# FHIR2CSV

Vorschlag / Programmierung F. Meineke

Version 28.4.2020

# Einführung

Situation: Statistiker, Biometriker, klinische Forscher (fortan:Nutzer) bevorzugen für die Werkzeuge der Datenanalyse klassische Tabellenformate (e.g.CSV).

Die generische Transformation von FHIR Bundles in Tabellenformate führt zu nicht handhabbaren Anzahlen von Tabellen. Die direkte FHIR Unterstützung in Anwendungen / Programmiersprachen ist noch nicht reif.

Tatsächlich sind praktisch alle Metadaten in der Analyse unnötig, da schon über FHIR Anfrage und Profilierung und MDR auseichend dokumentiert. Beispiel: Ist ein Körpergewicht in „Kg“ spezifiziert ist es nicht nötig codesystem (evtl. UCUM) oder Einheit mit in eine CSV auszuleiten.

Idee: Eine möglichst einfache Spezifikation des gewünschten Datensatzes generiert eine Abfrage auf einem FHIR Bundle, die wiederum eine CSV Datei erzeugt.

**Die resultierenden CSV Daten sind aber nur im Zusammenhang mit den ursprünglichen FHIR Daten und der Spezifikationsdatei zusammen semantisch interpretierbar, da sie selber keine Metadaten mehr enthalten.**

# Beispiele

## Beispiele: Patient

**FHIR Search**: <http://hapi.fhir.org/baseR4/Patient?_format=xml>

**Anfrage**: liefere ID, Geburtsdatum und Geschlecht in einer CSV Datei

**Spezifikation (einfach)**

<table of="Patient”>

<column label="PID" search="id/@value"/>

<column label="GENDER" search ="gender/@value"/>

<column label="BIRTHDATE" search ="birthDate/@value"/>

</tab>

**Spezifikation (mit Komplikation)**

<table of="Patient[gender and birthDate]”>

<column label="PID" search ="id/@value"/>

<column label="GENDER" search ="gender/@value"/>

<column label="BIRTHDATE" search ="birthDate/@value"/>

</tab>

**Bemerkung**

* [gender and birthDate] ist fakultativ; garantiert hier, dass alle Einträge diese Attribute besitzen
* Fun-Fact: Bitte erste Spalte einer CSV Datei nicht „ID“ nennen …. (wenn MS-Excel)

**Ergebnis**

PID;GENDER;BIRTHDATE

641059;male;1977-06-29

641057;male;1951-02-03

641056;male;1951-02-03

## Beispiel: Observation

**Anfrage**: liefere Patienten ID, Gewicht und Erhebungsdatum

**FHIR Search**: <http://hapi.fhir.org/baseR4/Observation?code=http://loinc.org|3141-9&_format=xml>

**Spezifikation (einfach)**

<table of="Observation">

<column label="PID" search ="$patientId"/>

<column label="WEIGHT" search ="valueQuantity/value/@value"/>

<column label="DATE" search ="effectiveDateTime/@value"/>

</tab>

**Bemerkung**:

* Dass es sich um ein Körpergewicht handelt garantiert die Abfrage über den LOINC Code
* $patientId ist hier eine Kurzform für „substring-after(subject/reference/@value,'Patient/')“

**Spezifikation (mit Komplikation)**

<table of="Observation[valueQuantity/unit/@value='Kg']">

<column label="PID" search ="$patientId"/>

<column label="WEIGHT" search ="valueQuantity[unit/@value='Kg']/value/@value"/>

<column label="DATE" search ="substring-before(effectiveDateTime/@value,'T')"/>

</tab>

**Bemerkung**:

* für valueQuantity in @resource wird garantiert, dass zumindest ein Wert in Kg vorliegt
* für valueQuantity in WEIGHT wird garantiert, dass auch nur dieser Wert in Kg genommen wird (und nicht der lb Wert…)
* DATE wurde hier auf Datum reduziert

**Ergebnis**

PID;WEIGHT;DATE

30163;75.9;2019-09-12

30163;77.7;2019-09-05

30163;74;2019-08-29

## Beispiel 3: Observation mit inkludierter Patient Ressource

**Anfrage**: liefere Patienten ID, Geschlecht, Geburtsdatum, Gewicht und Erhebungsdatum

**FHIR Search**: <http://hapi.fhir.org/baseR4/Observation?code=http://loinc.org|3141-9&_include=Observation:patient&_format=xml>

**Spezifikation (einfach)**

<table of="Observation">

<column label="PID" search ="$patient/id/@value"/>

<column label="GENDER" search ="$patient/gender/@value"/>

<column label="WEIGHT" search ="valueQuantity/value/@value"/>

<column label="DATE" search ="effectiveDateTime/@value"/>

</tab>

**Spezifikation (mit Komplikation)**

<table of="Observation[valueQuantity/unit/@value='Kg']">

<column label="PID" search ="$patient/id/@value"/>

<column label="GENDER" search ="$patient/gender/@value"/>

<column label="WEIGHT" search ="valueQuantity[unit/@value='Kg']/value/@value"/>

<column label="DATE" search ="substring-before(effectiveDateTime/@value,'T')"/>

</tab>

**Bemerkung**:

* $patient ist hier eine Kurzform für „//Patient[id/@value=$patientId])“ (letzlich sogar; „//Patient[id/@value= substring-after(subject/reference/@value,'Patient/'])“) ; zeigt auf die eingebettete Patienten Ressource

**Ergebnis**:

PID;GENDER;WEIGHT;DATE

30163;male;75.9;2019-09-12

30163;male;77.7;2019-09-05

30163;male;74;2019-08-29

## Beispiel 4: Bloodpressure mit Components

**Anfrage**: liefere Patienten ID, Sys, Dia und Erhebungsdatum

**FHIR Search**: <http://hapi.fhir.org/baseR4/Observation?code=http://loinc.org|85354-9&_format=xml>

**Spezifikation**

<table of="Observation">

<column label="PID" search ="$patientId"/>

<column label="DIA" search ="component[code/coding/code/@value='8462-4']/valueQuantity/value/@value"/>

<column label="SYS" search ="component[code/coding/code/@value='8480-6']/valueQuantity/value/@value"/>

<column label="DATE" search ="effectiveDateTime/@value"/>

</tab>

**Bemerkung**:

* Die SYS/DIA Werte sind als Component in einer Observation realisiert
* Componente XPATH greift auf jeweilige SYS/DIA Werte zu

**Ergebnis**:

PID;DIA;SYS;DATE

40772;60;107;2019-10-07T08:00:00

40772;62;109;2019-10-07T08:15:00

40772;64;111;2019-10-07T08:30:00

# Verwendete Lösung

Ein Prototyp liegt vor

* eine Stylesheet -Transformation (Generator.xsl) generiert aus der Spezifikation eine weitere Stylesheet-Transformation. Der Generator.xsl selber besteht derzeit aus ca. 60 Zeilen
* Als XSLT Prozessor kommt SAXON (Java) zum Einsatz;
* Generierte Stylesheets (Beispiel 3 derzeit 33 Zeilen) sind sehr klein
* Laufzeit sehr gering (<1s für Beispiele hier)

## Lösung in Java

Benötigt wird eine xslt-Library. Erfahrung derzeit mit saxon-he 9 (siehe <http://saxon.sourceforge.net/>) :

java -jar saxon9he.jar specification.xml Generate.xsl > runner.xsl

java -jar saxon9he.jar FHIR-Bundle.xml runner.xsl > reslult.csv

Prinzipiell ist die Lösung nicht nur Plattformunabhängig sondern auch Java-unabhängig. XSLT Prozessoren sind auch in anderen Sprachen verfügbar. [Wikipedia XSLT-Prozessor]

## Lösung in R

Benötigt wird das xslt Package und das XML2 Package.

Anmerkungen:

* Die libxslt ist etwas älter als saxon und insbesondere ein XSLT 1.0 Prozessort
* Das erforderte einige Anpassungen an das Generator Sytslesheet
* Die libxslt stolpert über die Namesspaces in einer fhir – xml Datei

<Bundle **xmlns="http://hl7.org/fhir**">

* Diese kann man entweder vorher entfernen (so wird es in der Demo gemacht) oder die search-Attribute und die XSLT müssen entsprechend Namespace-Präfixe tragen (hier ohne Demo)

## Vorteilhafte XPATH Lösungen

Innerhalb der Spezifikationen sind beliebige XPATH Ausdrücke möglich:

* Liefere Datum ohne Uhrzeit und Zeitzone
* Ersetze Dezimalpunkte gegen Dezimalkomma

Der Generator sieht folgende Variablen im Kontext der in „tab“ spezifizierten Ressource vor:

* Shortcut um Patient ID aus subject/reference zu extrahieren: $patientId
* Shortcut um Patient Ressource aus subject/reference zu nutzen: $patient

## Vorteile

* XSLT 1.0 ist seit 1999 W3C standardisiert
* Standard für die XML-Verarbeitung
* XSLT ist eine komplette Programmiersprache
* Keine Entwicklung notwendig; keine Tools notwendig; JETZT verfügbar und nutzbar
* Vollständige Trennung von Datenaufbereitung / - ausleitung und Nutzer-Applikation
* Saubere Spezifikation der Transformation

## Kritik

* Warum kein **FHIR Path?** (derzeit nicht klar wo hier hilfreich und ob auch allgemeine Template möglich sind; FHIR Path ist keine Programmiersprache Sprache (?oder?)– anders als XSLT)  Warum FHIR Path, wo wird das eingebettet, wo bietet es Vorteile[Kramer-2017]
* Warum kein gängigen ETL Werkzeuge (TOS o.ä)? --> Die Lösung hier kann vermutlich eingebettet werden (worin?) – derzeit sind keine Vorteile dafür zu erkennen.
* Warum ist die Lösung nicht in GitHub? --> Tja... kommt noch...

## Unsicherheiten

* Grenzen der Verwendung müssen noch ausgelotet werden
* **FHIR Daten müssen in XML (nicht JSON) vorliegen** um mit XSLT; eine einfache nachträgliche FHIR-JSON  FHIR-XML Wandlung ist nicht möglich (?oder?)
* Der Generator ist evtl. unnötig, da die „runner.xsl“ Stylesheets auch menschenlesbar sind und nur wenige Zeilen (< 50) umfassen; damit wäre auch die willkürliche Spezifikationsyntax obsolet; aber Konfiguration ist besser als Programmierung
* Übergabepunkt beim ETL
  + Wie spezifisch formuliere ich den FHIR Search Ausdruck (z.B. Werte mit fehlenden obligatorische Attributen ausleiten oder nicht? )
    - Möchte der Nutzer Datenqualität prüfen benötigt er die “Lücken”
  + Wie zerlege ich meine Anfrage / Wann nutze ich \_include?
  + Transformiere (Datum, Alter, Dezimalpunkte etc) ich bereits in der CSV Wandlung / in der Spezifikation? Oder kategorisch nicht?
  +  vermutlich ist alles obige DUP spezifisch mit dem Nutzer zu vereinbaren.

# Referenzen

[Wikipedia XSL] <https://de.wikipedia.org/wiki/XSL_Transformation>

[Wikipedia XSLT-Prozessor] <https://de.wikipedia.org/wiki/XSLT-Prozessor>

[Kramer-2017] <https://www.devdays.com/wp-content/uploads/2019/02/DD-2017-P-Overview-of-FHIRPath-Ewout-Kramer-2017-11-15.pdf>

[Saxon]: <http://saxon.sourceforge.net/>

[R-XSLT] <https://cran.r-project.org/web/packages/xslt/index.html>

[R-XML2] <https://cran.r-project.org/web/packages/xml2/index.html>

# Anhang R-Reference

import("xml2")

import("xslt")

export("convert")

# found same idea (no implementation) in

# XSLT Cookbook: Solutions and Examples for XML and XSLT Developers von Sal Mangano

# page 533

convert <- function(src,fhir,csv,generator="Generate-for-R.xsl") {

# Create specific Stylesheet for according to table specification

csv\_config <- read\_xml(src, package = "xslt")

generator\_xsl <- read\_xml(generator, package = "xslt")

csv\_xsl <- xml\_xslt(csv\_config, generator\_xsl)

tmpfile = tempfile(pattern ="read\_fhir",tmpdir=".",fileext=".xml")

strip\_namespace(fhir,tmpfile)

fhir\_data <- read\_xml(tmpfile, package="xslt")

file.remove(tmpfile)

# apply generated Stylesheet to fhir data

text=xml\_xslt(fhir\_data, csv\_xsl);

# write csv file

write(text,file=csv)

}

# FHIR Data contains nasty Namespace declarations, making search uncomfortable

strip\_namespace <- function(src,dst) {

# warn=FALSE suppreses missing EOL warning for last line

data\_as\_text <- readLines(src,warn=FALSE)

# Brute force: remove them and write temporary file

data\_as\_text\_no\_namespace = gsub(' xmlns="http://hl7.org/fhir"','',data\_as\_text)

writeLines(data\_as\_text\_no\_namespace,dst)

}