

<https://nodered.org/docs/user-guide/nodes>

<https://nodered.org/docs/user-guide/nodes#function>



Inject

The Inject node can be used to manual trigger a flow by clicking the node's button within the editor. It can also be used to automatically trigger flows at regular intervals.

The message sent by the Inject node can have its payload and topic properties set.

The payload can be set to a variety of different types:

- a flow or global context property value
- a String, number, boolean, Buffer or Object
- a Timestamp in milliseconds since January 1st, 1970

The interval can be set up to a maximum of 596 hours (approximately 24 days). If you are looking at intervals greater than one day - consider using a scheduler node that can cope with power outages and restarts.

The interval between times and at a specific time options use the standard cron system. This means that 20 minutes will be at the next hour, 20 minutes past and 40 minutes past - not in 20 minutes time. If you want every 20 minutes from now - use the interval option.

Since Node-RED 1.1.0, the Inject node can now set any property on the message.

Injizieren

Der Injektionsknoten kann verwendet werden, um einen Flow manuell auszulösen, indem Sie auf die Schaltfläche des Knotens im Editor klicken. Er kann auch verwendet werden, um Flows automatisch in regelmäßigen Abständen auszulösen.

Die von einem Injektionsknoten gesendete Nachricht kann ihre Nutzlast- und Themen-Eigenschaften haben.

Die Nutzlast kann auf verschiedene Arten eingestellt werden:

- ein Wert einer Fluss- oder globalen Kontexteigenschaft
- ein String, Zahl, boolean, Buffer oder Objekt
- ein Zeitstempel in Millisekunden seit dem 1. Januar 1970

Das Intervall kann auf maximal 596 Stunden (ungefähr 24 Tage) eingestellt werden. Wenn Sie Intervalle größer als einen Tag betrachten, erwägen Sie die Verwendung eines Zeitplanungsknotens, der mit Stromausfällen und Neustarts umgehen kann.

Die Optionen "Intervall zwischen Zeiten" und "zu einer bestimmten Zeit" verwenden das Standard-Cron-System. Dies bedeutet, dass 20 Minuten in der nächsten Stunde, 20 Minuten nach und 40 Minuten nach sind - nicht in 20 Minuten. Wenn Sie alle 20 Minuten von jetzt an möchten, verwenden Sie die Intervall-Option.

Seit Node-RED 1.1.0 kann der Injektionsknoten jede Eigenschaft der Nachricht setzen.



Debug

The Debug node can be used to display messages in the Debug sidebar within the editor.

The sidebar provides a structured view of the messages it is sent, making it easier to explore the message.

Alongside each message, the debug sidebar includes information about the time the message was received and which Debug node sent it. Clicking on the source node id will reveal that node within the workspace.

The button on the node can be used to enable or disable its output. It is recommended to disable or remove any Debug nodes that are not being used.

The node can also be configured to send all messages to the runtime log, or to send short (32 characters) to the status text under the debug node.

The page on [Working with messages](#) gives more information about using the Debug sidebar.

Debuggen

Der Debug-Knoten kann verwendet werden, um Nachrichten in der Debug-Sidebar im Editor anzuzeigen.

Die Sidebar bietet eine strukturierte Ansicht der gesendeten Nachrichten, was die Erkundung der Nachricht erleichtert.

Zusätzlich zu jeder Nachricht enthält die Debug-Sidebar Informationen über die Zeit, zu der die Nachricht empfangen wurde, und welcher Debug-Knoten sie gesendet hat. Durch Klicken auf die Quellknoten-ID wird dieser Knoten im Arbeitsbereich angezeigt.

Die Schaltfläche am Knoten kann verwendet werden, um seine Ausgabe zu aktivieren oder zu deaktivieren. Es wird empfohlen, nicht verwendete Debug-Knoten zu deaktivieren oder zu entfernen.

Der Knoten kann auch so konfiguriert werden, dass alle Nachrichten an das Laufzeitprotokoll gesendet werden oder dass kurze (32 Zeichen) zum Status-Text unter dem Debug-Knoten gesendet werden.

Die Seite zum Arbeiten mit Nachrichten enthält weitere Informationen zur Verwendung der Debug-Sidebar.



Writing Functions

The Function node allows JavaScript code to be run against the messages that are passed through it.

The message is passed in as an object called `msg`. By convention it will have a `msg.payload` property containing the body of the message.

Other nodes may attach their own properties to the message, and they should be described in their documentation.

Writing a Function

The code entered into the Function node represents the *body* of the function. The most simple function simply returns the message exactly as-is:

```
return msg;
```

If the function returns `null`, then no message is passed on and the flow ends.

The function must always return a `msg` object. Returning a number or string will result in an error.

The returned message object does not need to be same object as was passed in; the function can construct a completely new object before returning it.

For example:

```
var newMsg = { payload: msg.payload.length };
return newMsg;
```

Note : constructing a new message object will lose any message properties of the received message. This will break some flows, for example the HTTP In/Response flow requires the `msg.req` and `msg.res` properties to be

Funktionen schreiben

Der Funktionsknoten ermöglicht das Ausführen von JavaScript-Code gegen die durch ihn übergebenen Nachrichten.

Die Nachricht wird als Objekt namens "msg" übergeben. Konventionell hat es eine Eigenschaft "msg.payload", die den Inhalt der Nachricht enthält.

Andere Knoten können ihre eigenen Eigenschaften an die Nachricht anhängen, und sie sollten in ihrer Dokumentation beschrieben werden.

Das Schreiben einer Funktion

Der in den Funktionsknoten eingegebene Code repräsentiert den Körper der Funktion. Die einfachste Funktion gibt die Nachricht einfach so zurück:

```
return msg;
```

Wenn die Funktion `null` zurückgibt, wird keine Nachricht weitergeleitet und der Flow endet.

Die Funktion muss immer ein "msg"-Objekt zurückgeben. Das Zurückgeben einer Zahl oder eines Strings führt zu einem Fehler.

Das zurückgegebene Nachrichtenobjekt muss nicht dasselbe Objekt sein wie das, das übergeben wurde. Die Funktion kann ein völlig neues Objekt erstellen, bevor es zurückgegeben wird.

Zum Beispiel:

```
var newMsg = { payload: msg.payload.length };
return newMsg;
```

Hinweis: Das Konstruieren eines neuen Nachrichtenobjekts führt zum Verlust aller Nachrichteneigenschaften der empfangenen Nachricht. Dies wird einige Flows unterbrechen, z. B. der HTTP In/Response Flow erfordert, dass die Eigenschaften "msg.req" und "msg.res" von

<p>preserved end-to-end. In general, function nodes <i>should</i> return the message object they were passed having made any changes to its properties.</p> <p>Use <code>node.warn()</code> to show warnings in the sidebar to help you debug. For example:</p> <pre>node.warn("my var xyz = " + xyz);</pre> <p>See logging section below for more details.</p>	<p>Anfang bis Ende erhalten bleiben. Im Allgemeinen sollten Funktionsknoten das Nachrichtenobjekt, das sie erhalten haben, zurückgeben, nachdem sie Änderungen an dessen Eigenschaften vorgenommen haben.</p> <p>Verwenden Sie "node.warn()", um Warnungen in der Seitenleiste anzuzeigen und Ihnen bei der Fehlerbehebung zu helfen. Zum Beispiel:</p> <pre>node.warn("my var xyz = " + xyz);</pre> <p>Siehe den Abschnitt "Protokollierung" unten für weitere Details.</p>
<h2>Multiple Outputs</h2> <p>The function edit dialog allows the number of outputs to be changed. If there is more than one output, an array of messages can be returned by the function to send to the outputs.</p> <p>This makes it easy to write a function that sends the message to different outputs depending on some condition. For example, this function would send anything on topic banana to the second output rather than the first:</p> <pre>if (msg.topic === "banana") { return [null, msg]; } else { return [msg, null]; }</pre> <p>The following example passes the original message as-is on the first output and a message containing the payload length is passed to the second output:</p> <pre>var newMsg = { payload: msg.payload.length }; return [msg, newMsg];</pre> <h2>Handling arbitrary number of outputs</h2> <p><i>Since Node-RED 1.3</i></p> <p><code>node.outputCount</code> contains the number of outputs configured for the function node.</p> <p>This makes it possible to write generic functions that can handle variable</p>	<h2>Mehrere Ausgaben</h2> <p>Der Dialog zur Bearbeitung von Funktionen ermöglicht die Änderung der Anzahl der Ausgaben. Wenn es mehr als eine Ausgabe gibt, kann die Funktion ein Array von Nachrichten zurückgeben, das an die Ausgänge gesendet werden soll.</p> <p>Dies erleichtert das Schreiben einer Funktion, die die Nachricht je nach Bedingung an verschiedene Ausgänge sendet. Zum Beispiel würde diese Funktion alles mit dem Thema "banana" an den zweiten Ausgang senden anstatt an den ersten:</p> <pre>if (msg.topic === "banana") { return [null, msg]; } else { return [msg, null]; }</pre> <p>Das folgende Beispiel gibt die Originalnachricht unverändert am ersten Ausgang aus, und eine Nachricht, die die Länge der Nutzlast enthält, wird am zweiten Ausgang übergeben:</p> <pre>var newMsg = { payload: msg.payload.length }; return [msg, newMsg];</pre> <h2>Umgang mit einer beliebigen Anzahl von Ausgaben</h2> <p>Seit Node-RED 1.3</p> <p>"node.outputCount" enthält die Anzahl der für den Funktionsknoten konfigurierten Ausgänge.</p> <p>Dies ermöglicht das Schreiben von generischen Funktionen, die mit der Anzahl der Ausgänge umgehen können, die im Bearbeitungsdialog festgelegt wurden.</p>

<p>number of outputs set from the edit dialog.</p> <p>For example if you wish to spread incoming messages randomly between outputs, you could:</p> <pre>// Create an array same length as there are outputs const messages = new Array(node.outputCount) // Choose random output number to send the message to const chosenOutputIndex = Math.floor(Math.random() * node.outputCount); // Send the message only to chosen output messages[chosenOutputIndex] = msg; // Return the array containing chosen output return messages;</pre> <p>You can now configure number of outputs solely via edit dialog without making changes to the function itself.</p>	<p>Wenn Sie beispielsweise eingehende Nachrichten zufällig zwischen Ausgängen verteilen möchten, könnten Sie folgendes tun:</p> <pre>// Create an array same length as there are outputs const messages = new Array(node.outputCount) // Choose random output number to send the message to const chosenOutputIndex = Math.floor(Math.random() * node.outputCount); // Send the message only to chosen output messages[chosenOutputIndex] = msg; // Return the array containing chosen output return messages;</pre> <p>Sie können nun die Anzahl der Ausgänge ausschließlich über den Bearbeitungsdialog konfigurieren, ohne Änderungen an der Funktion selbst vorzunehmen.</p>
<h2>Multiple Messages</h2> <p>A function can return multiple messages on an output by returning an array of messages within the returned array. When multiple messages are returned for an output, subsequent nodes will receive the messages one at a time in the order they were returned.</p> <p>In the following example, msg1, msg2, msg3 will be sent to the first output. msg4 will be sent to the second output.</p> <pre>var msg1 = { payload:"first out of output 1" }; var msg2 = { payload:"second out of output 1" }; var msg3 = { payload:"third out of output 1" }; var msg4 = { payload:"only message from output 2" }; return [[msg1, msg2, msg3], msg4];</pre> <p>The following example splits the received payload into individual words and returns a message for each of the words.</p> <pre>var outputMsgs = []; var words = msg.payload.split(" "); for (var w in words) { outputMsgs.push({payload:words[w]}); }</pre>	<h2>Mehrere Nachrichten</h2> <p>Eine Funktion kann mehrere Nachrichten für einen Ausgang zurückgeben, indem sie ein Array von Nachrichten innerhalb des zurückgegebenen Arrays zurückgibt. Wenn für einen Ausgang mehrere Nachrichten zurückgegeben werden, erhalten nachfolgende Knoten die Nachrichten einzeln in der Reihenfolge, in der sie zurückgegeben wurden.</p> <p>Im folgenden Beispiel werden "msg1", "msg2", "msg3" an den ersten Ausgang gesendet. "msg4" wird an den zweiten Ausgang gesendet.</p> <pre>var msg1 = { payload:"first out of output 1" }; var msg2 = { payload:"second out of output 1" }; var msg3 = { payload:"third out of output 1" }; var msg4 = { payload:"only message from output 2" }; return [[msg1, msg2, msg3], msg4];</pre> <p>Das folgende Beispiel teilt die empfangene Nutzlast in einzelne Wörter auf und gibt für jedes der Wörter eine Nachricht zurück.</p> <pre>var outputMsgs = []; var words = msg.payload.split(" "); for (var w in words) { outputMsgs.push({payload:words[w]}); }</pre>

<pre>} return [outputMsgs];</pre>	<pre>} return [outputMsgs];</pre>
<p>Sending messages asynchronously</p> <p>If the function needs to perform an asynchronous action before sending a message it cannot return the message at the end of the function.</p> <p>Instead, it must make use of the <code>node . send ()</code> function, passing in the message(s) to be sent. It takes the same arrangement of messages as that can be returned, as described in the previous sections.</p> <p>For example:</p> <pre>doSomeAsyncWork(msg, function(result) { msg.payload = result; node.send(msg); }); return;</pre> <p>The Function node will clone every message object you pass to <code>node . send</code> to ensure there is no unintended modification of message objects that get reused in the function. Before Node-RED 1.0, the Function node would not clone the <i>first</i> message passed to <code>node . send</code>, but would clone the rest.</p> <p>The Function can request the runtime to <i>not clone</i> the first message passed to <code>node . send</code> by passing in <code>false</code> as a second argument to the function. It would do this if the message contains something that is not otherwise cloneable, or for performance reasons to minimise the overhead of sending messages:</p> <pre>node . send(msg, false);</pre>	<p>Nachrichten asynchron senden</p> <p>Wenn die Funktion vor dem Senden einer Nachricht eine asynchrone Aktion durchführen muss, kann sie die Funktion "node.send()" verwenden und die zu sendenden Nachrichten übergeben.</p> <p>Sie nimmt die gleiche Anordnung von Nachrichten wie die, die zurückgegeben werden können, wie in den vorherigen Abschnitten beschrieben.</p> <p>Zum Beispiel:</p> <pre>doSomeAsyncWork(msg, function(result) { msg.payload = result; node.send(msg); }); return;</pre> <p>Der Funktionsknoten kloniert jedes Nachrichtenobjekt, das Sie an "node.send()" übergeben, um sicherzustellen, dass keine unbeabsichtigten Änderungen an Nachrichtenobjekten vorgenommen werden, die in der Funktion wiederverwendet werden. Vor Node-RED 1.0 hat der Funktionsknoten das erste an "node.send()" übergebene Nachrichtenobjekt nicht geklont, sondern die restlichen schon.</p> <p>Der Funktionsknoten kann die Laufzeit auffordern, das erste an "node.send()" übergebene Nachrichtenobjekt nicht zu klonen, indem es "false" als zweites Argument an die Funktion übergibt. Er würde dies tun, wenn die Nachricht etwas enthält, das nicht anderweitig klonbar ist, oder aus Leistungsgründen, um den Overhead beim Senden von Nachrichten zu minimieren:</p> <pre>node.send(msg,false);</pre>
<p>Finishing with a message</p> <p><i>Since Node-RED 1.0</i></p> <p>If a Function node does asynchronous work with a message, the runtime will not automatically know when it has finished handling the message.</p>	<p>Abschließen mit einer Nachricht</p> <p>Seit Node-RED 1.0</p> <p>Wenn ein Funktionsknoten asynchrone Arbeit mit einer Nachricht durchführt, weiß die Laufzeit nicht automatisch, wann er mit der Verarbeitung der Nachricht fertig ist.</p>

<p>To help it do so, the Function node should call <code>node.done()</code> at the appropriate time. This will allow the runtime to properly track messages through the system.</p> <pre>doSomeAsyncWork(msg, function(result) { msg.payload = result; node.send(msg); node.done(); }); return;</pre>	<p>Um dies zu erleichtern, sollte der Funktionsknoten "<code>node.done()</code>" zum geeigneten Zeitpunkt aufrufen. Dies ermöglicht der Laufzeit, Nachrichten im System ordnungsgemäß zu verfolgen.</p> <pre>doSomeAsyncWork(msg, function(result) { msg.payload = result; node.send(msg); node.done(); }); return;</pre>
<p>Running code on start</p> <p><i>Since Node-RED 1.1.0</i></p> <p>With the 1.1.0 release, the Function node provides an On Start tab (labeled Setup before 1.3.0) where you can provide code that will run whenever the node is started. This can be used to setup any state the Function node requires.</p> <p>For example, it can initialise values in local context that the main Function will use:</p> <pre>if (context.get("counter") === undefined) { context.set("counter", 0) }</pre> <p>The On Start function can return a Promise if it needs to complete asynchronous work before the main Function can start processing messages. Any messages that arrive before the On Start function has completed will be queued up, and handled when it is ready.</p>	<p>Code beim Start ausführen</p> <p>Seit Node-RED 1.1.0</p> <p>Mit der Version 1.1.0 bietet der Funktionsknoten einen Tab "On Start" (vor 1.3.0 als "Setup" bezeichnet), auf dem Code bereitgestellt werden kann, der jedes Mal ausgeführt wird, wenn der Knoten gestartet wird. Dies kann verwendet werden, um jeden Zustand vorzubereiten, den der Funktionsknoten benötigt.</p> <p>Beispielsweise kann er Werte im lokalen Kontext initialisieren, die die Hauptfunktion verwenden wird:</p> <pre>if (context.get("counter") === undefined) { context.set("counter", 0) }</pre> <p>Die Funktion "On Start" kann ein Versprechen zurückgeben, wenn sie vor dem Hauptfunktionsstart eine asynchrone Arbeit abschließen muss. Nachrichten, die vor Abschluss der "On Start"-Funktion eintreffen, werden in einer Warteschlange gespeichert und verarbeitet, wenn sie bereit ist.</p>
<p>Tidying up</p> <p>If you do use asynchronous callback code in your functions then you may need to tidy up any outstanding requests, or close any connections, whenever the flow gets re-deployed. You can do this in two different ways.</p> <p>Either by adding a <code>close</code> event handler:</p>	<p>Aufräumen</p> <p>Wenn Sie in Ihren Funktionen asynchronen Callback-Code verwenden, müssen Sie möglicherweise ausstehende Anfragen aufräumen oder Verbindungen schließen, wenn der Flow neu bereitgestellt wird. Dies kann auf zwei verschiedene Arten erfolgen.</p> <p>Entweder durch Hinzufügen eines Schließen-Ereignishandlers:</p>

```
node.on('close', function() {
    // tidy up any async code here - shutdown connections
    and so on.
});
```

Or, *since Node-RED 1.1.0*, you can add code to the On Stop tab (previously labelled Close) in the node's edit dialog.

```
node.on('close', function() {
    // tidy up any async code here - shutdown connections and so on.
});
```

Oder, seit Node-RED 1.1.0, Sie können Code zum Tab "On Stop" hinzufügen (früher als "Close" bezeichnet) im Bearbeitungsdialog des Knotens.

Logging events

If a node needs to log something to the console, it can use one of the follow functions:

```
node.log("Something happened");
node.warn("Something happened you should know about");
node.error("Oh no, something bad happened");
```

Where the console output appears will depend on how your operating system and how you start Node-RED. If you start using a command line - that is the console where logging will appear. If you run as a system service then it may appear in the system log. If you run under an app like PM2 it will have it's own way for showing logs. On a Pi the install script adds a node - red - log command that will display the log.

The warn and error messages also get sent to the debug tab on the right side of the flow editor.

For finer grained logging, `node.trace()` and `node.debug()` are also available. If there is no logger configured to capture those levels, they will not be seen.

Handling errors

If the function encounters an error that should halt the current flow, it should return nothing. To trigger a Catch node on the same tab, the function should call `node.error` with the original message as a second argument:

```
node.error("hit an error", msg);
```

Ereignisse protokollieren

Wenn ein Knoten etwas in die Konsole protokollieren muss, kann er eine der folgenden Funktionen verwenden:

```
node.log("Something happened");
node.warn("Something happened you should know about");
node.error("Oh no, something bad happened");
```

Wo die Konsolenausgabe erscheint, hängt von Ihrem Betriebssystem und davon ab, wie Sie Node-RED starten. Wenn Sie die Befehlszeile verwenden - das ist die Konsole, in der das Protokoll erscheint. Wenn Sie es als Systemdienst ausführen, erscheint es möglicherweise im Systemprotokoll. Wenn Sie unter einer App wie PM2 laufen, hat sie ihre eigene Methode zum Anzeigen von Protokollen. Auf einem Raspberry Pi fügt das Installationsskript einen Befehl "node-red-log" hinzu, der das Protokoll anzeigt.

Die Warn- und Fehlermeldungen werden auch an den Debug-Tab auf der rechten Seite des Floweditors gesendet.

Für feinere Protokollierung sind auch `node.trace()` und `node.debug()` verfügbar. Wenn kein Logger konfiguriert ist, um diese Ebenen zu erfassen, werden sie nicht angezeigt.

Fehler behandeln

Wenn die Funktion auf einen Fehler stößt, der den aktuellen Flow stoppen sollte, sollte sie nichts zurückgeben. Um einen Catch-Knoten auf demselben Tab auszulösen, sollte die Funktion "node.error" mit der Originalnachricht als zweitem Argument aufrufen:

```
node.error("hit an error", msg);
```


Storing data

Aside from the `msg` object, the function can also store data in the context store.

More information about Context within Node-RED is available [here](#).

In the Function node there are three predefined variables that can be used to access context:

- `context` - the node's local context
- `flow` - the flow scope context
- `global` - the global scope context

The following examples use `flow` context, but apply equally well to `context` and `global`.

Note : these predefined variables are a feature of the Function node. If you are creating a custom node, check the [Creating Nodes guide](#) for how to access context.

There are two modes for accessing context; either synchronous or asynchronous. The built-in context stores provide both modes. Some stores may only provide asynchronous access and will throw an error if they are accessed synchronously.

To get a value from context:

```
var myCount = flow.get("count");
```

To set a value:

```
flow.set("count", 123);
```

The following example maintains a count of how many times the function has been run:

```
// initialise the counter to 0 if it doesn't exist already
var count = context.get('count')||0;
count += 1;
```

Speicherung von Daten

Neben dem `msg`-Objekt kann die Funktion auch Daten im Kontextspeicher ablegen.

Weitere Informationen zum Kontext in Node-RED finden Sie hier.

Im Function-Node gibt es drei vordefinierte Variablen, die zum Zugriff auf den Kontext verwendet werden können:

- `context` - the node's local context
- `flow` - the flow scope context
- `global` - the global scope context

Die folgenden Beispiele verwenden den Flow-Kontext, gelten jedoch gleichermaßen für den Kontext und den globalen Kontext.

Hinweis: Diese vordefinierten Variablen sind eine Funktion des Function-Nodes. Wenn Sie einen benutzerdefinierten Node erstellen, überprüfen Sie den Leitfaden zum Erstellen von Nodes, um auf den Kontext zuzugreifen.

Es gibt zwei Modi für den Zugriff auf den Kontext: synchron oder asynchron. Die integrierten Kontextspeicher bieten beide Modi. Einige Speicher bieten möglicherweise nur asynchronen Zugriff und werfen einen Fehler, wenn sie synchron aufgerufen werden.

Um einen Wert aus dem Kontext zu erhalten:

```
var myCount = flow.get("count");
```

Um einen Wert festzulegen:

```
flow.set("count", 123);
```

Das folgende Beispiel behält eine Zählung darüber bei, wie oft die Funktion ausgeführt wurde:

```
// initialise the counter to 0 if it doesn't exist already
var count = context.get('count')||0;
count += 1;
// store the value back
context.set('count',count);
// make it part of the outgoing msg object
```

<pre>// store the value back context.set('count',count); // make it part of the outgoing msg object msg.count = count; return msg;</pre>	<pre>msg.count = count; return msg;</pre>
<p>Get/Set multiple values</p> <p>Since Node-RED 0.19, it is also possible to get or set multiple values in one go:</p> <pre>// Node-RED 0.19 or later var values = flow.get(["count", "colour", "temperature"]); // values[0] is the 'count' value // values[1] is the 'colour' value // values[2] is the 'temperature' value // Node-RED 0.19 or later flow.set(["count", "colour", "temperature"], [123, "red", "12.5"]);</pre> <p>In this case, any missing values are set to null.</p>	<p>Mehrere Werte abrufen/setzen</p> <p>Seit Node-RED 0.19 ist es auch möglich, mehrere Werte auf einmal abzurufen oder festzulegen:</p> <pre>// Node-RED 0.19 or later var values = flow.get(["count", "colour", "temperature"]); // values[0] is the 'count' value // values[1] is the 'colour' value // values[2] is the 'temperature' value // Node-RED 0.19 or later flow.set(["count", "colour", "temperature"], [123, "red", "12.5"]);</pre> <p>In diesem Fall werden fehlende Werte auf null gesetzt.</p>
<p>Asynchronous context access</p> <p>If the context store requires asynchronous access, the get and set functions require an extra callback parameter.</p> <pre>// Get single value flow.get("count", function(err, myCount) { ... }); // Get multiple values flow.get(["count", "colour"], function(err, count, colour) { ... }) // Set single value flow.set("count", 123, function(err) { ... }) // Set multiple values flow.set(["count", "colour"], [123, "red"], function(err)</pre>	<p>Asynchroner Zugriff auf den Kontext</p> <p>Wenn der Kontextspeicher asynchronen Zugriff erfordert, benötigen die Funktionen get und set einen zusätzlichen Rückrufparameter.</p> <pre>// Get single value flow.get("count", function(err, myCount) { ... }); // Get multiple values flow.get(["count", "colour"], function(err, count, colour) { ... }) // Set single value flow.set("count", 123, function(err) { ... }) // Set multiple values flow.set(["count", "colour"], [123, "red"], function(err)</pre>

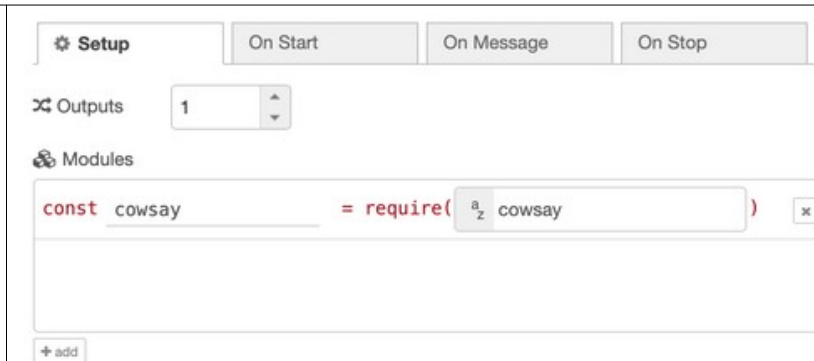
<pre>{ ... })</pre> <p>The first argument passed to the callback, <code>err</code>, is only set if an error occurred when accessing context.</p> <p>The asynchronous version of the count example becomes:</p> <pre>context.get('count', function(err, count) { if (err) { node.error(err, msg); } else { // initialise the counter to 0 if it doesn't exist already count = count 0; count += 1; // store the value back context.set('count',count, function(err) { if (err) { node.error(err, msg); } else { // make it part of the outgoing msg object msg.count = count; // send the message node.send(msg); } }); } });</pre>	<p>Das erste Argument, das dem Rückruf übergeben wird, <code>err</code>, wird nur gesetzt, wenn beim Zugriff auf den Kontext ein Fehler auftritt.</p> <p>Die asynchrone Version des Beispielcodes für die Zählung sieht folgendermaßen aus:</p> <pre>context.get('count', function(err, count) { if (err) { node.error(err, msg); } else { // initialise the counter to 0 if it doesn't exist already count = count 0; count += 1; // store the value back context.set('count',count, function(err) { if (err) { node.error(err, msg); } else { // make it part of the outgoing msg object msg.count = count; // send the message node.send(msg); } }); } });</pre>
<p>Multiple context stores</p> <p>With 0.19 it is possible to configure multiple context stores. For example, both a memory and file based store could be used.</p> <p>The get/set context functions accept an optional parameter to identify the store to use.</p> <pre>// Get value - sync var myCount = flow.get("count", storeName); // Get value - async</pre>	<p>Mehrere Kontextspeicher</p> <p>Ab Version 0.19 ist es möglich, mehrere Kontextspeicher zu konfigurieren. Zum Beispiel könnten sowohl ein speicherbasierter als auch ein dateibasierter Speicher verwendet werden.</p> <p>Die get/set-Kontextfunktionen akzeptieren einen optionalen Parameter zur Identifizierung des zu verwendenden Speichers.</p> <pre>// Get value - sync var myCount = flow.get("count", storeName); // Get value - async</pre>

<pre> flow.get("count", storeName, function(err, myCount) { ... }); // Set value - sync flow.set("count", 123, storeName); // Set value - async flow.set("count", 123, storeName, function(err) { ... }) </pre>	<pre> flow.get("count", storeName, function(err, myCount) { ... }); // Set value - sync flow.set("count", 123, storeName); // Set value - async flow.set("count", 123, storeName, function(err) { ... }) </pre>
<p>Global context</p> <p>The global context can be pre-populated with objects when Node-RED starts. This is defined in the main <i>settings.js</i> file under the <code>functionGlobalContext</code> property.</p> <p>This can be used to load additional modules within the Function node.</p> <p>Adding Status</p> <p>The function node can also provide it's own status decoration in the same way that other nodes can. To set the status, call the <code>node.status</code> function. For example</p> <pre> node.status({fill:"red", shape:"ring", text:"disconnected"}); node.status({fill:"green", shape:"dot", text:"connected"}); node.status({text:"Just text status"}); node.status({}); // to clear the status </pre> <p>For details of the accepted parameters see the Node Status documentation</p> <p>Any status updates can then also be caught by the Status node.</p>	<p>Globaler Kontext</p> <p>Der globale Kontext kann beim Starten von Node-RED mit Objekten vorbelegt werden.</p> <p>Dies wird in der Hauptdatei <code>settings.js</code> unter der Eigenschaft <code>functionGlobalContext</code> definiert.</p> <p>Dies kann verwendet werden, um zusätzliche Module innerhalb des Function-Nodes zu laden.</p> <p>Hinzufügen von Status</p> <p>Der Function-Node kann auch seine eigene Statusdekoration auf die gleiche Weise wie andere Nodes bereitstellen. Um den Status festzulegen, rufen Sie die Funktion <code>node.status</code> auf. Zum Beispiel:</p> <pre> node.status({fill:"red", shape:"ring", text:"disconnected"}); node.status({fill:"green", shape:"dot", text:"connected"}); node.status({text:"Nur Textstatus"}); node.status({}); // um den Status zu löschen </pre> <p>Für Details zu den akzeptierten Parametern siehe die Node-Status-Dokumentation.</p> <p>Alle Statusaktualisierungen können dann auch vom Status-Node erfasst werden.</p>
<p>Loading additional modules</p> <p>Using the <code>functionGlobalContext</code> option</p> <p>Additional node modules cannot be loaded directly within a Function node. They must be loaded in your <i>settings.js</i> file and added to the</p>	<p>Laden zusätzlicher Module</p> <p>Verwendung der Option <code>functionGlobalContext</code></p> <p>Zusätzliche Node-Module können nicht direkt innerhalb eines Function-Nodes geladen werden. Sie müssen in Ihrer <code>settings.js</code>-Datei geladen und der Eigenschaft <code>functionGlobalContext</code> hinzugefügt werden.</p>

<p>functionGlobalContext property.</p> <p>For example, the built-in os module can be made available to all functions by adding the following to your <i>settings.js</i> file.</p> <pre>functionGlobalContext: { osModule:require('os') }</pre> <p>at which point, the module can be referenced within a function as <code>global.get('osModule')</code>.</p> <p>Modules loaded from your settings file must be installed in the same directory as the settings file. For most users that will be the default user directory - <code>~/.node-red</code>:</p> <pre>cd ~/.node-red npm install name_of_3rd_party_module</pre>	<p>Zum Beispiel kann das eingebaute os-Modul für alle Funktionen verfügbar gemacht werden, indem Sie folgendes zu Ihrer settings.js-Datei hinzufügen.</p> <pre>functionGlobalContext: { osModule: require('os') }</pre> <p>An dieser Stelle kann das Modul in einer Funktion als <code>global.get('osModule')</code> referenziert werden.</p> <p>Module, die aus Ihrer Einstellungsdatei geladen werden, müssen im gleichen Verzeichnis wie die Einstellungsdatei installiert sein. Für die meisten Benutzer wird das standardmäßige Benutzerverzeichnis sein - <code>~/.node-red</code>:</p> <pre>cd ~/.node-red npm install name_of_3rd_party_module</pre>
<p>Using the functionExternalModules option</p> <p><i>Since Node-RED 1.3.0</i></p> <p>By setting <code>functionExternalModules</code> to <code>true</code> in your <i>settings.js</i> file, the Function node's edit dialog will provide a list where you can add additional modules that should be available to the node. You also specify the variable that will be used to refer to the module in the node's code.</p>	<p>Verwendung der Option functionExternalModules</p> <p>Seit Node-RED 1.3.0</p> <p>Durch Festlegen von <code>functionExternalModules</code> auf <code>true</code> in Ihrer settings.js-Datei stellt der Bearbeitungsdialog des Function-Nodes eine Liste bereit, in der Sie zusätzliche Module hinzufügen können, die dem Node zur Verfügung stehen sollen. Sie geben auch die Variable an, die im Code des Nodes verwendet wird, um auf das Modul zu verweisen.</p>



The modules are automatically installed under
~/.node-red/externalModules/ when the node is deployed.



Die Module werden automatisch unter ~/.node-red/externalModules/ installiert, wenn der Node bereitgestellt wird.

Handling a Timeout

Since Node-RED 3.1.0

It is possible to set a timeout for the function node on the Setup tab. This value, in seconds, is how long the runtime will allow the Function node to run for before raising an error. If set to 0, the default, no timeout is applied.

Behandlung eines Zeitüberschreitungsfehlers

Seit Node-RED 3.1.0

Es ist möglich, auf dem Setup-Tabulator des Function-Nodes ein Zeitlimit festzulegen. Dieser Wert gibt an, wie lange die Laufzeit dem Function-Node gestatten wird, ausgeführt zu werden, bevor ein Fehler gemeldet wird. Wenn er auf 0, dem Standardwert, gesetzt ist, wird keine Zeitüberschreitung angewendet.

API Reference

The following objects are available within the Function node.

node

- `node.id` : the id of the Function node - *added in 0.19*
- `node.name` : the name of the Function node - *added in 0.19*
- `node.outputCount` : number of outputs set for Function node - *added in 1.3*
- `node.log(..)` : [log a message](#)
- `node.warn(..)` : [log a warning message](#)
- `node.error(..)` : [log an error message](#)

API-Referenz

Die folgenden Objekte stehen innerhalb des Function-Nodes zur Verfügung.

node

- `node.id` : die ID des Function-Nodes
- `node.name` : der Name des Function-Nodes
- `node.outputCount` : Anzahl der für den Function-Node festgelegten Ausgänge
- `node.log(..)` : [log a message](#) eine Nachricht protokollieren
- `node.warn(..)` : [log a warning message](#) eine Warnungsnachricht protokollieren
- `node.error(..)` : [log an error message](#) eine Fehlermeldung protokollieren

- `node.debug(..)` : [log a debug message](#)
- `node.trace(..)` : [log a trace message](#)
- `node.on(..)` : [register an event handler](#)
- `node.status(..)` : [update the node status](#)
- `node.send(..)` : [send a message](#)
- `node.done(..)` : [finish with a message](#)

context

- `context.get(..)` : get a node-scoped context property
- `context.set(..)` : set a node-scoped context property
- `context.keys(..)` : return a list of all node-scoped context property keys
- `context.flow` : same as `flow`
- `context.global` : same as `global`

flow

- `flow.get(..)` : get a flow-scoped context property
- `flow.set(..)` : set a flow-scoped context property
- `flow.keys(..)` : return a list of all flow-scoped context property keys

global

- `global.get(..)` : get a global-scoped context property
- `global.set(..)` : set a global-scoped context property
- `global.keys(..)` : return a list of all global-scoped context property keys

RED

- `RED.util.cloneMessage(..)` : safely clones a message object so it can be reused

- `node.debug(..)` : [log a debug message](#) eine Debug-Nachricht protokollieren
- `node.trace(..)` : [log a trace message](#) eine Trace-Nachricht protokollieren
- `node.on(..)` : [register an event handler](#) einen Ereignishandler registrieren
- `node.status(..)` : [update the node status](#) den Status des Nodes aktualisieren
- `node.send(..)` : [send a message](#) eine Nachricht senden
- `node.done(..)` : [finish with a message](#) mit einer Nachricht abschließen

context

- `context.get(..)` : einen auf den Node bezogenen Kontextwert abrufen
- `context.set(..)` : einen auf den Node bezogenen Kontextwert setzen
- `context.keys(..)` : eine Liste aller auf den Node bezogenen Kontextschlüssel zurückgeben
- `context.flow` : dasselbe wie `flow`
- `context.global` : dasselbe wie `global`

flow


- `flow.get(..)` : einen auf den Flow bezogenen Kontextwert abrufen
- `flow.set(..)` : einen auf den Flow bezogenen Kontextwert setzen
- `flow.keys(..)` : eine Liste aller auf den Flow bezogenen Kontextschlüssel zurückgeben


global

- `global.get(..)` : einen globalen Kontextwert abrufen
- `global.set(..)` : einen globalen Kontextwert setzen
- `global.keys(..)` : eine Liste aller globalen Kontextschlüssel zurückgeben

RED

- `RED.util.cloneMessage(..)` : klonen eines Nachrichtenobjekts auf sichere Weise, um es erneut zu verwenden

<p>env</p> <ul style="list-style-type: none"> • <code>env.get(. .)</code> : get an environment variable <p>Other modules and functions</p> <p>The Function node also makes the following modules and functions available:</p> <ul style="list-style-type: none"> • Buffer - the Node.js Buffer module • console - the Node.js console module (<code>node.log</code> is the preferred method of logging) • util - the Node.js util module • setTimeout/clearTimeout - the javascript timeout functions. • setInterval/clearInterval - the javascript interval functions. <p>Note: the function node automatically clears any outstanding timeouts or interval timers whenever it is stopped or re-deployed.</p>	<p>env</p> <ul style="list-style-type: none"> • <code>env.get(. .)</code> : eine Umgebungsvariable abrufen <p>Andere Module und Funktionen</p> <p>Der Function-Node stellt auch die folgenden Module und Funktionen zur Verfügung:</p> <ul style="list-style-type: none"> • Buffer - das Node.js Buffer-Modul • console - das Node.js console-Modul (<code>node.log</code> ist die bevorzugte Methode zum Protokollieren) • util - das Node.js util-Modul • setTimeout/clearTimeout - die JavaScript Timeout-Funktionen. • setInterval/clearInterval - die JavaScript Interval-Funktionen. <p>Hinweis: Der Function-Node löscht automatisch alle ausstehenden Timeouts oder Intervall-Timer, wenn er gestoppt oder neu bereitgestellt wird.</p>
	
<p>Change</p> <p>The Change node can be used to modify a message's properties and set context properties without having to resort to a Function node.</p> <p>Each node can be configured with multiple operations that are applied in order. The available operations are:</p> <ul style="list-style-type: none"> • Set - set a property. The value can be a variety of different types, or can be taken from an existing message or context property. • Change - search and replace parts of a message property. • Move - move or rename a property. • Delete - delete a property. 	<p>Änderung</p> <p>Der Änderungs-Node kann verwendet werden, um Eigenschaften einer Nachricht zu ändern und Kontexteigenschaften festzulegen, ohne auf einen Function-Node zurückgreifen zu müssen.</p> <p>Jeder Node kann mit mehreren Operationen konfiguriert werden, die in Reihenfolge angewendet werden. Die verfügbaren Operationen sind:</p> <ul style="list-style-type: none"> • Set -eine Eigenschaft festlegen. Der Wert kann verschiedene Typen haben oder aus einer vorhandenen Nachricht oder Kontexteigenschaft stammen. • Change - Teile einer Nachrichteneigenschaft suchen und ersetzen. • Move - eine Eigenschaft verschieben oder umbenennen. • Delete - eine Eigenschaft löschen.

<p>When setting a property, the value can also be the result of a JSONata expression. JSONata is a declarative query and transformation language for JSON data.</p>	<p>Beim Festlegen einer Eigenschaft kann der Wert auch das Ergebnis eines JSONata-Ausdrucks sein. JSONata ist eine deklarative Abfrage- und Transformatiessprache.</p>
	
<p>Switch</p> <p>The Switch node allows messages to be routed to different branches of a flow by evaluating a set of rules against each message.</p> <p>The name "switch" comes from the "switch statement" that is common to many programming languages. It is not a reference to a physical switch</p> <p>The node is configured with the property to test - which can be either a message property or a context property.</p> <p>There are four types of rule:</p> <ul style="list-style-type: none"> • Value rules are evaluated against the configured property • Sequence rules can be used on message sequences, such as those generated by the Split node • A JSONata Expression can be provided that will be evaluated against the whole message and will match if the expression returns a <code>true</code> value. • An Otherwise rule can be used to match if none of the preceding rules have matched. <p>The node will route a message to all outputs corresponding to matching rules. But it can also be configured to stop evaluating rules when it finds one that matches.</p>	<p>Switch</p> <p>Der Switch-Node ermöglicht es, Nachrichten durch Auswerten einer Reihe von Regeln gegen jede Nachricht zu verschiedenen Zweigen eines Flusses zu leiten.</p> <p>Der Name "Switch" stammt vom "Switch-Statement", das in vielen Programmiersprachen üblich ist. Es ist keine Referenz zu einem physischen Schalter.</p> <p>Der Node ist mit der zu testenden Eigenschaft konfiguriert - die entweder eine Nachrichteneigenschaft oder eine Kontexteigenschaft sein kann.</p> <p>Es gibt vier Arten von Regeln:</p> <ul style="list-style-type: none"> • Wertregeln werden gegen die konfigurierte Eigenschaft ausgewertet. • Sequenzregeln können auf Nachrichtensequenzen angewendet werden, wie sie vom Split-Node generiert werden. • Ein JSONata-Ausdruck kann bereitgestellt werden, der gegen die gesamte Nachricht ausgewertet wird und übereinstimmt, wenn der Ausdruck einen Wahrheitswert zurückgibt. • Eine Sonst-Regel kann verwendet werden, um übereinzustimmen, wenn keine der vorherigen Regeln übereinstimmt. <p>Der Node leitet eine Nachricht zu allen Ausgängen, die den übereinstimmenden Regeln entsprechen. Er kann jedoch auch so konfiguriert werden, dass er das Auswerten von Regeln stoppt, wenn er eine findet, die übereinstimmt.</p>



Template

The Template node can be used to generate text using a message's properties to fill out a template.

It uses the [Mustache](#) templating language to generate the result.

For example, a template of:

This is the payload: {{payload}} !

Will replace {{payload}} with the value of the message's payload property.

By default, Mustache will replace certain characters with their HTML escape codes. To stop that happening, you can use triple braces: {{{payload}}}.

Mustache supports simple loops on lists. For example, if msg.payload contains an array of names, such as: ["Nick", "Dave", "Claire"], the following template will create an HTML list of the names:

```
<ul>
{{#payload}}
  <li>{{.}}</li>
{{/payload}}
</ul>
```

```
<ul>
  <li>Nick</li>
  <li>Dave</li>
  <li>Claire</li>
</ul>
```

The node will set the configured message or context property with the result of the template. If the template generates valid JSON or YAML content, it can be

Vorlage

Der Vorlagen-Node kann verwendet werden, um Text zu generieren, wobei die Eigenschaften einer Nachricht verwendet werden, um eine Vorlage auszufüllen.

Er verwendet die Mustache-Vorlagensprache, um das Ergebnis zu generieren.

Beispielsweise wird eine Vorlage wie diese:

This is the payload: {{payload}} !

die Nutzlast: {{{payload}}}! {{{payload}}} durch den Wert der Eigenschaft "payload" der Nachricht ersetzen.

Standardmäßig ersetzt Mustache bestimmte Zeichen durch ihre HTML-Entitäten. Um dies zu verhindern, können dreifache geschweifte Klammern verwendet werden: {{{payload}}}.

Mustache unterstützt einfache Schleifen auf Listen. Wenn beispielsweise msg.payload ein Array von Namen enthält, wie z.B.: ["Nick", "Dave", "Claire"], erstellt die folgende Vorlage eine HTML-Liste der Namen:

```
<ul>
{{#payload}}
  <li>{{.}}</li>
{{/payload}}
</ul>
```

```
<ul>
  <li>Nick</li>
  <li>Dave</li>
  <li>Claire</li>
</ul>
```

Der Node setzt die konfigurierte Nachrichten- oder Kontexteigenschaft mit dem Ergebnis der Vorlage. Wenn die Vorlage gültigen JSON- oder YAML-Inhalt generiert, kann sie so konfiguriert werden, dass das Ergebnis in das entsprechende JavaScript-Objekt geparkt wird.

configured to parse the result to the corresponding JavaScript Object.	