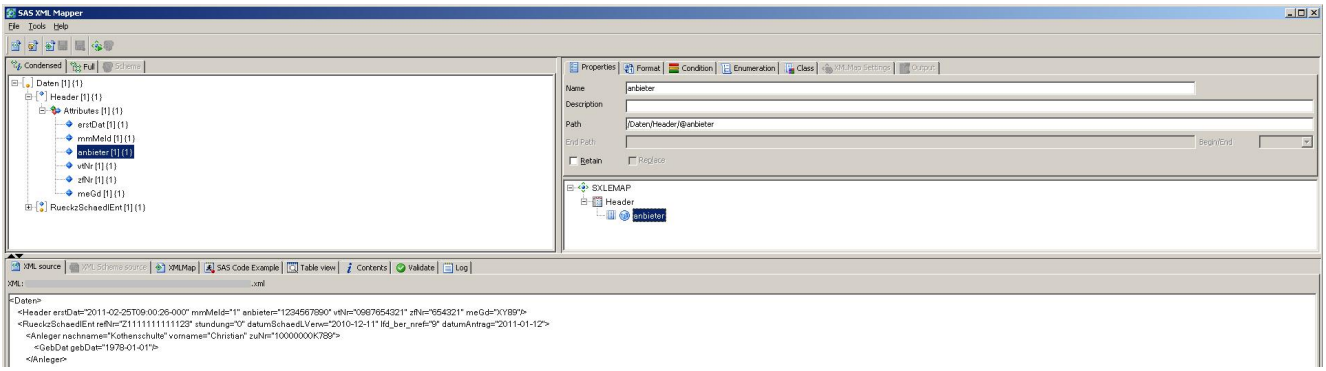

Abbildung 5: XML Mapper

Besonders nützlich sind die Vorschau („Table view“) und der Beispiel-Code („SAS Code Example“).
In der Vorschau wird angezeigt, wie die als Vorlage dienende XML-Datei mit der gerade erzeugten XMLMap eingelesen werden würde.
Der Beispiel-Code enthält die benötigten Filename- und Libname-Statements, die Prozeduren Datasets, Contents und Print, sowie Data-Steps zum Einlesen der XML-Daten. So erhält man direkt einen lauffähigen Programmrahmen zur Verarbeitung der XML-Datei.
Die vorläufigen Ergebnisse der Prozedur Contents können während der Erstellung der XMLMap unter dem Reiter „Contents“ eingesehen werden. Dies ist sehr hilfreich, da so verwendete Formate und Längen schnell identifiziert werden können.
Da bei der Erzeugung von XMLMaps aus eingelesenen XML-Dateien pro Feld immer Format und Länge aus der konkreten Ausprägung ermittelt werden, empfiehlt es sich,

diese Vorgaben zu überarbeiten. Wenn in einem Feld „Nachname“ der Wert „Miller“ steht, wird das Feld auch nur mit einer Länge 5 angelegt!
Bei der Validierung von eingelesenen Dateien kann der XML Mapper 3 unterschiedliche Stufen berücksichtigen (Hoch, niedrig, keine Validierung). Um Folgefehler zu vermeiden, empfiehlt sich die höchste Stufe. Die Ergebnisse der Validierung befinden sich unter „Validate“.
Daneben gibt es einen Reiter „Log“, der alle durchgeführten Aktionen des XML Mappers auflistet.
Sowohl bei der Validierung als auch bei dem Log erhält man mit einem Doppelklick auf einen Eintrag weitere Informationen.
Bei XML-Dateien oder –Schemas mit Verweis auf weitere Dateien oder Rekursion kann es beim Einlesen manchmal/selten zu Problemen („out-of-memory error“, dokumentierter Fehler) kommen.
Insgesamt handelt es sich bei dem XML Mapper um ein einfaches und nützliches Programm, das nach kurzer Einarbeitungszeit bei der Erstellung von XMLMaps unterstützt.
Die sehr gute Dokumentation erreicht man direkt im XML Mapper über [Help] [Help Topics].

4 XML-Druckaufbereitung

Für den Anwendungsfall sollen XML-Meldungen in einer Liste ausgegeben werden.
Dabei sollen die Meldungen nach Kerninformationen klassifiziert werden.
Um sich von der Komplexität des Einlesens und Interpretierens der XML-Daten zu lösen, ist es denkbar, den XML-Strom, der in einer Zeile vorliegt, analog zu gängigen Anzeige-Programmen formatiert (Zeilenumbruch und Einrückung) auszugeben.

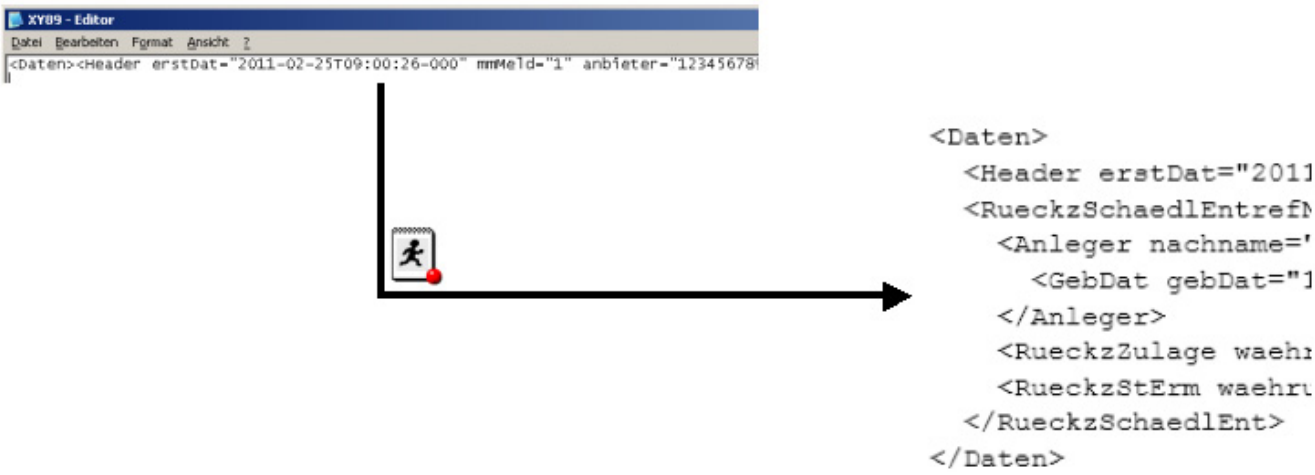

Abbildung 6: XML-Druckaufbereitung

Aus organisatorischen Gründen muss die Infrastruktur auf dem Großrechner (unter z/OS) genutzt werden. Außerdem ist das Angebot an so genannten „Beautifiern“ in diesem Umfeld begrenzt. Es bietet sich hier an, anhand eines selbst geschriebenen Skripts die Vor- und Nachteile eines „Beautifiers“ zu betrachten.

Einen Einstieg für die Entwicklung eines „Beautifiers“ bietet der Programmrahmen im Anhang (siehe *6 Anhang*). Er enthält eine Grundstruktur mit Variablen zur Steuerung, Programmcode und Kommentaren.
Der Rahmen soll nur einen Denkanstoß liefern und ist so bestimmt nicht fehlerfrei und/oder fertig.
Insgesamt bietet ein solches Skript eine vernünftige Aufbereitung des XML-Stroms, kann jedoch keine Inhalte interpretieren oder extrahieren.
Ein Auslesen einzelner Kerninformationen ist nicht möglich. Dafür kann es universell für jeden XML-Strom verwendet werden.
Letztendlich ist es für den Anwendungsfall nur mit Abstrichen geeignet, da die fachliche Interpretierbarkeit fehlt.

5 Lösungsweg / Fazit

Aufgrund der hohen Komplexität der XML-Meldungen kann einfaches XML-Einlesen (ohne XMLMap) in SAS nicht verwendet werden.
Denkbar wäre die Nutzung von XMLMaps.
Einerseits könnte man die Kerninformationen aus der XML-Datei extrahieren und druckaufbereitet ausgeben, andererseits die gesamte XML-Datei einlesen und fachlich interpretiert ausgeben.
Das Druckaufbereiten des XML-Stroms mit einem „Beautifier“ stellt eine einfache Lösungs-Möglichkeit dar, hat jedoch 2 Nachteile: Zum einen werden die Daten nicht interpretiert (kryptische Feldbezeichnungen bleiben erhalten), zum anderen werden keine Kerninformationen ausgelesen.
Grundsätzlich kann die Aufgabenstellung mit SAS unter Verwendung von XMLMaps gelöst werden.
Jedoch sprechen aus organisatorischen Gründen die Menge (>40), die Komplexität und die Änderungszyklen (1 bis 2x pro Jahr) der XML-Meldungen gegen eine Umsetzung mit SAS. Das Verhältnis von Aufwand und Nutzen erfordert die Konzentration der XML-Verarbeitung an einer zentralen Stelle.

Letztendlich soll das Aufbereiten zur Archivierung bereits im verarbeitenden System (ungleich SAS) erfolgen.

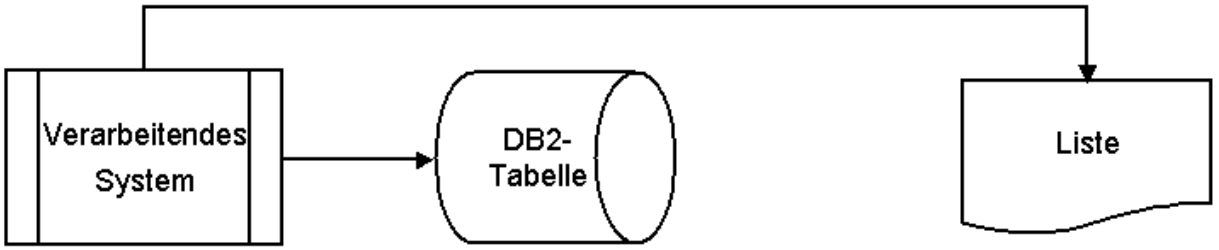

Abbildung 7: Lösungsweg

Da das verarbeitende System beim Lesen oder Schreiben der Meldungen die XML-Inhalte im Zugriff hat, ist es sinnvoll, die gesamte XML-Verarbeitung (Verarbeitungslogik und Archivierung) an dieser einen Stelle zu bündeln.

6 Anhang

```
data work.TEST;
    infile XY89 truncover lrecl=30000;
    input XML_DATEN $30000.;
  /* Initialisieren der Steuervariablen */
  EINRUECKEN = 0;          /* Zaehler fuer Einrueckung   */
  MAXZEILE = 165;          /* Max. Zeichenanzahl/Zeile   */
  LETZTSCHLIESS = 'N';     /* Merker, ob letztes Tag     */
                           /*  ein schliessendes war     */
  SCANMODE = 'J';               /* Schalter, ob gescannt wird  */
  SCANCHAR = ' ';               /* Merker fuer String-Beginn   */
  LAENGEXML = length(XML_DATEN); /* Laenge des XML-Stroms      */
  ZEILE = 0;                    /* Zaehler fuer Pos. in Zeile  */
                           /* Schleife zur Zeichenverarbetiung  */
  do I=1 to LAENGEXML;
                           /* Aktuelles Zeichen ermitteln       */
     AKTZEICHEN = substr(XML_DATEN,I,1);
         /* Wenn das Ende noch nicht erreicht    */
         /* ist, das naechste und die            */
         /* naechsten 2 Zeichen merken           */
         /* Pruefen, ob schliessendes Tag */
         [...]
         /* String Anfang oder Ende liegt vor */
          /* Modus aendern, Zeichen merken */
          /* Pruefen, ob schliessendes Tag */
     [...]
     if (SCANMODE eq 'J' and AKTZEICHEN in ('>')
         and NAECHSTZEICHEN eq '<'
         and (NAECHST2ZEICHEN ne '</'
              or (NAECHST2ZEICHEN eq '</'
                  and LETZTSCHLIESS eq 'J')))
         or (MAXZEILE eq ZEILE) then do;
             /* Zeilenumbruch */
       if SCANMODE eq 'J'
         and AKTZEICHEN in ('>')
         and NAECHSTZEICHEN eq '<'
         and (NAECHST2ZEICHEN ne '</'
            or (NAECHST2ZEICHEN eq '</'
                and LETZTSCHLIESS eq 'J')) then do;
         /* Beruecksichtigung Tag fuer Einrueckung */
         /* Zeichen ausgeben, Zaehler erhoehen */
         /* Zeile einruecken */
         [...]
       end;
       else do;
         /* Aktuelles Zeichen ausgeben, neue Zeile */
```

```
          /* Einruecken und merken */
          [...]
        end;
      end;
      else do;
        [...] /* Aktuelles Zeichen ausgeben, Zaehler erhoehen */
      end;
      [...] /* Zeichen merken */
   end;
run;
```

Literatur

[1] SAS OnlineDoc 9.1.3

[2] SUGI 29 Hands-on Workshops Paper 119-29; Reading and Writing XML files from SAS®; Miriam Cisternas, Ovation Research Group, Carlsbad, CA; Ricardo Cisternas, MGC Data Services, Carlsbad, CA

[3] SAS® XML Mapper to the Rescue; Carol A. Martell, UNC Highway Safety Research Center, Chapel Hill, NC

[4] Margit Becher : XML. W3L-Verlag Witten, 2009