9. 1.3.7 loop mit Funktions-nodes

https://flows.nodered.org/flow/47323c946caa06cf3c7e

Natürlich stehen innerhalb eines function-nodes die in Javascript möglichen Schleifen Konstrukte zur Verfügung. Darum soll es in diesem Kapitel nicht gehen. Hier soll es um die Möglichkeit gehen, wie man Schleifen über einzelne nodes hinweg bilden kann. Dies ist immer dann nötig/ratsam, wenn innerhalb der Schleife die Funktionalität eines anderen nodes benötigt wird.
Derzeit (V0.18.4) wird dies noch nicht direkt unter stützt. Man kann sich einen solchen Flow leicht selbst bauen.
Wie sieht sowas aus?

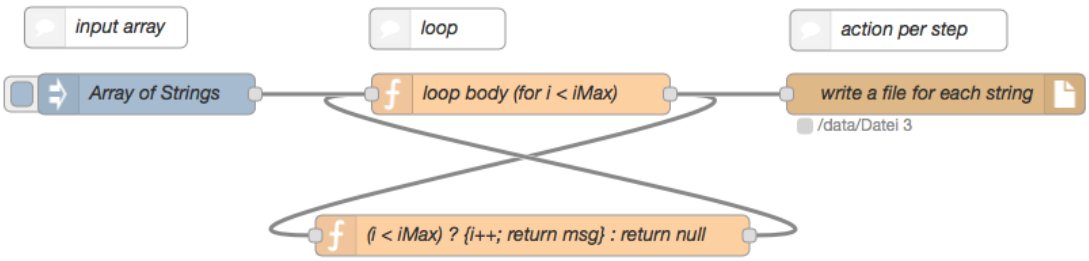

Bild 1: Simple loop

Im Beispiel wird ein Array mit Strings angenommen. Eine Schleife soll nun dafür sorgen, dass für jeden String eine eigne Datei geschrieben wird.
Die Schleife muss nun das input Array durchlaufen und für jeden String eine Output-msg inkl. Filename bilden. Dabei muss Folgendes beachtet werden.

Das Input-Array muss entweder zwischengespeichert werden, um bei jedem loop zur Verfügung zu stehen, oder es muss als Teil der Output – msg durchgereicht werden.
Zur Zwischenspeicherung kann der node context benutzt werden, falls kein weiteres node auf weitere Eigenschaften des Input Array zugreifen muss. Natürlich geht auch der globale context oder der flow context. Den sollte man aber nicht zumüllen.
Beim flow context ist außerdem zu beachten, dass dieser nicht mehr „flow global“ ist, wenn die loop in einem Subflow gepackt wird.
Gute geeignet ist die Variante, das Eingangs Array einfach mit der msg weiterzureichen. Damit stehen dann auch alle Input-Array Eigenschaften den beteiligten Flows zur Verfügung.

Das kann einfach im oberen function-node geschehen:

```
if( msg.items === undefined ) msg.items = msg.payload;
```

Beim ersten Inject gibt es msg.item noch nicht, weshalb es angelegt wird.
Die selbe Frage stellt sich bei der Zählvariable. Auch hier bietet es sich an, diese direkt als msg Property durchzureichen:

```
if( msg.i === undefined ) msg.i = 0;
```

Damit ist die Schleife komplett. Der ganze Schleifenkörper im oberen function-node könnte so aussehen:

```
if( msg.i     === undefined ) msg.i = 0;
if( msg.items === undefined ) msg.items = msg.payload;
let content  = msg.items[ msg.i ];
let filename = "/data/Datei "+(msg.i + 1).toString();
msg.payload = content;
msg.filename = filename;

return msg;
```

mögliche Probleme
Die Konstruktion nach Bild1 könnte problematisch sein, wenn das Aktion-Node, die Nachrichten nicht so schnell verarbeiten kann, wie sie von der loop geliefert werden.
In diesem Fall kann die loop verzögert werden.

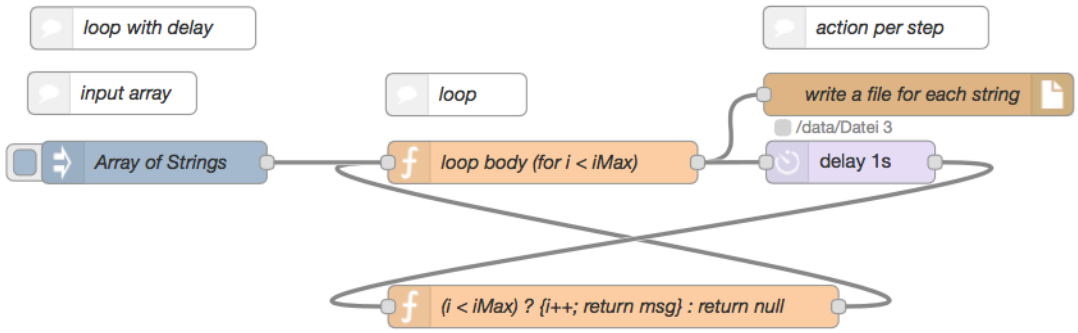

Bild 2: loop mit Verzögerung

Viele nodes geben selbst ein Resultat zurück, auf das reagiert/gewartet werden soll. Datenbank nodes sind typische Beispiele. In diesem Fall kann das node direkt in die loop platziert werden.

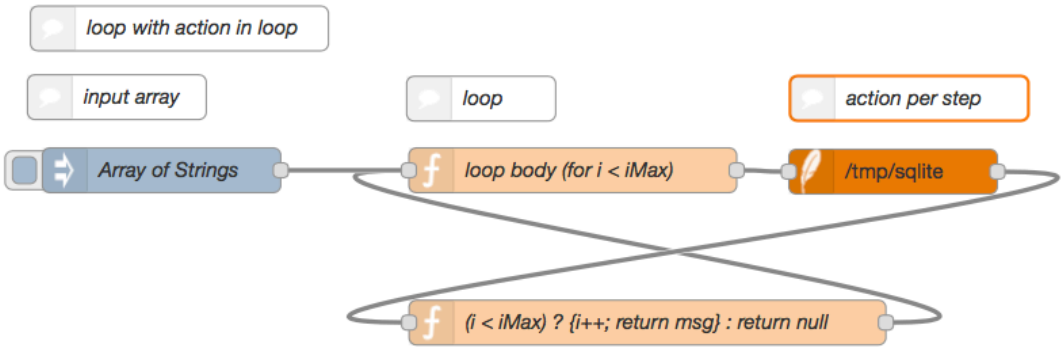

Bild 3: loop mit action-node im loop

Output nach Abschluss der Iterationen

Manchmal dient die loop dazu eine msg aufzubereiten, die danach weiterverarbeitet werden soll. In diesem Fall darf am die msg erst nach Ablauf der loop generiert werden. Im folgenden Beispiel besitzt der loop-body function-node 2 Ausgänge. AM unteren Ausgang werden die Zwischenergebnisse per step bereitgestellt wie in den vorigen Beispielen. Am oberen Ausgang hingegen wird eine Nachricht erst nach Ablauf des loop‘s zur Verfügung gestellt.

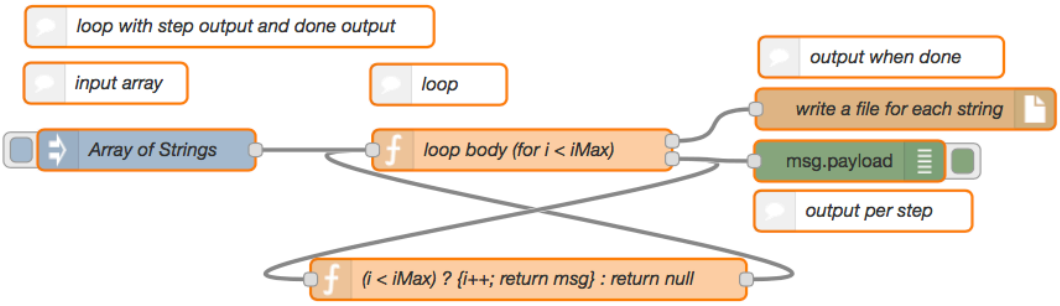

Bild 4: loop mit Ausgang per step und Ausgang 'wenn fertig'

Im Funktionsdialog müssen dazu 2 Outputs eingestellt werden. Die Funktion kann dann z.B. so aussehen:

```
if( msg.i     === undefined ) msg.i = 0;
if( msg.items === undefined ) msg.items = msg.payload;

let content  = msg.items[ msg.i ];
let filename = "/data/kDatei.txt";
msg.filename = filename;

if ((msg.i) < msg.items.length ) { //only actual Array-Item-Content
```

```
                                    //on Output 1
   msg.payload = content;
   return [null, msg];
} else {                            //whole Array as text on Output 2
   msg.payload = msg.items.join("\r\n");
   return [msg, null];
}
```

Hinweis
Die hier gezeigten Beispiele sollen lediglich demonstrieren, wie man Loop's mit function-nodes realisieren kann. Allerdings lassen sich solche Konstrukte meist vermeiden indem man entsprechend spezialisierte nodes verwendet.
Die Aufgabenstellung der Beispiele lassen sich einfach mit einem Splitt-node realisieren .

Werden derartige Loops häufig benötigt, kann man recht einfach ein Subflow dafür erstellen. Siehe z.B.
https://flows.nodered.org/flow/43501a1b424434de0ffb