JSON2Spreadsheet

Mit dem JSONReader Step kann man eine oder mehrere JSON-Dateien einlesen und als Spreadsheet ausgeben. Oft wird dieser Prozess auch als parsen oder parsing bezeichnet. Das JSON-Format wird oft in Verbindung mit sog. REST-APIs verwendet.

JSON-Einlesen ohne Skripting

Für einfach strukturierte JSON-Dateien kann man auch den Step VisualJSON2Spreadsheet verwenden. Dieser macht die Verarbeitung durch eine graphische Oberfläche noch einfacher.

Tutorial: REST-API Anbindung

Lernen Sie in unserem Tutorial REST API Anbindung mit Synesty, wie man jede beliebige HTTP-API mit JSON oder XML anbinden kann - auch ohne Add-Ons von Synesty und ohne Programmierung in PHP, Java oder Javascript.

Beispiel

Im Beispiel werden wir einen Flow erstellen, der eine JSON-Datei von einer URL per HTTP herunter lädt, diese einliest und als CSV-Datei umwandelt. Der resultierende Flow kann wie folgt aussehen:

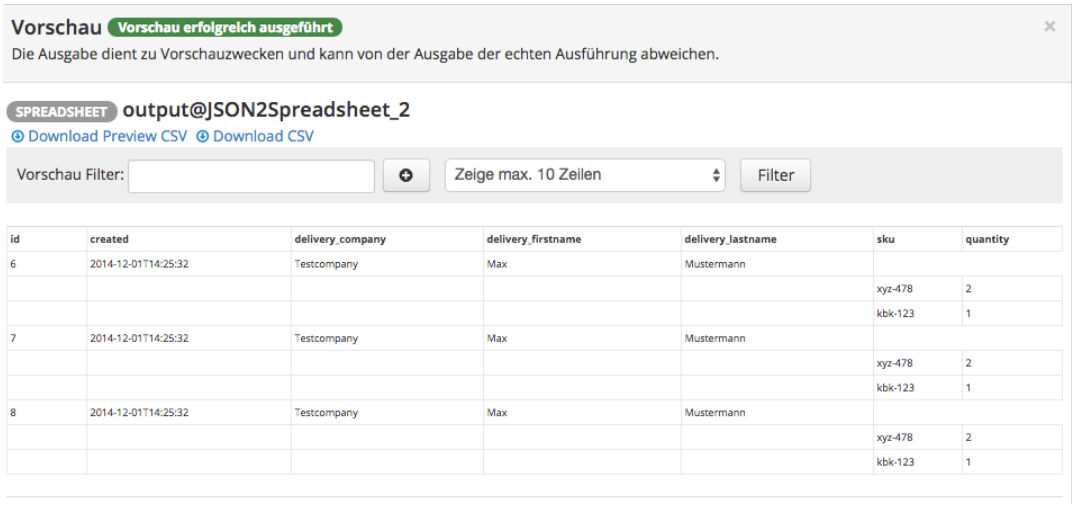
Quelldatei im JSON-Format

Gegeben sei folgende JSON-Datei mit Bestelldaten (2 Bestellungen mit jeweils 2 Positionen):

```
{
    "results": [
        {
            "id": 1,
            "created": "2014-12-01T14:25:32",
            "delivery_company": "Testcompany",
            "delivery_firstname": "Max",
            "delivery_lastname": "Mustermann",
            "order_rows": [
                {
                    "sku": "xyz-478",
                    "quantity": 2
                },
                {
                    "sku": "kbk-123",
                    "quantity": 1
                }
            ]
        },
        {
            "id": 2,
            "created": "2014-12-01T14:25:32",
            "delivery_company": "Testcompany",
            "delivery_firstname": "Max",
            "delivery_lastname": "Mustermann",
            "order_rows": [
                {
                    "sku": "xyz-478",
                    "quantity": 2
                },
                {
                    "sku": "kbk-123",
                    "quantity": 1
                }
            ]
        }
    ]
}
```


Ziel nach dem Einlesen

Die Daten sollen eingelesen werden, so dass diese als Tabelle (Spreadsheet) zur Verfügung stehen:



JSON-Einlesen Variante 1

Um JSON-Datei einzulesen benötigen Sie den JSON2Spreadsheet Step mit folgendem Parsing-Skript:

```
<#assign row = target.addRow()>

<#list json["results"] as r >

        <#assign row = target.addRow()>
        ${addColumns(row, r)}

        <#list r["order_rows"] as o>

                <#assign orderRow = target.addRow()>
                ${addColumns(orderRow, o)}

        </#list>

</#list>
```

Dieses Parsing-Skript nutzt die Helfer-Funktion *addColumns(),* welche automatisch versucht Unterknoten des JSON-Knotens als Spalten hinzuzufügen.

Parameter von *addColumns()*

Syntax:

```
${addColumns(row, node, [prefix], [options])}
```

Hinweis: Der *options* Parameter ersetzt die alten Konfigurationsparameter *columnsArray* und *mode*. Bitte ab 26.04.2017 nur noch die hier aufgeführte Art der Konfiguration nutzen.

Parameter		Beschreibung
row		Die Zeile (das Row-Objekt des Spreadsheets), der die Spalten hinzugefügt werden sollen.
node		Der Ober-Knoten (Node) im JSON-Baum, dessen Unterknoten als Spalten hinzugefügt werden sollen.

prefix		optional: Dieser optionale Prefix wird allen hinzugefügten Spaltennamen vorangestellt.
		Beispiel:
		`${addColumns(row, j['shipping_address'], "shipto_address_")}`
		`${addColumns(row, j['billing_address'], "billto_address_")}`
		Hier werden Spalten eines JSON-Objekts *shipping_address* und *billing_address* hinzugefügt. Da beide Adressen Spalten wie *Vor name*, *Nachname* enthalten können, ist es sinnvoll diese durch ein Prefix (hier *shipping_address_* und *billto_address_*) von einander abzugrenzen und unterscheidbar zu machen.

options		Ein Hash/Map (Schlüssel/Wert-Paare) mit optionalen Konfigurationsparametern. Die Notation der *options* ist technisch gesehen ein JSON-Objekt.
		Einige Parameter gehören zusammen (z.B. *columns* und *mode)*, einige stehen für sich allein oder greifen nur unter bestimmten Bedingungen. Es wird versucht in der Beschreibung deutlich zu machen, wann ein Paramter anwendbar ist.
		
		Beispiel:
		Hier werden die beiden Parameter *columns* und *mode* dargestellt, die etwas weiter unten noch genauer erklärt werden.
		
		2 Parameter: columns und mode
		
		`{"columns":["zip", "city"], "mode":"include"}`
		
		`# Etwas lesbarer formatiert, sieht das so aus:`
		`{`
		`"columns":[`
		`"zip",`
		`"city"`
		`],`
		`"mode":"include"`
		`}`
		
		D.h. ein kompletter *addColumns*-Befehl inkl. *options* sieht wie folgt aus:
		
		`${addColumns(row, order['shipping_address'], "", {"columns":["zip", "city"], "mode":"include"})}`
		
		Auf die einzelnen Optionsparameter wird im folgenden eingegangen.
		
		Wichtig
		
		Wird **options** ohne den **prefix** Parameter verwendet, dann ist für prefix ein Platzhalter in Form von "" einzufügen.
		
		Beispiel: ${addColumns(row, order['shipping_address'], "", {"columns":["zip", "city"], "mode":"include"})}

	columns *(optional)*
	
	&
	
	mode *(optional)*
		columns: Ein String-**Array** von Feldnamen, die im Spreadsheet *hinzugefügt* (include) oder ausgeschlossen (exclude) werden sollen.
		
		mode: Über *mode* kann man steuern, ob die im Parameter *columns* definierten Spalten ausgegeben oder ignoriert werden.
		
		• *include* (default) - die Spalten in *columnsArray* werden ausgegeben
		• *exclude* - die Spalten in *columnsArray* werden ignoriert / ausgeschlossen
		
		Beispiel: Nur bestimmte Feld ausgeben (include)
		
		`${addColumns(row, order['shipping_address'], "shipto_address_", {"columns":["zip", "city"]})}`
		
		Diese Zeile sorgt dafür, dass nur die Felder **zip** und **city** des Knotens *shipping_address* im Spreadsheet landen. Alle anderen Address-Felder wie z.B. *street* oder *county* werden nicht ausgegeben.
		Den optionalen Parameter **mode** kann man weglassen, da dieser per Default auf "include" steht. D.h. man hätte obige Zeile auch so schreiben können:
		
		`${addColumns(row, order['shipping_address'], "shipto_address_", {"columns":["zip", "city"], "mode":"include"})}`
		
		Beispiel: Bestimmte Felder ausschließen (exclude)
		
		`${addColumns(row, order['shipping_address'], "shipto_address_", {"columns":["zip", "city"], "mode":"exclude"})}`
		
		Diese Zeile sorgt dafür, dass alle Felder außer **zip** und **city** des Knotens *shipping_address* im Spreadsheet landen.
		In diesem Fall muss der Parameter **mode** angegeben werden.
		
		Wichtig
		
		Die eckigen Klammern bei *["zip", "city"]* sind wichtig, da dies die Freemarker-Notation für ein Array sind. Bitte nicht vergessen!

	skipEmptyValues *(optional)*	Über *skipEmptyValues (true/false, default: false)* kann man steuern, ob man eine Spalte nur hinzufügen möchte, wenn der Wert der JSON-Node nicht leer ist. Das ist praktisch bei riesigen JSON-Dateien, die haufenweise leere Tags enthalten, die man nicht im Output-Spreadsheet sehen möchte. Damit kann man das resultierende Spreadsheet auf die nötigsten nicht-leeren Spalten beschränken. `${addColumns(row, order['shipping_address'], "shipto_address_", {"columns":["zip","city"], "skipEmptyValues":true})}` Diese Zeile sorgt dafür, dass die Felder *zip* und *city* dem Spreadsheet hinzugefügt werden, allerdings nur wenn das Feld jeweils gefüllt ist. Wenn z.B. die Spalte *zip* immer leer wäre, dann würde im Output-Spreadsheet keine Spalte für *zip* auftauchen.

	delimiter *(optional)* **&** ***textqualifier*** *(optional)*	Beide Parameter gehören zusammen und sind nur anwendbar, wenn als node (siehe weiter der 2. Parameter) ein JSON-Array übergeben wird. **delimiter:** Das Trennzeichen, durch welches die einzelnen Werte der Array-Elemente getrennt werden. **Default**: Komma (,) **textqualifier:** Optionales Zeichen, durch welches die einzelnen Werte der Array-Elemente in der Ausgabe umschlossen werden sollen. Üblicherweise Hochkomma (') oder doppelte Anführungszeichen ("). **Default**: leer **Achtung bei doppelten Hochkomma** Schreiben Sie dazu: **"textqualifier":"\""** Hier muss das Hochkomma durch einen Backslash (\) escaped werden.

	autoExpand
	
	(optional)
		Dieser Parameter steuert, ob und wie Unter-Objekte *automatisch "aufgeklappt"* werden, wenn deren Wert ebenfalls wieder ein JSONObject ist.
		Ohne Angabe von *autoExpand*, werden diese komplexen Felder einfach als JSON-String ausgegeben. Man könnte dieses komplexe Feld durch einen weiteren Aufruf der ${addColumns()} Funktion als Spalten hinzufügen.
		Durch die Option autoExpand, kann man sich diesen weiteren Aufruf sparen, und die Felder werden sofort hinzugefügt. Da dieses Verhalten nicht immer gewünscht ist, ist dies optional. Das spart unter Umständen einiges an Tipparbeit.
		
		• *asColumns* - fügt die Felder des Unterobjekts als neue Spalten hinzu. Dabei werden die Feldnamen des Unterobjekts automatisch mit dem Schlüssel (Key) des Felders ge-prefixt (z.B. billing_address_**street**, billing_address_**city**)
		• *asRows* - Spezial-Fall: Jeder Schlüssel des Unterobjekts erscheint in einer extra Zeile und die Felder des Objekts in extra Spalten.
		
		Beispiel *asColumns*:
		
		JSON Input
		
		`{ "orders": [ { "order_id": "O123", "billing_address": { "city": "Jena", "street": "Meine Straße 1a" } } ] }`
		
		`<#assign row = target.addRow()> <#list json["orders"] as j > <#assign row = target.addRow()> ${addColumns(row, j, "order_", {"autoExpand":"asColumns"})} </#list>`
		
		Ergebnis:
		
		order_order_id | **order_billing_address_city** | **order_billing_address_city**
		O123 | Jena | Meine Straße 1a
		
		Beispiel *asRows:*

JSON Input

```
{
  "Items":[
     {
        "id1":{
           "Filename":"f1.jpg",
           "Link":"http://www.mywebsite.de/f1.jpg"
        }
     },
     {
        "id2":{
           "Filename":"f2.jpg",
           "Link":"http://www.mywebsite.de/f2.jpg"
        }
     }
  ]
}
```

```
<#assign row = target.addRow()>

<#list json["Items"] as j >
    <#assign row = target.addRow()>
    ${addColumns(row, j, "Item", {"autoExpand":"asRows"})}
</#list>
```

Ergebnis

Item	Item_Filename	Item_Link
id1	f1.jpg	`http://www.mywebsite.de/f1.jpg`
id2	f2.jpg	`http://www.mywebsite.de/f2.jpg`

Unterstützung für JSON-Arrays

Es gibt eine Erweiterung für JSON-Arrays, d.h. wenn der *node* Parameter ein Array ist. Diese praktische Erweiterung erlaubt es die Werte der Array-Elemente komma-separiert in eine Spalte zu schreiben.

Beispiel:

Ausgangsdaten: Gegeben sei folgendes JSON-Array als Bestandteil eines JSON-Dokuments:

JSON

```
"tax_lines": [
    {
        "price": "159.65",
        "rate": 0.19,
        "title": "MwSt"
    },
    {
        "price": "123.65",
        "rate": 0.17,
        "title": "Tax2"
    }
]
```

Ziel:

tax_infos_price	tax_infos_rate	tax_infos_title
159.65,123.65	0.19,0.17	MwSt,Tax2

TransformationTemplate:

```
${addColumns(row, json['tax_lines'], "tax_infos_")}
```

Trenner anpassen:

Angenommen, Sie wollen statt Komma einen anderen Trenner nutzen z.B. Pipe, dann können Sie das optional mit angeben:

```
${addColumns(row, json['tax_lines'], "tax_infos", { "delimiter": "|"})}
```

Textqualifier anpassen:

Angenommen, Sie wollen statt zusätzlich jeden Wert noch z.B. durch einfache Hochkomma umschließen:

```
${addColumns(row, json['tax_lines'], "tax_infos", { "delimiter": "|", "textqualifier":"'"})}
```

Mit doppelten Hochkomma umschließen

Für doppelte Hochkomma können Sie schreiben

"textqualifier":"\""

Hier muss das Hochkomma durch einen Backslash (****) escaped werden.

Nur bestimmte Felder hinzufügen:

Angenommen, Sie möchten nur die Spalten *price* und *rate* hinzufügen, aber nicht *title,* dann können Sie dies über die Optionen *columns* und *mode* steuern.

```
${addColumns(row, json['tax_lines'], "tax_infos", { "delimiter": "|", "columns":["price", "rate"]})}
```

Bestimmte Felder ausschließen:

Angenommen Sie wollen die Spalten *price* und *rate* ausschließen, d.h. nur *title* soll erscheinen.

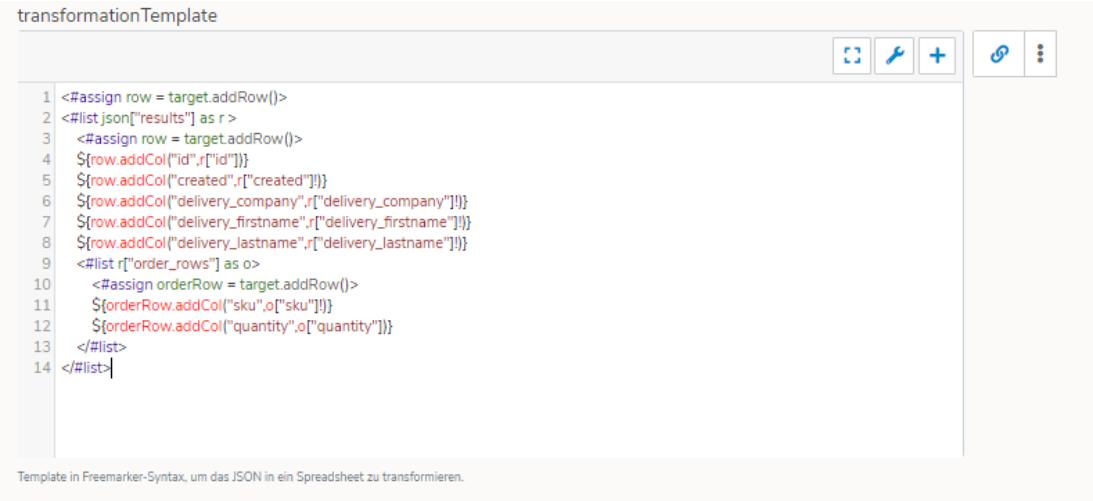
```
${addColumns(row, json['tax_lines'], "tax_infos", { "delimiter": "|", "columns":["price", "rate"], "mode":"
exclude"})}
```

JSON-Einlesen Variante 2

Diese Variate nutzt nicht die addColumns() Funktion, sondern fügt jede Spalte einzeln hinzu. Das ist der manuelle Weg. Dieser Weg bietet volle Flexibilität, da man hier auf jede Spalte Einfluss nehmen kann und auch IF-ELSE Logik einbauen kann. In einigen Fällen kann dies gewünscht sein, wenn addColumns() nicht das gewünschte Ergebnis liefert.

```
<#assign row = target.addRow()>

<#list json["results"] as r >

        <#assign row = target.addRow()>

        ${row.addCol("id",r["id"])}
        ${row.addCol("created",r["created"]!)}
        ${row.addCol("delivery_company",r["delivery_company"]!)}
        ${row.addCol("delivery_firstname",r["delivery_firstname"]!)}
        ${row.addCol("delivery_lastname",r["delivery_lastname"]!)}


        <#list r["order_rows"] as o>

                <#assign orderRow = target.addRow()>

                ${orderRow.addCol("sku",o["sku"]!)}
                 ${orderRow.addCol("quantity",o["quantity"])}

        </#list>

</#list>
```

In der Step-Konfiguration sieht das ganze so aus:



JSON-Einlesen Variante 3

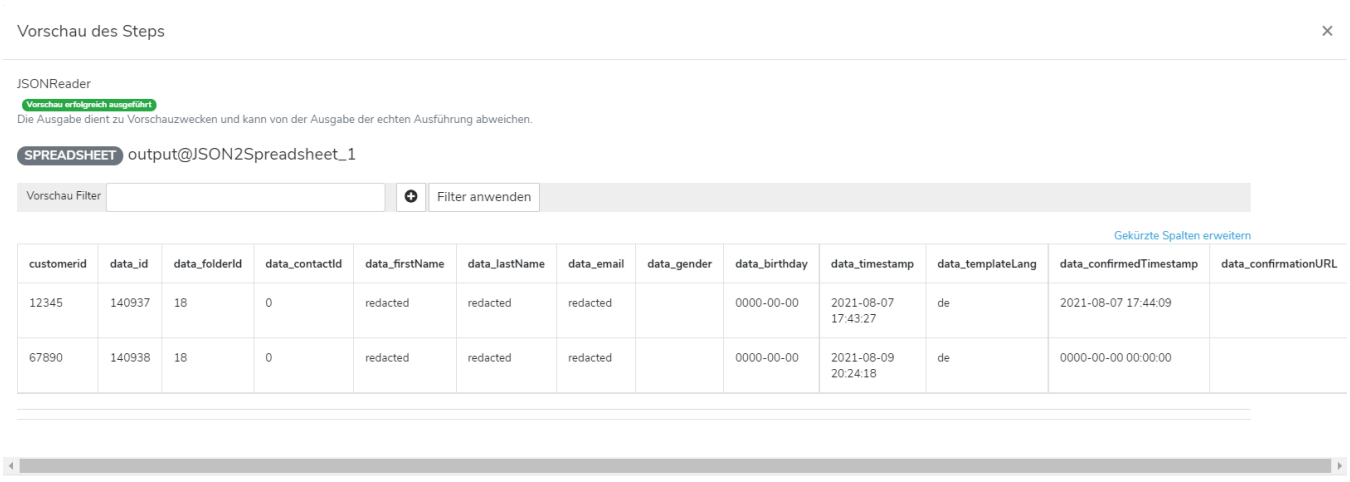
Eine weitere Form von JSON-Struktur ist ein sog. JSON-Object bestehend aus Key-Value-Paaren. D.h. im Gegensatz zum JSON-Array oben, besteht die Struktur nicht aus einer Liste, sondern direkt aus Schlüssel-Wert-Paaren. Folgendes Beispiel zeigt, wie man über diese Paare iteriert und Key und Value ausgibt. Dabei wird Variante 1 (*addColumns()*) und 2 (*addCol()*) kombiniert.

```
{
    "12345": {
        "id": 140937,
        "folderId": "18",
        "contactId": "0",
        "firstName": "redacted",
        "lastName": "redacted",
        "email": "redacted",
        "gender": null,
        "birthday": "0000-00-00",
        "timestamp": "2021-08-07 17:43:27",
        "templateLang": "de",
        "confirmedTimestamp": "2021-08-07 17:44:09",
        "confirmationURL": null
    },
    "67890": {
        "id": 140938,
        "folderId": "18",
        "contactId": "0",
        "firstName": "redacted",
        "lastName": "redacted",
        "email": "redacted",
        "gender": null,
        "birthday": "0000-00-00",
        "timestamp": "2021-08-09 20:24:18",
        "templateLang": "de",
        "confirmedTimestamp": "0000-00-00 00:00:00",
        "confirmationURL": null
    }
}
```


parsingCode für JSON Reader:

```
<#assign row = target.addRow()>

<#list json as key, value>
  <#assign row = target.addRow()>
  ${row.addCol("customerid", key)}
  ${addColumns(row, value, "data_")}
</#list>
```

Ergebnis:



Mit <*#list json as key, value*> kann man über die einzelnen Key-Value-Paare dieser Map iterieren und auf den Key und den Value zugreifen.
Dies ist die Freemarker-Schreibweise für Key-Value-Paare einer Map / Hash.

JSON-Parsing im Detail

Die Logik verhält sich analog zum Einlesen von XML-Dateien, die hier beschrieben ist.

JSON in Spreadsheet-Spalten einlesen

Der JSON2Spreadsheet Step kann ausser FILE und FILELIST Inputs auch mit dem Typ SPREADSHEET umgehen - d.h. ein Spreadsheet, in dem in einer Spalte ein JSON-String steht.

Wann braucht man das?

Meistens wird das gebraucht wenn man vorher mit dem Step SpreadsheetURLDownload arbeitet und z.B. eine REST-API anbindet. Die Ausgabe des SpreadsheetURLDownload ist ein Spreadsheet, in dem in einer Spalte die JSON-Antwort des API-Calls steht (so ist das zumindest bei REST-APIs der Fall).

Anwendung

Sobald Sie dem JSON2Spreadsheet Step das Input-Spreadsheet übergeben, erscheint eine weitere Option ***spreadsheetJSONColumn***, mit der man bestimmt, in welcher Spalte dieses Spreadsheets der JSON-String steht, der geparst werden soll.

> Mit anderen Worten: statt einer Liste von JSON-Dateien (FILELIST) wird eine eine Liste von Spalten verarbeitet, in denen JSON drin steht.

Zugriff auf die Spreadsheet-Spalten

Man kann im *transformationTemplate* auch auf die anderen Spalten des Input-Spreadsheets zuzugreifen. Meistens wird das genutzt in Verbindung mit dem SpreadsheetURLDownload und der dortigen Möglichkeit Spalten über die Option **outputSourceColumns** durchzuschleifen (z.B. Artikelnummer / SKU).

Dafür existiert eine Variable **inputRow**.
Damit haben Sie immer die aktuelle Zeile des Input-Spreadsheets verfügbar und können auf die Spaltenwerte zugreifen.

Beispiel:

```
${inputRow.get("meinSKUSpalte")!}
```

Angenommen im Input-Spreadsheet gibt es eine Spalte *meineSKUSpalte,* die eine Artikelnummer enthält, und die auch wieder in der Ausgabe des JSON2Spreadsheet Steps auftauchen soll.
Sie können im JSON2Spreadsheet dann grob sowas machen, um eine Spalte mit dieser Artikelnummer hinzuzufügen:

```
${row.addCol("SKU", inputRow.get("meinSKUSpalte") )}
```

> Diese Funktion funktioniert analog auch im Step XML2Spreadsheet.

Vorlage mit komplettem Beispiel

Das obige Beispiel steht als komplette Vorlage bereit, die Sie mit einem Klick in ein Projekt installieren können.

- Vorlage jetzt anzeigen

Weitere Beispiele

Weitere auch etwas komplexere Beispiele finden Sie auf der Seite XML und JSON Parsing am Beispiel.