Datastore Import und Export

Diese Seite beschreibt verschiedene Wege, um Daten in einen Datastore zu importieren und zu exportieren.

Grundlagen

- jeder Datensatz hat zwei wichtige Spalten *identifier* und *folder*
- dieser *identifier* muss zusammen mit der Spalte *folder* **eindeutig** sein
- sie müssen sich vor dem Import Gedanken machen, was bei einem Datensatz der *identifier* sein kann
    - **Beispiele**: Artikelnummer, SKU, Auftragsnummer
- *identifier* können Sie sich als ID / Nummer eines Datensatzes vorstellen
- *folder* kann man sich als Ordner, wie in einem Dateisystem vorstellen. *folder* dienen der Strukturierung. z.B. um Datensätze nach verschiedenen Lieferanten zu strukturieren. Jeder Lieferant hätte dabei einen eigenen *folder*
- *folder* kann leer gelassen werden. Dadurch wird *folder* automatisch auf *default* gesetzt.

Name	Beschreibung
folder *(Pflichtfeld)*	Bezeichnung für einen Ordner, zur Gruppierung. Der *folder* ist Teil des Primärschlüssels eines Datensatzes. ***Default-Wert (wenn leer oder nicht vorhanden):*** default
identifier *(Pflichtfeld)*	Eindeutiger Bezeichner des Datensatzes, der zusammen mit **folder** in einem **Datastore** eindeutig sein muss. z.B. Artikelnummer, SKU.

Import mit einem Flow

Um Datensätze zu importieren brauchen Sie einen Flow und den Step SpreadsheetDatastoreWriter.
Dieser Step kann ein Spreadsheet aus einer beliebigen Quelle in einen Datastore schreiben.

> Eine der wichtigsten Einstellungen, die Sie im SpreadsheetDatastoreWriter Step treffen müssen, ist dem Step zu sagen, in welchen Spalten *identifier* und *folder* stehen.

Beispiel: Die Einrichtung des Imports einer CSV-Datei wird in diesem Beispiel ausführlich erklärt.

Einfacher nicht-hierarchischer Import

Wenn wir von "einfachem" Import sprechen, dann meinen wir den Import von sog. flachen Daten. D.h. einzelne Datensätze, die in keiner Beziehung zu anderen Datensätzen stehen.

Fall 1: Import im gleichen Folder

identifier	folder	name
SKU1	default	Schuh 1
SKU2	default	Hose 1

- hier werden 2 verschiedene Datensätze im Folder *default* importiert
- die Spalte *folder* kann auch weggelassen werden. Dann wird automatisch *folder=default* gesetzt.
- beide *identifier sind* unterschiedlich und eindeutig

identifier	folder	name
SKU1	default	Schuh 1
SKU1	default	Hose 1

- dieser Import ist problematisch, weil der *identifier* doppelt vorkommt und zusammen mit dem folder nicht eindeutig ist.
- d.h. der Import wird durchlaufen, aber es kommt zu dem Effekt, dass nicht alle Datensätze angelegt werden
- bei diesem Import wird der zweite Datensatz "gewinnen". D.h. im Datastore wird nur **SKU1, Hose 1** drin stehen, weil der erste Datensatz überschrieben wird
- mit anderen Worten: im gleichen Folder müssen die *identifier* eindeutig sein, ansonsten gewinnt immer der letzte

Fall 2: Import in verschiedene Folder

identifier	folder	name
SKU1	LieferantA	Schuh 1
SKU2	LieferantB	Hose 1

- hier werden 2 Datensätze in 2 verschiedene Folder importiert ("zwei unterschiedliche Produkte von zwei unterschiedlichen Lieferanten")

identifier	folder	name
SKU1	LieferantA	Schuh 1
SKU1	LieferantB	Schuh 1

- hier werden 2 Datensätze in 2 verschiedene Folder importiert ("der gleiche Schuh, aber von 2 unterschiedlichen Lieferanten")
- obwohl beide *identifier* gleich sind funktioniert dieser Import: das liegt daran, dass beide Datensätze in unterschiedlichen Foldern liegen

> **Zur Erinnerung**
>
> Zur Erinnerung: im gleichen folder müssen identifier unterschiedlich sein. Sind die folder unterschiedlich, dann können identifier gleich sein

Hierarchischer Import Parent-Variant

Wenn wir von hierarchischen Daten bzw. Parent-Variant Import sprechen, dann meinen wir damit den Import von Daten, zwischen denen es eine Eltern-Kind Beziehung (z.B. Stamm-Artikel und dazugehörige Varianten) gibt und die sich im gleichen Datastore befinden.

Bei hierarchischen Daten kommt zum *identifier* und *folder* noch eine weitere wichtige Spalte hinzu: der *parent_identifier*

parent_identifier *(Optional)*	Der identifier des referenzierten Parent Datensatzes im gleichen Datastore und Folder. Wenn gesetzt, dann wird dieser Datensatz mit dem Parent verknüpft und bildet einen Datensatz vom Typ "*Variant*". **Default-Wert:** <leer>

Fall 3: Import im gleichen Folder

identifier	folder	parent_identifier	name
SKU1	default		Schuh 1
SKU1-red	default	SKU1	Schuh 1 rot
SKU1-blue	default	SKU1	Schuh 1 blau
SKU2	default		Hose 1
SKU2-S	default	SKU2	Hose 1 Größe S
SKU2-M	default	SKU2	Hose 1 Größe M
SKU2-L	default	SKU2	Hose 1 Größe L

- hier werden insgesamt 7 Datensätze in den gleichen Datastore und im gleichen *folder* importiert
- 2 Datensätze (SKU1 und SKU2) bezeichnen wir als sog. Parent-Datensätze bzw. *parents*
- SKU1 hat 2 Varianten-Datensätze bzw. *variants*
- SKU2 hat 3 *variants*
- Jede Variante wird über die Spalte *parent_identifier,* dem *parent* zugeordnet
- beim *parent* muss die Spalte *parent_identifier* leer sein
- bei *variant*-Zeilen beobachtet man häufig, dass der *identifier* zusammengesetzt ist z.B. Artikelnummer des *Parent*, Bindestrich und dann eine Nummer oder Attribut der *Variante.*

Fall 4: Import in verschiedene Folder

identifier	folder	parent_identifier	name
SKU1	LieferantA		Schuh 1
SKU1-red	LieferantA	SKU1	Schuh 1 rot
SKU1-blue	LieferantA	SKU1	Schuh 1 blau
SKU1	LieferantB		Schuh 1
SKU1-red	LieferantB	SKU1	Schuh 1 rot
SKU1-blue	LieferantB	SKU1	Schuh 1 blau

- hier werden insgesamt 6 Datensätze im gleichen Datastore aber in 2 verschiedene *folder* importiert ("der gleiche Schuh, aber von 2 verschiedenen Lieferanten")
- 2 *parents* mit jeweils 2 *variants*

Hierarchischer Import Master-Child

Vorab: *Master-Child* ist sehr ähnlich zu *Parent-Variant*. Beide Konzepte beschreiben eine Eltern-Kind-Beziehung, aber mit folgenden Unterschieden:

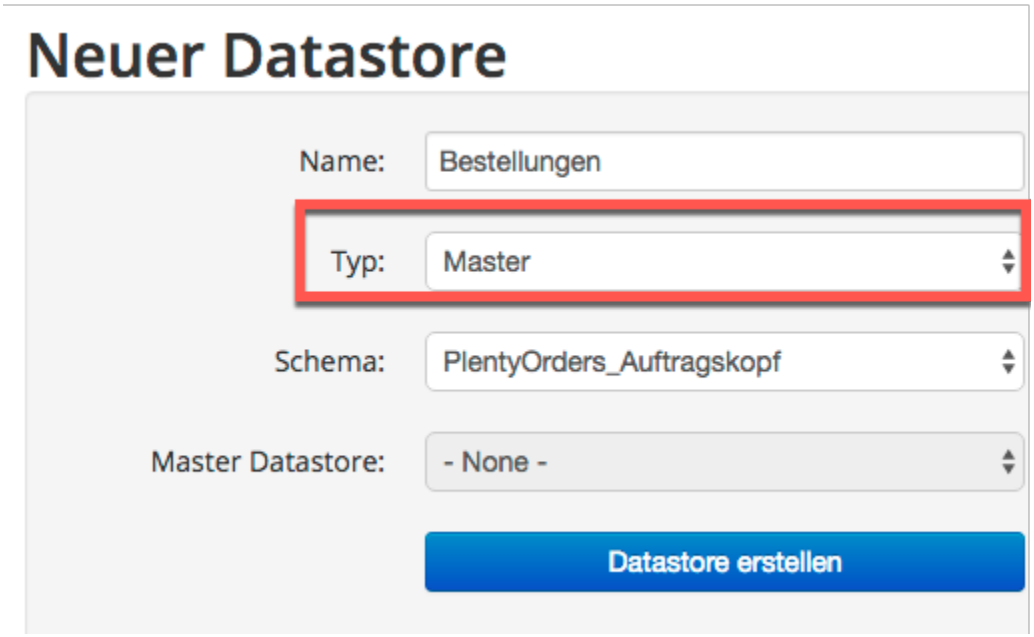
- Bei *Parent-Variant* liegen die Datensätze im *selben* Datastore
  - dadurch haben Parent- und Variant-Datensätze das gleiche Schema
- bei Master-Child liegen Datensätze in *unterschiedlichen* Datastores
  - dadurch können Master- und Child-Datensätze unterschiedliche Schemas verwenden

Wir haben uns für diese 2 unterschiedlichen Begriffe entschieden, um diese beiden Konzepte auseinander halten zu können.

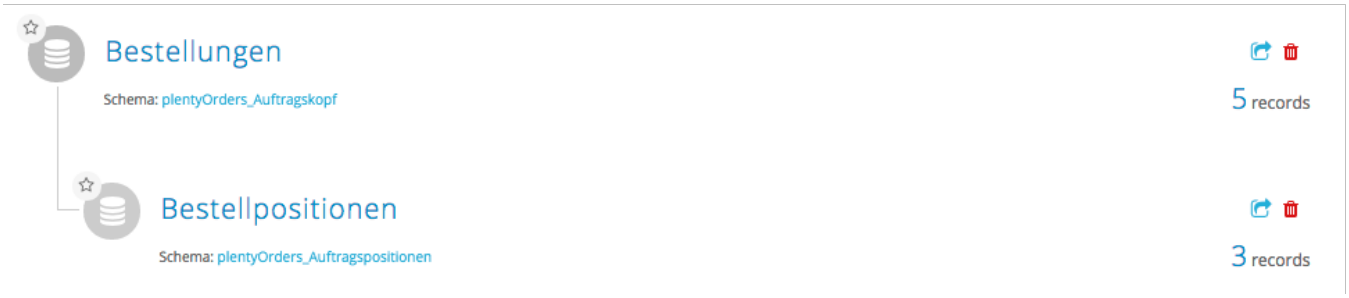
Bei Master-Child spielt die Spalte *master_identifier* die wichtige Rolle, um Child-Datensatz mit dem Master-Datensatz zu verknüpfen. (*master_identifier ist bei Master-Child das, was bei Parent-Variant der parent_identifier ist*)

Voraussetzung

Um mit Master-Child zu arbeiten sind 2 Datastores notwendig, und zwar ein **Master-Datastore** und ein **Child-Datastore**.



In der Datastore Ansicht, werden Master-Child Datastores eingerückt dargestellt.



Bestellpositionen ist der Child-Datastore vom Master-Datastore Bestellungen.

Fall 5: Import in Master und in Child Datastore

Beim Import in 2 getrennte Datastore braucht man auch 2 *DatastoreWriter* Steps.
Zuerst müssen die Master-Datensätze in den Master-Datastore geschrieben werden, danach die Child-Datensätze in den Child-Datastore.
Dazu brauchen Sie 2 Spreadsheets. Ein Spreadsheet beinhaltet die Master-Datensätze, das zweite die Child-Datensätze.

Master-Spreadsheet

identifier	folder	master_identifier	name
Auftrag1	default		Bestellung 1
Auftrag2	default		Bestellung 2

- das Master-Spreadsheet beinhaltet nur die Daten für den Master-Datastore
- die Spalte *master_identifier* muss leer bleiben

Child-Spreadsheet

identifier	folder	master_identifier	name
Auftrag1-SKU1-red	default	Auftrag1	Schuh 1 rot aus Auftrag 1
Auftrag1-SKU2-M	default	Auftrag1	Hose 1 Größe M aus Auftrag 1
Auftrag2-SKU2-L	default	Auftrag2	Hose 1 Größe L aus Auftrag 2

- das Child-Spreadsheet beinhaltet nur die Daten für den Child-Datastore
- über die Spalte *master_identifier* wird der Child-Datensatz mit dem Master-Datensatz verknüpft

Ein Flow, der *Master-Spreadsheet* und *Child-Spreadsheet* importiert hat meistens folgenden Aufbau:

- über 2 SpreadsheetFilter wird ein Input-Spreadsheet (beliebige Quelle) in Master-Datensätze und Child-Datensätze aufgeteilt.
- die beiden SpreadsheetDatastoreWriter Steps schreiben dann jeweils die Ergebnisse der SpreadsheetFilter Steps in den Datastore
  - das Master-Spreadsheet in den Master-Datastore, und das Child-Spreadsheet in den Child-Datastore

Optionale weitere identifier-Spalten

Jeder Datensatz verfügt noch über 2 weitere identifier-Spalten *identifier2* und *identifier3*.
Diese Spalten sind optional und können mit weiteren IDs/Nummern belegt werden, die für einen Datensatz wichtig sind. Sie spielen bei der Verwendung mit dem Querverweis eine wichtige Rolle.
identifier2 und *identifier3* werden häufig dazu verwendet z.B. EAN und Lieferanten-Artikelnummer zu einem Datensatz zu speichern - mit anderen Worten IDs eines Drittsystems. Mit dem Querverweise kann man dann den *identifier* ermitteln, welcher beim Import zum Update eines Datensatzes notwendig ist.

Video Tutorial

Ein Videotutorial zum Thema Querverweis findet man hier. Ausserdem gibt es weitere Infos im Handbuch.

Beispiel: identifier per Querverweis über identifier2 herausfinden

Nehmen wir an, wir importieren Artikelstammdaten in einen Datastore.

- ArtikelNummer = SKU1
- EAN = 1234567891234
- Produkttitel
- Preis

identifier	identifier2	name	preis
SKU1	1234567891234	Schuh 1	59,99 EUR

Nun nehmen wir an, Sie bekommen von einem Lieferanten täglich eine Datei mit aktuellen Preisen, in der folgendes enthalten ist:

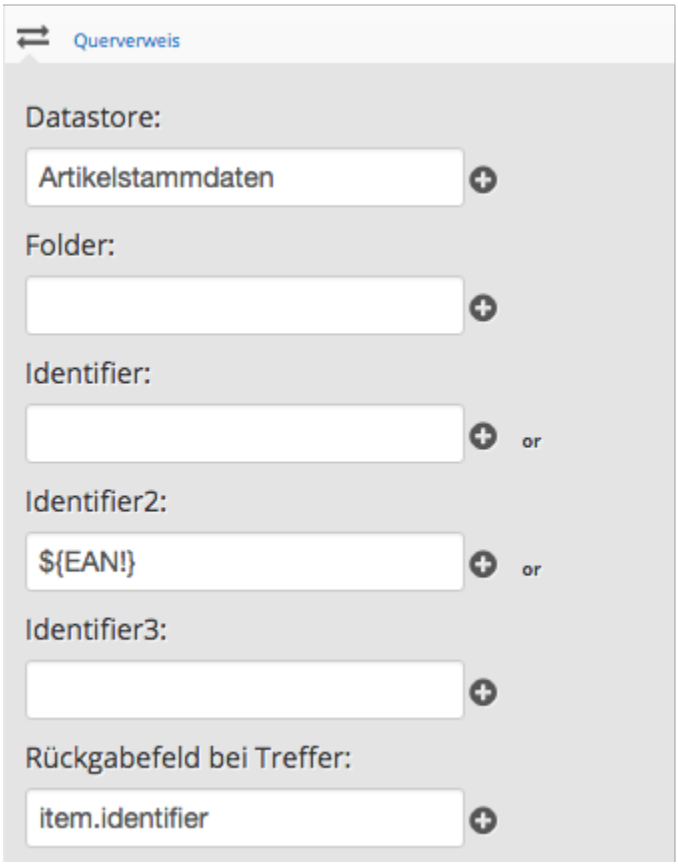
- EAN
- Preis

EAN	Preis
1234567891234	49,99 EUR

Diese Lieferantendatei beinhaltet nur die EAN, nicht aber ihre eigenen Artikelnummer, die im *identifier* gespeichert ist.
Nehmen wir weiter an, Sie möchten die Lieferantendatei um die Spalte mit Ihrer eigenen Artikelnummer anreichern wie folgt:

EAN	Preis	identifier
1234567891234	49,99 EUR	SKU1

Um das zu erreichen, nutzt man man die Querverweis-Funktion im SpreadsheetMapper. (Hinweis für Excel-Nutzer: Der Querverweis in Synesty Studio ist vergleichbar mit einem SVerweis in Excel)



Die folgende Spaltenkonfiguration liest sich wie folgt:

- gib mir aus dem Datastore "Artikelstammdaten"
- das Feld *identifier,*
- des Datensatzes in dem in *identifier2* die EAN steht.

In SQL übersetzt könnte es sich so lesen:

```
SELECT identifier FROM Artikelstammdaten WHERE identifier2 = $EAN
```

Der Querverweis liefert als Ergebnis in dieser Spalte für jede Zeile den identifier (unsere interne Artikelnummer, also im Beispiel SKU1).
Sollte es keinen Treffer geben (d.h. es gibt keinen Datensatz bei dem in identifier2 die gesuchte EAN steht), dann gibt die Querverweis-Funktion ***No Record found*** in der entsprechenden Spalte aus.

EAN	Preis	identifier
1234567891234	49,99 EUR	SKU1
987665 (nicht im Datastore)	19,99 EUR	No Record found

Mit diesem angereicherten Ergebnis-Spreadsheet kann man nun weiterarbeiten und z.B. ein Preis-Update im eigenen Shop machen.
(Hinweis: Oft ist es so, dass Shop- oder ERP-Systeme die eigene interne Nummer zum Abgleich brauchen. D.h. den identifier statt der externen EAN. Da externe Lieferanten aber niemals diese interne Nummer kennen, sondern meistens EAN schicken, ist der Querverweis notwendig. Voraussetzung ist natürlich, dass die EAN auch im Datastore hinterlegt ist.)

Wie kann ich Zeilen mit *No Record found* ignorieren / herausfiltern?

Um alle Zeilen zu entfernen, in denen in einer Spalte der Wert ***No Record found*** des Querverweises auftaucht, benutzt man den SpreadsheetFilter.

Ein SpreadsheetFilter mit dieser Filterbedinung lässt alle Zeilen durch, die im *identifier* nicht "*No Record found*" enthalten - mit anderen Worten: entferne alle Zeilen, in denen im *identifier* "*No Record found*" steht.

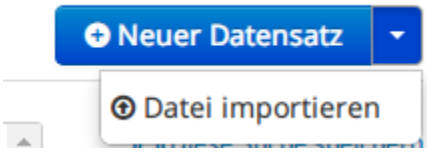
Duplikate

identifier2 und *identifier3* dürfen Duplikate enthalten (im Gegensatz zu *identifier*). Sie müssen hier selbst darauf achten, wie Sie mit Duplikaten umgehen. Beim Querverweis können Duplikate problematisch sein, da der Querverweis bei mehreren Treffern nur eines zurück gibt. Bei Duplikaten ist es dann Zufall, welcher Datensatz als Treffer gewählt wird.

Wenn Sie herausfinden möchten, ob Sie evtl. Duplikate in *identifier2* oder *identifier3* haben, dann können Sie das mit den Spaltenfunktionen herausfinden.

Manueller Import ohne Flow

Datastores verfügen auch über eine manuelle Importfunktion für CSV- oder Exceldateien. Um eine Datei zu Importieren gehen Sie in den gewünschten Datastore und wählen im Dropdown Menü des "Neuer Datensatz" Buttons die Option "Datei importieren" .



Wählen Sie anschließend die Datei aus und klicken den "Upload und Vorschau" Button. Nachdem die Datei verarbeitet wurden gelangen Sie in die Transformy Ansicht, mit allen Spalten des Datastores. Sie können nun die Quell-Spalten aus der Datei den entsprechenden Spalten des Datastores zuordnen. Über den 

Button starten Sie den Import der Datensätze.

Datenexport mit Flow

Wenn man Daten in einen Datastore hinein importiert, dann will man meistens auch diese Daten wieder heraus holen - sprich exportieren.

Dafür verwendet man den Step **SearchDatastore.**

Ein typischer Export-Flow, welcher Daten aus einem Datastore exportiert hat folgenden Aufbau:

- **SearchDatastore** (bietet die gleichen Suchoptionen wie die Datastore Suche)
- **Mapper SpreadsheetMapper** (Konfiguration der Export-Spalten mit Transformy)
- **SpreadsheetCSVWriter** (erzeugt CSV-Datei aus dem zuvor konfigurierten Spreadsheet)
- z.B. **FTPUpload** (lädt die Datei auf einen FTP-Server)

Datensätze vom DatastoreWriter lesen

Datensätze vom DatastoreWriter lesen

Wenn Sie durch einen *DatastoreWriter* geschriebenen Datensätze abrufen möchten, verwenden Sie bitte die entsprechenden Output-Spreadsheets des DatastoreWriters wie z.B. *newAndUpdatedRecords*.

Verwenden Sie dafür NICHT den SearchDatastore-Step!
Warum nicht? Unsere Datenbank, eine sog. NoSQL Datenbank, arbeitet "eventual consistent". Das heißt, dass es vorkommen kann, dass die gerade geschriebenen Datensätze nicht sofort auf alle Datenbanken und Indizes geschrieben wurden. Damit kann es vorkommen, dass der SearchDatastore Step noch veraltete Daten erhält. Der Vorteil dieses Verhaltens ist allerdings, dass der *DatastoreWriter* dafür sehr schnell importiert. Verwenden Sie daher zum Lesen eigener Schreibvorgänge (sog. *Read your own writes*) ausschließlich die erwähnten Output-Spreadsheets.

Export hierarchischer Daten mit Zugriff auf Parent / Master Zeile

Im genannten *SpreadsheetMapper* Step können Sie die zu exportierenden Spalten bearbeiten und haben bei hierarchischen Daten darüber hinaus auch noch die Möglichkeit auf die Parent-Zeile bzw. Master-Zeile (bei Child-Datatstore-Zeilen) zuzugreifen.

Hierarchische Daten mit Zugriff auf Parent / Master Zeile exportieren

Wir empfehlen auch den Abschnitt Spreadsheets - Auf Relationen zugreifen, der beschreibt, wie man beim Export hierarchischer Daten auf die Parent- bzw. Master-Zeile zugreift. Das ist nützlich wenn man z.B. Auftragspositionen mit Daten aus dem Auftragskopf anreichern möchte.

Beispiel Export Auftragsdaten als XML

Ein typisches Beispiel sind Auftragsdaten (Bestellungen) und den dazugehörigen Auftragspositionen (bzw. Bestellpositionen).

Das Skript-Beispiel geht von folgendem Szenario aus:

- Daten kommen aus einem SearchDatastore Step mit *showChildren=false* (das ist **wichtig:** Aus dem SearchDatastore dürfen nur die Auftragskopfdaten raus kommen. Auf die Auftragspositionen kann man trotzdem zugreifen per zweiter Schleife *row.children())* Das klingt zwar etwas unlogisch, liegt aber daran, dass man hier Spreadsheet und Freemarker Script kombiniert. Im Spreadsheet (das was man in der Vorschau sieht), darf nur die Kopfzeile rauskommen. Per Skript kommt man dann aber auch an die Children-Zeilen, um diese auszugeben).
- Die Auftragsdaten sind in 2 Datastores abgelegt, die als Master-Child Datastores verknüft sind

```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrders>

<#list spreadsheet@SearchMasterDatastore_1.getRows() as row>

<#if (row.children()?? && row.children()?size > 0) >
        <Name>${row.get("shipping_address_given_name")!} ${row.get("shipping_address_family_name")!}</Name>
        <Street>${row.get("shipping_address_address1")!}</Street>
        <City>${row.get("shipping_address_city")!}</City>
        <Zip>${row.get("shipping_address_zip")!}</Zip>
        <Country>${row.get("shipping_address_country")!}</Country>
    <Items>

    <#list row.children() as ch>

       <Item PartNumber="${ch['line_items_sku']!}">
          <ProductName><![CDATA[${ch['line_items_title']!}]]></ProductName>
          <Quantity>${ch['line_items_quantity']!}</Quantity>
          <Price>${ch['line_items_price']!}</Price>
          <Currency>${row.get("currency")!}</Currency>
       </Item>
   </#list>

    </Items>
 </PurchaseOrder>
</#if>

</#list>
</PurchaseOrders>
```

Das entscheidende sind die beiden verschachtelten Schleifen (<#list>).

- *<#list spreadsheet@SearchMasterDatastore_1.getRows() as row>* iteriert über die Daten des Master-Datastore, der die Auftragskopfdaten enthält
- *<#list row.children() as ch>* greift auf die Children-Zeilen, also die Auftragspositionen der jeweiligen Kopfzeile zu und iteriert darüber.

Das komplette Beispiel findet sich in der Vorlage Hierarchische XML oder JSON Datei aus Spreadsheet erzeugen (komplexeres Beispiel).