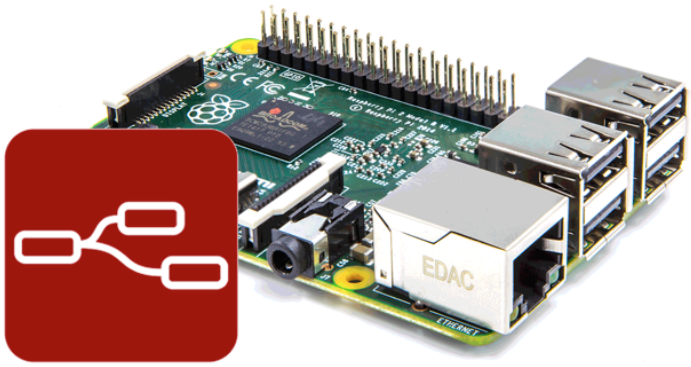
Sommersemester 2016
betreuender Dozent:
Prof. Dr. rer. nat. Thomas Scheffler

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Node-RED und der Raspberry Pi 2
-eine Einführung-



als Projekt zur Vorbereitung der Bachelor-Arbeit

vorgelegt von
Jakob Laue
Matrikelnummer 793883
laue@beuth-hochschule.de
Fachbereich VII – Elektrotechnik
Elektronik und Kommunikationssysteme

Inhaltsverzeichnis

1) Einleitung

Node-RED ist ein Werkzeug für visuelle Programmierung. Es wurde Ende 2013 von Nick O' Leary und Dave Conway-Jones, beide Wissenschaftler der IBM Emerging Technologies Group, entwickelt. Ihr Ziel war es, auf einfachem Wege Hardware mit Webdiensten und Programmierschnittstellen zu verknüpfen. Node-RED hat in kürzester Zeit an Popularität gewonnen und dient heute als beliebtes Tool für das Internet der Dinge (Internet of Things, IoT). Node-RED ist ein Open Source-Projekt und basiert auf Node.js, weshalb sich das Wort 'Node' im Namen wiederfindet. Der Zusatz 'RED' hat keine tiefere Bedeutung.
Dave Conway-Jones schlug den Namen 'Node-RED' vor, da sich dieser wie 'Code Red' anhört.
Mit 'Code Red' bezeichnet man eine Familie von Computerwürmern [1] [2].

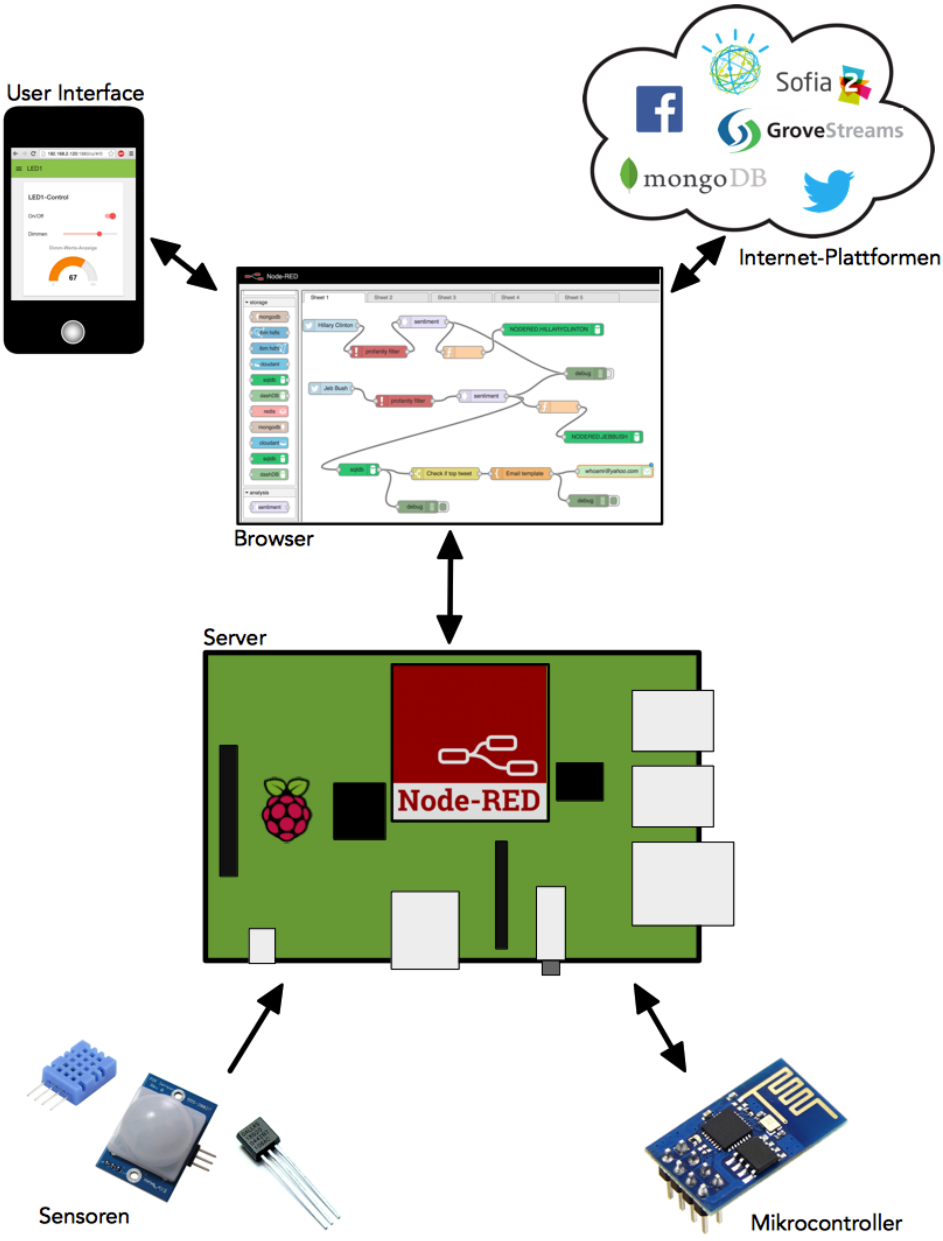


Abbildung 1: Anwendungsschema Node-RED

Abbildung 1 stellt die Anwendung von Node-RED dar. Node-RED läuft auf dem Raspberry Pi, welcher als Webserver für den Browser-basierten Flow-Editor dient. Im Flow-Editor werden dann die eigentlichen Anwendungen, sogenannte Flows, erstellt. Flows können mit vielen Online-Diensten, IoT-Plattformen und Web-Apps kommunizieren. Einige Beispiele hierfür sind IBM Watson, MongoDB, Emoncms, Aquila, Owntracks, Twilio und Twitter. Mit dem User Interface integriert man Smartphones und Tablets in den Flow. Auch Sensordaten können in Node-RED eingelesen und verarbeitet werden. Ebenso ist die Integration von Mikrocontrollern (zum Beispiel Arduino oder ESP8266) möglich. Node-RED kann man auf jedem Windows-, Mac OS- oder Linux-System installieren. Da der IoT-Gedanke immer mehr im Vordergrund steht, wird diesbezüglich gern auf Einplatinencomputer zurückgegriffen, die immer eingeschaltet bleiben.
Als Hardwareplattform wird neben dem Raspberry Pi auch der BeagleBone Black unterstützt [3] [4].

2) Flows, Nodes und Nachrichten

Flows werden im Browser erstellt und anschließend auf dem Raspberry Pi ausgeführt und gespeichert. Flows bestehen aus mehreren vorgefertigten Code-Blöcken, sogenannten Nodes, die man über ihre Ein- und Ausgänge miteinander verbindet. Ein Node besteht einerseits aus JavaScript-Code und andererseits aus HTML-Code. Der HTML-Code bestimmt, wie der Node im Flow-Editor aussieht, wie er dort konfiguriert werden kann und welcher Info-Text dem Node beigefügt wird [5].

Entlang der Flows fließen Nachrichten. Nachrichten in Node-RED sind JavaScript-Objekte. Ein JavaScript-Objekt kann mehrere Eigenschaften aufweisen:

```
var x = {vorname: "Karl", nachname: "Petersen"};
```

In diesem Beispiel hat das Objekt 'x' die Eigenschaften 'vorname' und 'nachname'. Der Zugriff auf die Eigenschaften geschieht mit:

```
var y = x.vorname;
var z = x.nachname;
```

In Node-RED heißt ein Nachrichten-Objekt meist 'msg' und es weist standardmäßig drei Eigenschaften auf:

```
var msg = {payload: "Nutzdaten",
           topic: "Raspberry",
           msgid: "9392aeb8.6c6d5"};
```

msg.payload: Enthält die Nutzdaten einer Nachricht. Das können zum Beispiel der Text eines Tweets, der Inhalt einer Webseite oder der Messwert eines Sensors sein.

msg.topic: Ein benutzerdefinierter String, der es ermöglicht, die Nachricht zu kategorisieren.

msg.msgid: Eine Kennung der Nachricht.

Nodes können Nachrichten weitere Eigenschaften anfügen.

Das Menü oben rechts im Flow-Editor bietet die Möglichkeit, Flows zu importieren oder zu exportieren. Zum Importieren 'import → Clipboard' wählen. Es erscheint ein Dialogfenster. In dieses fügt man einen Flow im JavaScript Object Notation-Format (JSON) aus der Zwischenablage ein. Der gewünschte Flow erscheint und kann per Mausklick in der Arbeitsfläche abgelegt werden. Zum Exportieren zunächst alle Nodes des Flows markieren und anschließend im Menü 'export → Clipboard' wählen. Es erscheint ein Dialogfenster, welches den markierten Flow im JSON-Format bereitstellt [6].

2.1) Input, Output und Processing

Per Default sind bereits viele Nodes vorinstalliert. Das sind vorgefertigte Code-Blöcke, die es dem Anwender erlauben, ohne oder mit nur wenig selbst geschriebenem Code komplexe Flows zu konstruieren. Nodes arbeiten mit einer Eingangsnachricht und generieren eine oder mehrere Ausgangsnachricht(en).

Es gibt Input Nodes, Output Nodes und Processing Nodes. Es ist möglich, Nodes mit der gewohnten Tastenkombination 'cmd → c' und 'cmd → v' (Mac) beziehungsweise mit 'strg → c' und 'strg → v' (Windows) zu kopieren und einzufügen.

Input Nodes: Sie speisen den Flow mit Daten. Das können beispielsweise Daten von einem GPIO Pin des Raspberry Pis, einer WebSocket-Verbindung oder einem Twitter-Account sein. Input Nodes besitzen mindestens einen Ausgang, dargestellt durch ein kleines graues Rechteck.

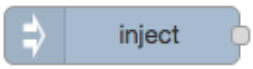


Abbildung 2: Inject Node als Beispiel für ein Input Node

Output Nodes: Sie senden Daten aus dem Flow heraus an andere Dienste. Das können zum Beispiel Tweets sein, aber auch Daten für eine serielle Schnittstelle oder für TCP-Verbindungen. Output Nodes besitzen mindestens einen Eingang.

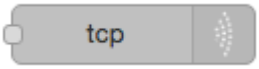


Abbildung 3: Output Node für eine TCP-Verbindung

Processing/Function Nodes: Sie erlauben es, eingehende Daten zu verarbeiten. Processing Nodes besitzen einen Eingang und mindestens einen Ausgang. Unter anderem können eigene Codes geschrieben, Zahlenbereiche verändert und Datentypen umgewandelt werden.

Abbildung 4: Processing Node für eigenen Code

2.2) Beschreibung der wichtigsten Nodes

Nodes sind in Kategorien eingeteilt. Es sind acht Kategorien vorinstalliert: Input, Output, Function, Social, Storage, Analysis, Advanced, Raspberry Pi

Man kann seine eigenen Nodes programmieren. Somit gehen aus der Node-RED-Community dauerhaft neue Nodes hervor, die mit bestimmter Hardware oder bestimmten Programmierschnittstellen (zum Beispiel von Fitness- oder Wetter-Apps) kommunizieren können. Neue Nodes sind einfach über die Kommandozeile zu installieren (siehe Abschnitt 4.4).

Werfen wir nun einen Blick auf die wichtigsten Nodes und ihre Funktion.

Input Nodes

Inject In: Sendet als 'msg.payload' beispielsweise einen Zeitstempel oder einen benutzerdefinierten Text. Die Nachricht kann auf Mausklick, in bestimmten Intervallen oder zu definierten Zeiten in den Flow gespeist werden. Inject Nodes dienen somit als Trigger für den Flow.

Catch In: Fängt Fehlermeldungen ab und generiert eine Nachricht mit Fehler-Eigenschaft. Die Fehler-Eigenschaft beschreibt detailliert den Fehler und von welchem Node dieser stammt.

Mqtt In: Ist ein MQTT-Subscriber eines definierten MQTT-Brokers und lauscht auf einem definierten Topic. Gibt alle Daten, die auf jenem Topic veröffentlicht werden, in einer neuen Nachricht zurück.

Http In: Empfängt HTTP-Anfragen und gibt den HTTP-Body am Ausgang als Nachricht zurück.

Websocket In: Dient als Endpunkt für Browser bei der Duplex-Kommunikation mit dem Websocket-Protokoll.

Tcp In: Nimmt eingehende TCP-Anfragen auf einem definierten Port an und stellt die empfangenen Daten am Ausgang als Nachricht zur Verfügung.

Udp In: Empfängt UDP-Pakete auf einem definierten Port und stellt die empfangenen Daten am Ausgang als Nachricht zur Verfügung.

Serial In: Liest Daten aus der lokalen seriellen Schnittstelle.

Output Nodes

Debug: Zeigt Nachrichten, Fehler und Warnungen rechts im Debug-Bereich an. Ein Mausklick auf das grüne Rechteck rechts am Debug Node schaltet ihn ein oder aus. Dies ist nützlich, wenn man in mehreren Flows mehrere Debug Nodes verwendet und die Zahl an Debug-Meldungen gemindert werden soll. Wenn man die Maus über eine Debug-Meldung hält, dann wird der Debug Node, von dem diese Meldung stammt, rot umrandet. Um Eigenschaften und Inhalte von Nachrichten vollständig anzuzeigen, muss der Ausgang des Debug Nodes als 'complete msg object' konfiguriert werden:

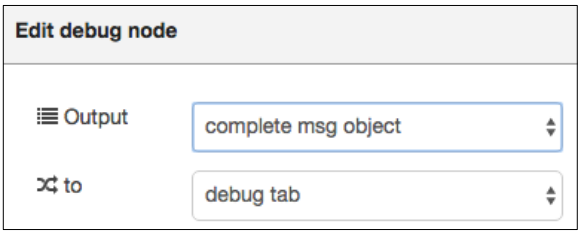


Abbildung 5: Konfiguration eines Debug Nodes

Mqtt Out: Ist ein MQTT-Subscriber eines definierten MQTT-Brokers. Veröffentlicht 'msg.payload' der eingehenden Nachrichten auf einem definierten Topic.

Http Out: Sendet HTTP-Antworten auf HTTP-Anfragen, die in einem Http In Node eingegangen sind.

Websocket Out: Sendet auf dem konfigurierten Websocket 'msg.payload' aus. Wenn msg._session definiert ist, wird nur zum ursprünglichen Client gesendet, andernfalls an alle verbundenen Clients.

Tcp Out: Sendet beziehungsweise antwortet auf einem definierten TCP-Port.

Udp Out: Sendet ein UDP-Paket auf einem definierten Port zu einer definierten IP-Adresse.

Serial Out: Sendet Daten an die definierte serielle Schnittstelle.

Processing Nodes

Function: Hier kann eigener JavaScript-Code stehen. Um den Flow fortzusetzen, sendet man mit 'return' eine oder mehrere Nachrichten an den Ausgang. Um den Nachrichtenfluss zu unterbrechen, verwendet man 'return null'. Nodes ignorieren null-Nachrichten.

Delay: Verzögert Nachrichten um eine definierte oder zufällige Zeit. Kann auch benutzt werden, um ausschließlich eine definierte Anzahl an Nachrichten pro Zeiteinheit passieren zu lassen.

Trigger: Erzeugt zwei Nachrichten am Ausgang, wenn eine Nachricht am Eingang anliegt. Die beiden erzeugten Nachrichten sind um ein definiertes Zeitintervall voneinander getrennt.

Comment: Erzeugt einen Textkommentar, um komplexe Flows übersichtlicher zu machen.

Switch: Einfaches Treffen von Entscheidungen. Routing von eingehenden Nachrichten beziehungsweise deren Eigenschaften an unterschiedliche Ausgänge.

Change: Verändern, Hinzufügen und Löschen von Nachricht-Eigenschaften.

Rbe (report by exception): Lässt Nachrichten nur passieren, wenn sich eine eingehende Nachricht-Eigenschaft verändert hat (rbe-Modus) oder wenn sich eine Eigenschaft um einen definierten Wert ändert (deadband-Modus). Unterstützt nur numerische Werte.

Range: Lineares Skalieren von Zahlenbereichen (zum Beispiel Eingangswerte '0-10' zu Ausgangswerten '0-255). Unterstützt nur numerische Werte.

Http Request: Konstruieren von HTTP-Anfragen, um URLs abzurufen. Die Request-Methode (PUT, GET,..) wird im Konfigurationsdialog festgelegt oder in einem Function Node selbst programmiert.

Csv: Versucht 'msg.payload' in das CSV-Format oder aus dem CSV-Format zu wandeln. Ein empfangener CSV-String wird in ein JavaScript-Objekt gewandelt, ein empfangenes JavaScript-Objekt wird in einen CSV-String gewandelt.

Json: Wandelt ein empfangenes JavaScript-Objekt ins JSON-Format oder einen empfangenen JSON-String in ein JavaScript-Objekt.

Html: Extrahiert Elemente aus einem HTML-Dokument und stellt sie in 'msg.payload' zur Verfügung. Die Auswahl der HTML-Elemente geschieht über einstellbare CSS-Selektoren.

Xml: Ein empfangenes JavaScript-Objekt wird in einen XML-String gewandelt, ein empfangener XML-String wird in ein JavaScript-Objekt gewandelt.

Social Nodes

Email In: Empfängt Emails von einem IMAP-Server. Der Text der Email wird in 'msg.payload' gespeichert, der Betreff in 'msg.topic'.

Email Out: Sendet 'msg.payload' als Email über einen SMTP-Server an einen definierten Adressaten. Wenn 'msg.payload' aus Binärdaten besteht, wird es in einen Anhang gewandelt. Als Betreff dient 'msg.topic'.

Twitter In: Hört auf Tweets zu vorgegebenen Schlüsselwörtern und gibt sie als Nachricht zurück.

Twitter Out: Erstellt einen Tweet aus 'msg.payload' auf einem definierten Account.

Raspberry Pi Nodes

Gpio In: Generiert 'msg.payload' mit '1' oder '0', abhängig vom Status eines Pins und setzt 'msg.topic' auf 'pi/<Nummer des Pins>'. Es können Pullup- und Pulldown-Widerstände geschaltet werden.

Gpio Out: Erwartet im Digital-Modus ein 'msg.payload' mit '1' oder '0' beziehungsweise 'true' oder 'false' und setzt den ausgewählten Pin entsprechend high oder low. Im PWM-Modus muss der empfangene Wert zwischen '0' und '100' liegen.

Hinweis: Die Raspberry Pi Nodes funktionieren nur mit RPI.GPIO-python-library-Versionen ab 0.5.8.
Die Pin-Nummern in Node-RED entsprechen den physikalischen Pin-Nummern auf dem Raspberry Pi (siehe Abschnitt 4.2, Abbildung 17) [7] [8] [9].

2.3) Der Function Node

Dieser Abschnitt stellt einige Beispiele vor, die beim Schreiben seines eigenen JavaScript-Codes im Function Node hilfreich sein können. Das Buch-Symbol oben rechts im Kofigurationsdialog erlaubt es, geschriebenen Code abzuspeichern und abzurufen.

Beispiel 1: Eingehende Nachrichten verändern

```
msg.payload += "Weitere Nutzdaten";
return msg;
```

Beispiel 2: Eine neue Nachricht erzeugen

```
var newMsg = { payload: "Neue Nutzdaten" };
return newMsg;
```

Beispiel 3: Eine Nachricht an mehrere Ausgänge senden

Im Konfigurationsdialog kann man den Function Node mit einer gewünschten Anzahl von Ausgängen versehen:

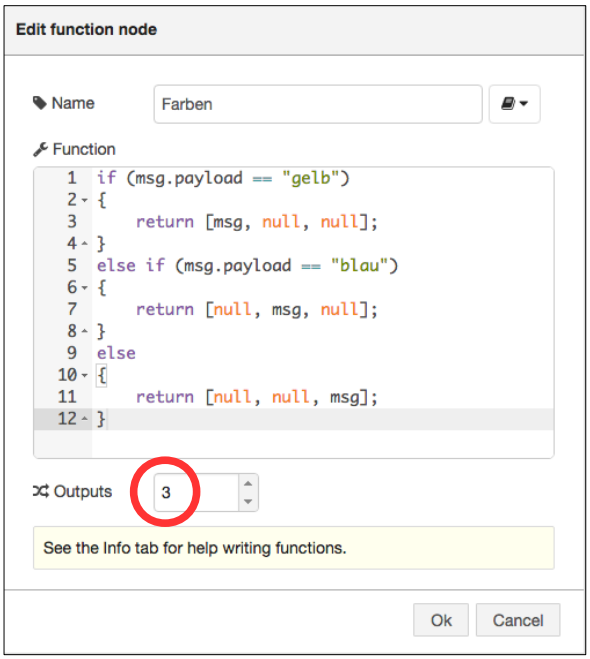


Abbildung 6: Function Node mit mehreren Ausgängen konfigurieren

Der Code in Abbildung 6 zeigt, wie eine Nachricht mithilfe eines zurückgegebenen Arrays an verschiedene Ausgänge gelangt. Zeile 3 sendet die Nachricht an den ersten Ausgang, Zeile 7 sendet an den zweiten Ausgang und Zeile 11 sendet an den dritten Ausgang.

Beispiel 4: Eine Sequenz von Nachrichten an einen Ausgang senden

Es ist möglich, Nachrichten aufeinanderfolgend an einen Ausgang zu senden. Dazu wird die gewünschte Sequenz als ein Array von Nachrichten angegeben. Dieses Array muss wiederum innerhalb des zurückgegebenen Arrays stehen, wenn an mehrere Ausgänge gesendet werden soll.

```
var msg1 = { payload: "An" };
var msg2 = { payload: "Ausgang" };
var msg3 = { payload: "Eins" };
var msg4 = { payload: "Nur an Ausgang Zwei" };
return [ [ msg1, msg2, msg3 ], msg4 ];
```

Dieser Code sendet 'msg1', 'msg2' und 'msg3' (in dieser Reihenfolge) an den ersten Ausgang und 'msg4' an den zweiten Ausgang.

Beispiel 5: Das Context-Objekt

Das Context-Objekt ist nützlich, wenn man zwischen Aufrufen des Function Nodes Informationen speichern möchte. Der folgende Code benutzt das Context-Objekt, um die Anzahl der verarbeiteten Nachrichten zu erfassen:

```
var x = context.get("zaehle") || 0; //initialisiere
x += 1;
context.set("zaehle",x); //speichere
msg.count = x;
return msg;
```

Falls die Context-Variable 'zaehle' exisitiert, wird ihr Wert in 'x' gespeichert, andernfalls wird 'x' zu '0' initialisiert. Danach wird 'x' inkrementiert und mit der Methode 'set()' in der Context-Variable 'zaehle' gespeichert. Jede passierende Nachricht erhält dann den aktuellen Zählwert als neue Eigenschaft 'count'. Context-Variablen sind nur innerhalb eines Function Nodes gültig.

Beispiel 6: Das Flow-Context-Objekt

Mit dem Flow-Objekt ist es möglich, globale Variablen zu erstellen, die für alle Function Nodes innerhalb eines Flows zugänglich sind. In einem ersten Function Node könnte folgender Code stehen:

```
var glob = "gelb";
flow.set("farbe",glob);
return msg;
```

Hier wird im ersten Schritt die Variable 'glob' deklariert und ihr der String 'gelb' zugewiesen.
Die Methode 'set()' speichert 'glob' in der Flow-Variable 'farbe' und macht sie innerhalb des Flows verfügbar.

In einem zweiten Function Node erfolgt der Zugriff auf die Flow-Variable mit der Methode 'get()':

```
var meineFarbe = flow.get("farbe"); //'meineFarbe' ist jetzt 'gelb'
msg.payload = meineFarbe;
return msg;
```

Auch Switch Nodes und Change Nodes können auf das Flow-Objekt zugreifen:

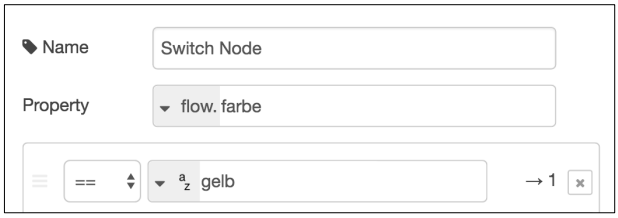


Abbildung 7: Zugriff auf das Flow-Objekt in einem Switch Node

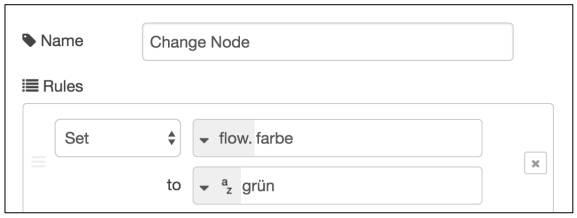


Abbildung 8: Zugriff auf das Flow-Objekt in einem Change Node

Beispiel 7: Das globale Context-Objekt

Es gibt auch ein globales Context-Objekt, welches in allen Flows verfügbar ist. Somit kann man globale Variablen erstellen, die auch außerhalb eines Flows erreichbar sind:

```
var glob2 = global.get("zahl")||0; //initialisiere
glob2 = 8;
global.set("zahl",glob2);
return msg;
```

Falls die globale Variable 'zahl' bereits existiert, wird ihr Inhalt in 'glob2' gespeichert. Andernfalls wird 'glob2' zu '0' initialisiert. Danach wird 'glob2' der Wert '8' zugewiesen. Die Methode 'set()' schreibt diesen Wert aus 'glob2' in die globale Context-Variable 'zahl' und stellt diese flow-übergreifend zur Verfügung.

In einem anderen Flow kann die globale Context-Variable 'zahl' wieder mit der Methode 'get()' gelesen werden:

```
var meineZahl = global.get("zahl"); //'meineZahl' ist jetzt 8
msg.payload = meineZahl;
return msg;
```

Auch Switch Nodes und Change Nodes können auf das globale Context-Objekt zugreifen:

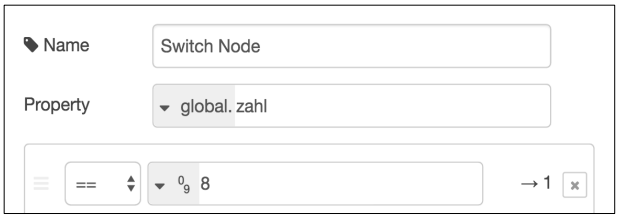


Abbildung 9: Zugriff auf das globale Context-Objekt in einem Switch Node

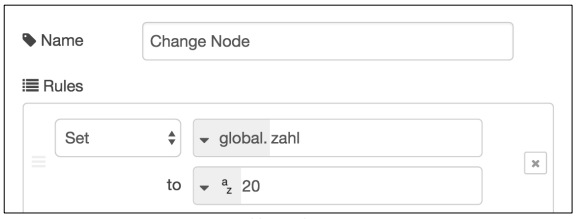


Abbildung 10: Zugriff auf das globale Context-Objekt in einem Change Node

Die Context-Objekte werden zurückgesetzt, sobald man einen Flow neu ausführt oder Node-RED neu startet.

Beispiel 8: Die send()-Methode

Das node-Modul beinhaltet die Methode 'send()', mit der Nachrichten an den Ausgang des Function Nodes gelangen, unabhängig von der 'return'-Angabe [5] [8]:

```
var newMsg = {payload: "Test"}; //neue Nachricht
node.send(newMsg); //sende neue Nachricht
return msg; //lasse eingehende Nachricht passieren
```

3) Node-RED installieren und starten

Alle in diesem Dokument angegebenen Terminal-Befehle gelten für den Raspberry Pi 2 Model B und wurden getestet auf den Betriebssystemen Raspbian Jessie (von Mai 2016) und Raspbian Wheezy (von Februar 2015). Es wird vorausgesetzt, dass der Raspberry Pi über eine Internetverbindung verfügt und dass man sich per ssh auf dem Raspberry Pi einloggen kann:

```
ssh pi@[IP-Adresse-des-Pis]
```

3.1) Raspbian Jessie

Seit November 2015 ist Node-RED auf Raspbian Jessie bereits vorinstalliert. Die folgenden Terminal-Befehle führen ein Update von Node-RED durch:

```
sudo apt-get update
sudo apt-get install nodered
```

Node-RED starten:

```
node-red-start
```

Nachdem man mit 'ctrl → c' zurück zur Konsole gewechselt hat, kann man Node-RED mit dem Befehl

```
node-red-stop
```

beenden [10].

3.2) Raspbian Wheezy

In Raspbian Wheezy erfolgt die Installation mithilfe des Node Package Managers (npm):

Packet Cache und Source List aktualisieren:

```
curl -sL https://deb.nodesource.com/setup | sudo bash -
```

Node.js und npm installieren:

```
sudo apt-get install nodejs
```

Vergewissern, dass Node.js und npm installiert wurden:

```
node -v
npm -v
```

npm updaten:

```
sudo npm -g install npm@2.1.1
npm -v
```

Node-RED installieren:

```
sudo npm install -g --unsafe-perm node-red
```

Node-RED starten:

```
node-red-pi --max-old-space-size=128
```

Die Tastenkombination 'ctrl → c' beendet Node-RED [11].

3.3) Öffnen des Flow-Editors

Nachdem Node-RED auf dem Raspberry Pi gestartet wurde, ist der Flow-Editor erreichbar. Dazu im Browser eines anderen Computers eingeben:

```
[Ip-Adresse-des-Pis]:1880
```

Oder, bei Verwendung des Browsers auf dem Raspberry Pi selbst:

```
localhost:1880
```

Die IP-Adresse des Raspberry Pis findet man heraus mit:

```
hostname -I
```

Im Flow-Editor sind links die Nodes zu sehen. In der Mitte befindet sich die Arbeitsfläche, auf die man per Drag and Drop Nodes platziert, diese anschließend konfiguriert und dann zu einem Flow verbindet. Der Ausgabebereich rechts besteht aus einem Info-Tab und einem Debug-Tab. Der Info-Tab zeigt Hilfsinformationen zu dem gerade selektierten Node an. Der Debug-Tab zeigt die Ausgabe von Debug Nodes an sowie Fehler und Warnungen. Darüber befindet sich der Deploy-Knopf. Ein Klick hierauf stellt den Flow auf dem Server bereit [6].

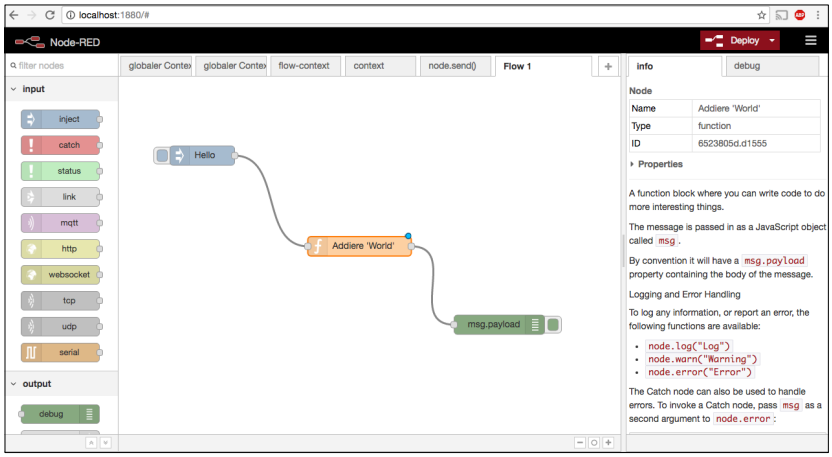


Abbildung 11: Der Flow-Editor

4) Beispiel-Flows

Nach dem Start von Node-RED auf dem Raspberry Pi und dem Öffnen des Flow-Editors im Browser können wir nun einige Beispiel-Flows erstellen.

4.1) Hello World

Dieser Hello World-Flow verdeutlicht den Fluss von Nachrichten in Node-RED. Zuerst ziehen wir einen Inject Node und einen Debug Node auf die Arbeitsfläche.

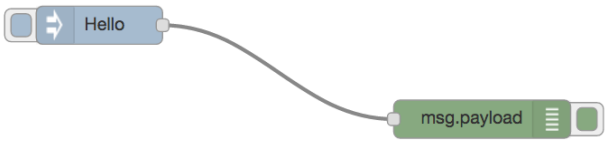


Abbildung 12: Der erste Flow

Ein Doppelklick auf den Inject Node öffnet den Konfigurationsdialog. Wir wählen als Payload einen String aus und geben ihn als 'Hello' an:

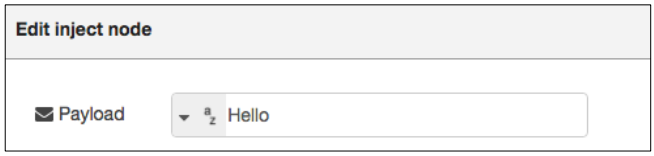


Abbildung 13: Konfiguration des Inject Nodes

Wir vervollständigen den Flow mit einem Debug Node und verbinden beide Nodes. Ein Klick auf 'Deploy' führt den Flow aus. Wenn wir nun links am Inject Node auf das blaue Rechteck klicken, senden wir unsere Nachricht und im Debug-Bereich wird diese angezeigt.

Im nächsten Schritt erweitern wir den Flow um einen Function Node. In diesem verändern wir 'msg.payload' der eingehenden Nachricht:

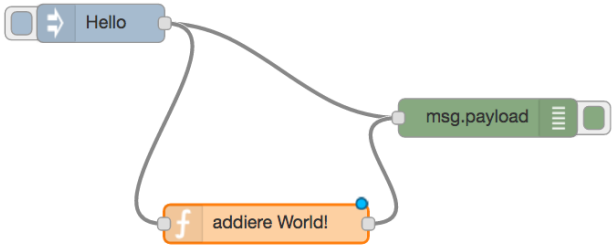


Abbildung 14: Hello World-Flow

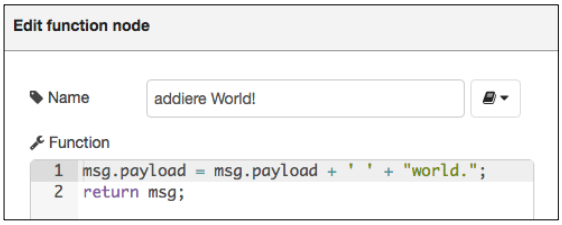


Abbildung 15: Code für den Function Node

Durch die Änderung ist ein erneuter Start des Flows nötig. Wenn wir anschließend mit dem Klick auf den Inject Node die Nachricht nochmals generieren, dann wandert diese einmal direkt in den Debug Node und einmal in den Function Node. Dort wird sie verarbeitet und dann an den Debug Node gegeben. Rechts im Debug Bereich sind beide Nachrichten zu sehen:

Abbildung 16: Debug-Anzeige

4.2) Togglen einer LED

Zunächst eine LED wie gezeigt an Pin 7 des Raspberry Pis anschließen. Danach einen Inject Node, einen Function Node sowie einen Gpio Out Node aus der Raspberry Pi-Kategorie zu folgendem Flow verbinden:

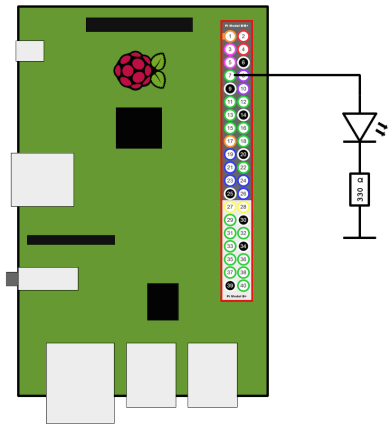


Abbildung 17: LED an Pin 7

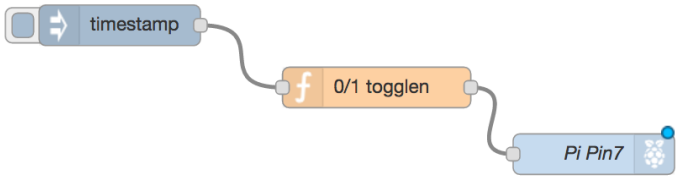


Abbildung 18: LED-Flow

Der Inject Node soll jede Sekunde einen Zeitstempel senden. In dem Function Node wird dieser mit '1' oder '0' überschrieben. Mithilfe des Context-Objektes merken wir uns den momentanen Zustand:

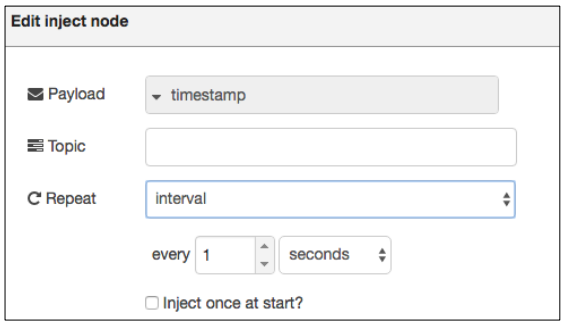


Abbildung 19: Konfiguration des Inject Nodes

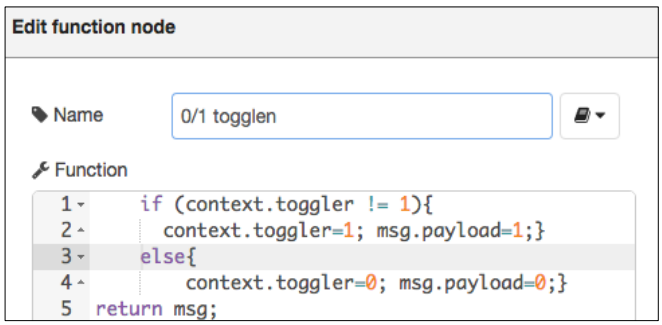


Abbildung 20: Code für den Function Node

Hinweis: Der in Abbildung 20 verwendete Zugriff auf das Context-Objekt wird in neueren Versionen von Node-RED noch unterstützt, soll aber zugunsten der in Abschnitt 2.3 vorgestellten Methoden 'set()' und 'get()' abgelöst werden [8].

In dem Gpio Out Node konfigurieren wir Pin 7 und 'Digital output':

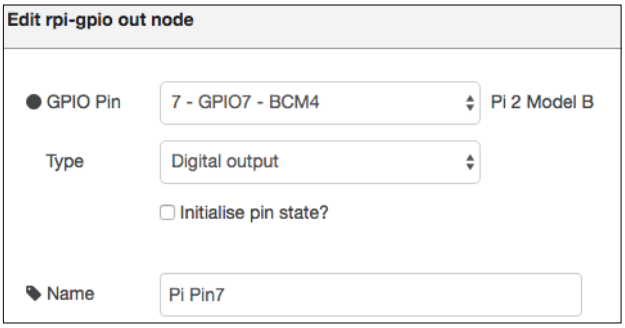


Abbildung 21: Konfiguration des Gpio Out Nodes

Nach einem Klick auf 'Deploy' blinkt nun die LED.

4.3) LED steuern per Twitter

Dieses Beispiel zeigt wie man per Twitter eine LED steuern kann.

Hinweis: Wenn ein Flow mit der Twitter-API kommunizieren soll und innerhalb von 15 Minuten mehrfach neu ausgeführt wird, so werfen die Twitter Nodes eine Fehlermeldung 'tweet rate limit hit'. Es ist dann nicht mehr möglich, weitere Tweets zu empfangen oder zu senden. Debug Nodes helfen, solche Fehlermeldungen zu erfassen. Je öfter der Flow neu ausgeführt wird, desto länger muss man warten, ehe der Flow wieder funktioniert. Es wird daher empfohlen, den gesamten Flow zu konfigurieren und erst dann auszuführen.

Der Flow beinhaltet zwei Twitter Input Nodes, zwei Function Nodes, drei Debug Nodes, zwei Execute Nodes und einen Raspberry Pi-Gpio Out Node:

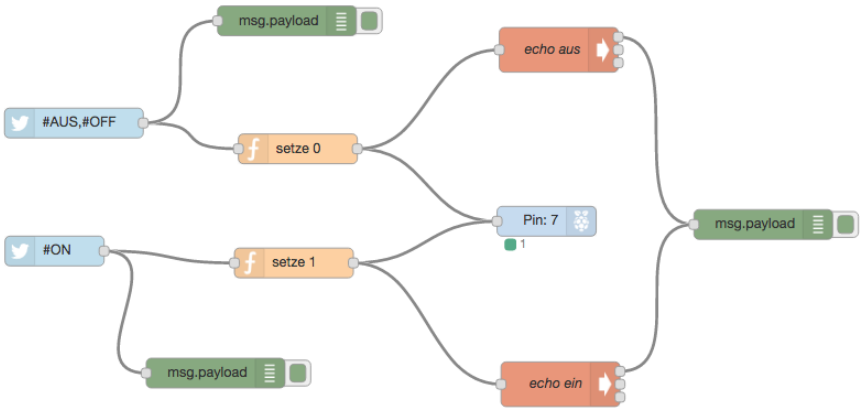


Abbildung 22: Twitter-LED-Flow

Bei dem erstmaligen Konfigurieren der Twitter Nodes muss der gewünschte Twitter-Account autorisiert werden:

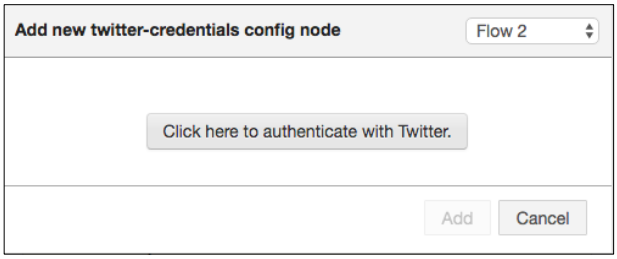


Abbildung 23: Autorisieren eines Twitter-Accounts

Der erste Twitter Node sammelt alle öffentlichen Tweets mit den Hashtags #AUS und #OFF. Der zweite Twitter Node sammelt alle öffentlichen Tweets mit dem Hashtag #AN. Die Ausgänge der Twitter Nodes verbinden wir jeweils mit einem Debug Node. So sind die gesammelten Tweets im Debug-Bereich sichtbar.

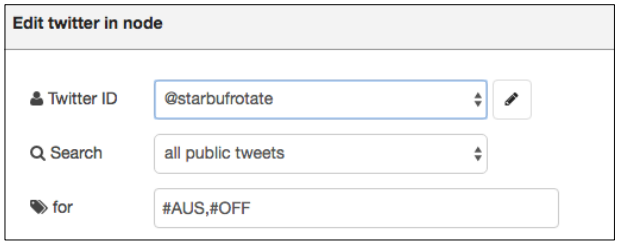


Abbildung 24: Konfiguration eines Twitter Nodes

Treffen passende Tweets ein, so generieren die Twitter Nodes eine Nachricht 'msg'. Die Function Nodes empfangen diese, setzen 'msg.payload' auf '0' beziehungsweise '1' und geben die Nachricht weiter an ihren Ausgang:

Abbildung 25: Code für den oberen Function Node

Abbildung 26: Code für den unteren Function Node

Der Gpio Out Node erwartet 'msg.payload=1' oder 'msg.payload=0'. Folglich schaltet er den angegebenen Pin 7 high, sobald er eine Nachricht des unteren Function Nodes empfängt und low, sobald er eine Nachricht des oberen Function Nodes empfängt.

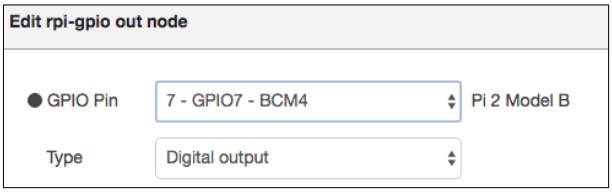


Abbildung 27: Konfiguration des Gpio Out Nodes

Mit den Exec Nodes führen wir zwei einfache Shell-Befehle (echo) aus, sodass die Debug Nodes das Ein- und Ausschalten der LED dokumentieren:

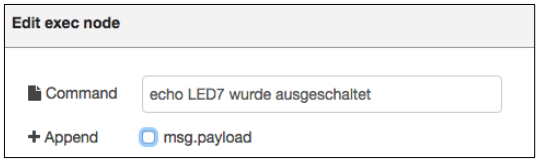


Abbildung 28: Konfiguration des oberen Exec Nodes

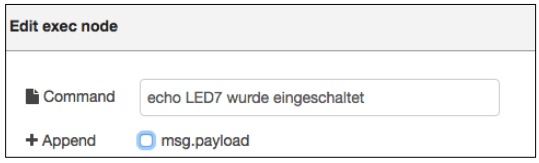


Abbildung 29: Konfiguration des unteren Exec Nodes

Der Flow ist somit vollständig und kann ausgeführt werden. Die LED wird nun aufleuchten, sobald jemand mit dem Hashtag #ON twittert und sie wird ausgehen, sobald jemand mit dem Hashtag #OFF oder #AUS twittert. Den Exec Node haben wir hier nur für einen echo-Befehl genutzt, doch hier könnten auch Shell-Skripte ausgeführt werden.

4.4) Das User Interface

Das User Interface (UI) ist eine konfigurierbare Benutzeroberfläche, von der aus man mit einem Flow interagieren kann. Sie wird im Flow selbst mithilfe von UI-Nodes erstellt und ist dann unter

```
[IP-des-Pis]:1880/ui
```

erreichbar. Unter anderem stehen Taster, Schalter und Schieberegler zur Verfügung, ebenso können Graphen erstellt werden. Das User Interface eignet sich sehr gut als App für Smartphones und Tablets. Zuerst installieren wir die UI-Nodes und wechseln dafür in das Node-RED-Verzeichnis.

Aus dem home-Verzeichnis:

```
cd \.node-red
```

Aus einem höheren Verzeichnis:

```
cd home/pi/.node-red
```

UI-Nodes installieren:

```
npm install node-red-contrib-ui
```

Danach Node-RED neu starten und den Flow-Editor im Browser neu laden [12] [13]. In der Node-Palette sollten nun die UI-Nodes sichtbar sein. In diesem Beispiel wollen wir zwei LEDs schalten und dimmen. Zunächst verbinden wir die LEDs wie gezeigt mit Pin 7 und Pin 10. Danach verbinden wir zwei Switch Nodes aus der UI-Kategorie und zwei Gpio Out Nodes zu folgendem Flow:

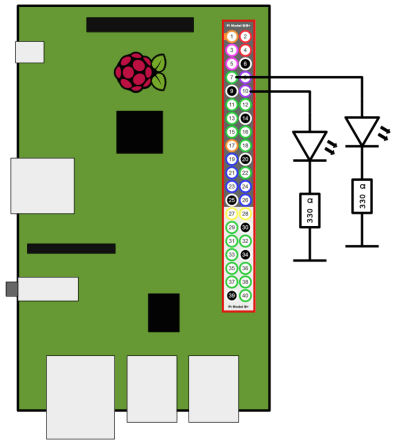


Abbildung 30: LEDs an Pin 7 und Pin 10

Abbildung 31: Erster UI-Flow

Wir konfigurieren die Switch Nodes wie folgt:

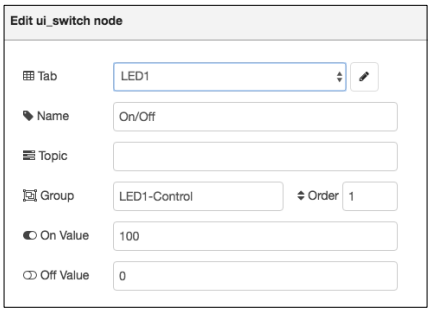


Abbildung 32: Konfiguration des ersten Switch Nodes

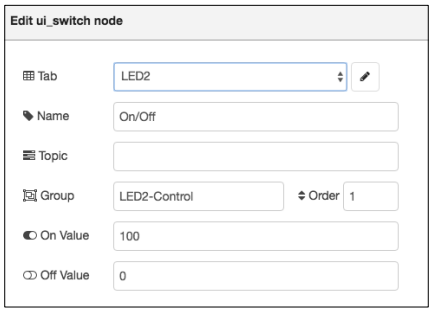


Abbildung 33: Konfiguration des zweiten Switch Nodes

Verschiedene Einträge im Feld 'Tab' erzeugen im User Interface unterschiedliche Tabs, zwischen denen man wechseln kann. Wir erzeugen jeweils einen für die erste und die zweite LED. Gehören UI-Elemente zur gleichen Gruppe, so werden sie im User Interface in einem abgeschlossenen Bereich zusammengefasst. Die Gpio Out Nodes konfigurieren wir für Pin 7 und Pin 10, ebenso wählen wir 'PWM output'.

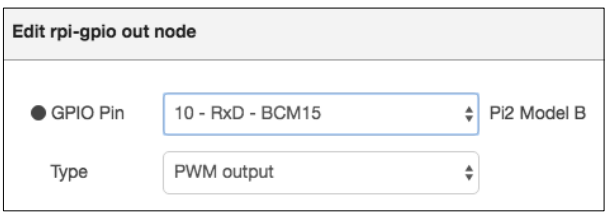


Abbildung 34: Konfiguration der Gpio Out Nodes

Nun den Flow ausführen und das User Interface aufrufen. Es sollte folgendermaßen aussehen und die LEDs steuerbar sein. Der Balken oben links erlaubt das Wechseln zwischen den beiden Tabs 'LED1' und 'LED2':

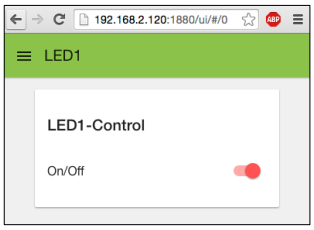


Abbildung 35: Tab 1 im ersten User Interface

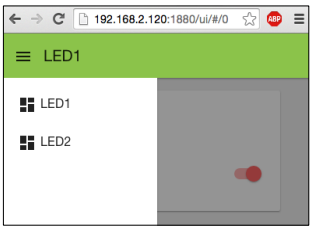


Abbildung 36: Wechseln zwischen den Tabs

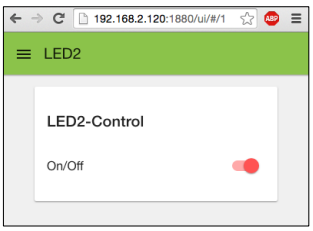


Abbildung 37: Tab 2 im ersten User Interface

Im nächsten Schritt wollen wir die LEDs dimmen. Dazu erweitern wir den Flow um zwei Slider Nodes, die wir ebenfalls den Tabs 'LED1' und 'LED2' sowie den Gruppen 'LED1-Control' und 'LED2-Control' zuordnen. Der Wertebereich reicht wie zuvor von '0' bis '100'.

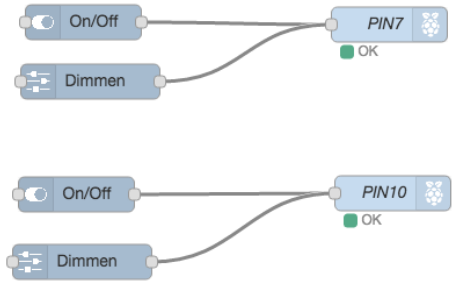


Abbildung 38: Erweiterter UI-Flow

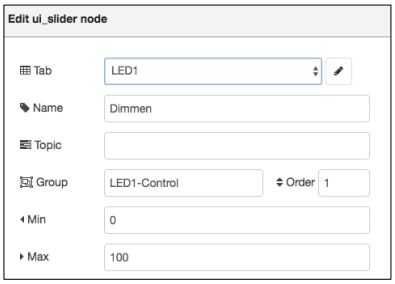


Abbildung 39: Konfiguration des ersten Slider Nodes

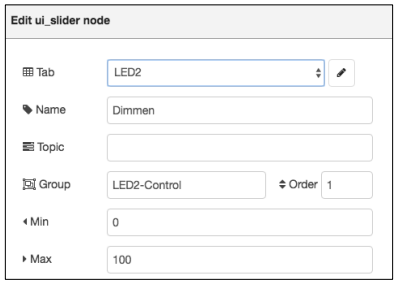


Abbildung 40: Konfiguration des zweiten Slider Nodes

Jetzt den Flow nochmals ausführen. Im User Interface sind unter beiden Tabs nun Schieberegler sichtbar und wir können die LEDs dimmen:

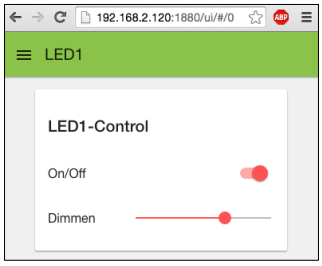


Abbildung 41: Erweitertes User Interface

Zuletzt erweitern wir den Flow um zwei Gauge Nodes, um im User Interface eine Werteanzeige zu erhalten. Auch diese Nodes ordnen wir wie vorher den gleichen Tabs und Gruppen zu:

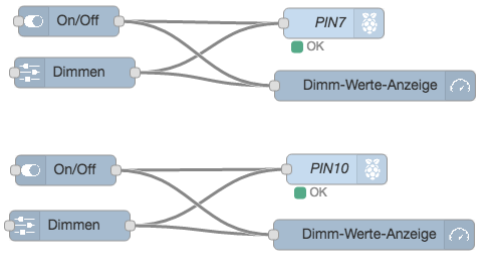


Abbildung 42: Vollständiger UI-Flow

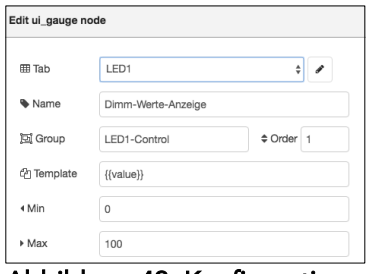


Abbildung 43: Konfiguration des ersten Gauge Nodes

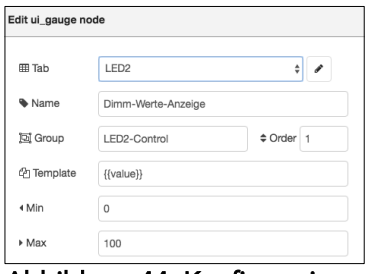


Abbildung 44: Konfiguration des zweiten Gauge Nodes

Nach dem erneuten Ausführen des Flows ist das User Interface vollständig. Die Werteanzeige stellt nun Änderungen an dem Schieberegler und dem Off/On-Schalter dar. Ist das User Interface auf mehreren Geräten geöffnet, so werden alle Aktivitäten synchronisiert.

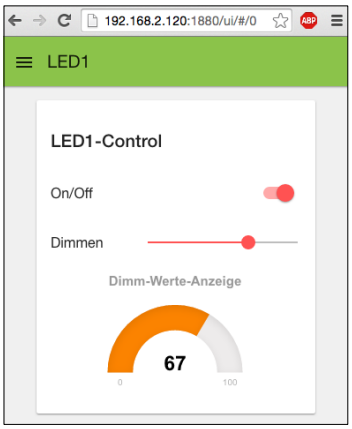


Abbildung 45: Vollständiges User Interface

4.5) Wettervorhersage mit Taster

Dieser Flow erstellt auf Knopfdruck eine Wettervorhersage für Berlin und veröffentlicht sie auf Twitter. Zu Beginn verbinden wir Pin 12 mit einem Taster:

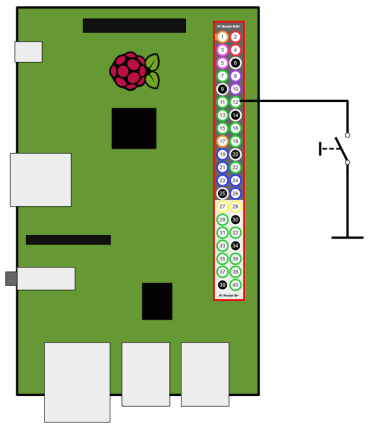


Abbildung 46: Taster an Pin 12

Der Flow ist folgendermaßen aufgebaut:

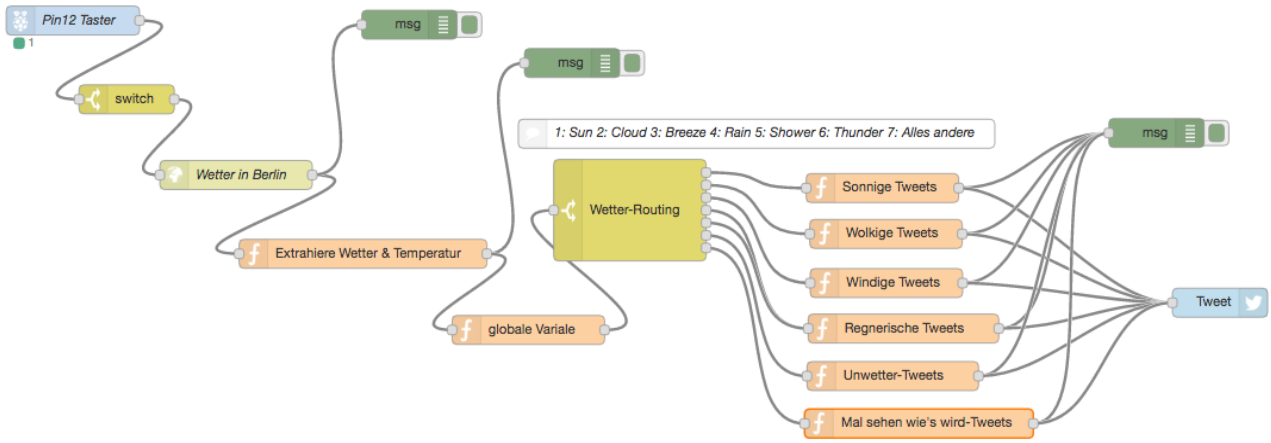


Abbildung 47: Vollständiger Wetter-Flow

Zuerst benötigen wir einen Gpio In Node für den Raspberry Pi. Wir weisen ihm Pin 12 zu und konfigurieren einen Pullup-Widerstand. Ein Druck auf den Taster soll den Flow triggern. Zum Testen kann man für diesen Zweck auch immer einen Inject Node verwenden:

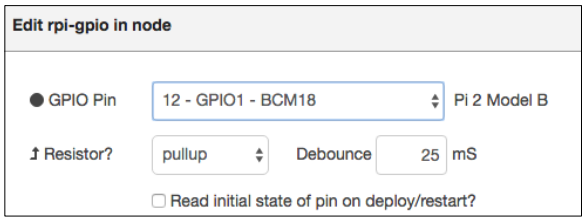


Abbildung 48: Konfiguration des Gpio In Nodes

Der Gpio In Node sendet 'msg.payload=0', wenn der Taster gedrückt wird und sofort danach 'msg.payload=1', wenn man den Taster loslässt. Da der Flow aber nur einmal getriggert werden soll, verwenden wir einen Switch Node, der die empfangene Nachricht nur weiterleitet, wenn sie 'msg.payload=0' enthält:

Abbildung 49: Konfiguration des Switch Nodes

Im nächsten Schritt fragen wir die Yahoo Weather API nach Wetterdaten ab. Die folgende URL liefert ein JSON-Objekt mit aktuellen Wetterdaten für Berlin. Man kann zum Testen einfach die URL in einem Browser eingeben, um das JSON-Objekt zu sehen:

https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D%22berlin%2C%20ak%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys

Um Wetterdaten für eine andere Stadt zu erhalten, kann 'berlin' in der URL gegen einen anderen Städtenamen ersetzt werden. In Node-RED geschieht die Abfrage der Wetterdaten mittels eines HTTP Request Nodes. Wir konfigurieren als Methode 'GET', geben die obige URL und ein JSON-Objekt als gewünschtes Rückgabeformat an:

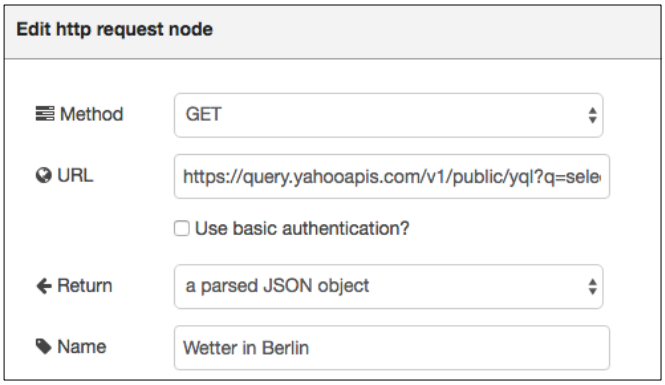


Abbildung 50: Konfiguration des HTTP Request Nodes

Wir verbinden den Ausgang des HTTP Request Nodes mit einem Debug Node, um die Abfrage der Wetterdaten zu überprüfen. Den Debug Node konfigurieren wir, wie auch alle weiteren Debug Nodes, wie in Abbildung 5, für die Anzeige des gesamten empfangenen Nachrichten-Objektes. Bei einem Druck auf den Taster generiert der HTTP Request Node nun ein Nachrichten-Objekt 'msg' und hängt diesem das JSON-Objekt als 'msg.payload' an. Um im Debug-Bereich das gesamte JSON-Objekt lesen zu können, muss man in der Datei 'settings.js' die Anzahl angezeigter Zeichen erhöhen (siehe Abschnitt 5.1).

Im nächsten Schritt wollen wir in einem Function Node für uns relevante Informationen aus dem JSON-Objekt extrahieren. Jedoch ist das JSON-Objekt noch sehr unübersichtlich. Für einen besseren Überblick kann man einen Online-JSON-Viewer verwenden: http://jsonviewer.stack.hu/ . Man fügt das JSON-Objekt ein und bekommt es in einer übersichtlichen Struktur zurück.

In dem Code greifen wir zuerst auf die Beschreibung des heutigen Wetters zu und speichern sie in der Variable 'wetter_heute'. Außerdem speichern wir die heutige Höchsttemperatur in der Variable 'temp_fahrenheit' (Zeile eins bis drei). Anschließend rechnen wir die Höchsttemperatur von Grad Fahrenheit in Grad Celsius um, runden das Ergebnis auf und speichern es ab (Zeile vier). Danach generieren wir ein neues Nachrichten-Objekt 'msg1'. Dieses erhält als 'payload'-Eigenschaft die Beschreibung des heutigen Wetters sowie als 'temp'-Eigenschaft die heutige Höchsttemperatur in Grad Celsius (Zeile sechs). Das neue Nachrichten-Objekt 'msg1' stellen wir an dem Ausgang des Function Nodes zur Verfügung (Zeile sieben):

```
//Code für den Function Node 'Extrahiere Wetter & Temperatur'
1 var vorhersage = msg.payload.query.results.channel.item.forecast;
2 var wetter_heute = vorhersage[0].text;
3 var temp_fahrenheit = msg.payload.query.results.channel.item.condition.temp;
4 var temp_celsius = Math.round((temp_fahrenheit - 32)* (5/9));
5
6 msg1 = {payload: wetter_heute, temp: temp_celsius};
7 return msg1;
```

Verbinden wir nun den Ausgang des Function Nodes mit einem weiteren Debug Node und starten den Flow. Bei einem Druck auf den Taster erscheint im Debug-Bereich das neue Objekt. Wie gewünscht enthält es als 'msg.payload' die Beschreibung des heutigen Wetters auf Englisch und als 'msg.temp' die Höchsttemperatur in Grad Celsius.

Als nächstes folgt ein weiterer Function Node. In diesem erzeugen wir mithilfe des Flow-Context-Objektes eine globale Variable 'save'. Sie wird mit jedem Druck auf den Taster inkrementiert und auf '0' zurückgesetzt, wenn sie den Wert '4' angenommen hat. Den aktuellen Wert von 'save' fügen wir der passierenden Nachricht als 'msg.count' an. Wir werden die globale Variable zu einem späteren Zeitpunkt in anderen Function Nodes wiederverwenden.

```
//Code für den Function Node 'globale Variable'
1 var global = flow.get('save')||0; //initialisiere
2
3 if (global > 3){
4     flow.set('save','0'); //setze zurück
5     msg.count= 0;
6     return msg;
7     } else{
8         global++; //inkrementiere
9         flow.set('save', global);
10       }
11
12 msg.count = global;
13 return msg;
```

Es folgt ein weiterer Switch Node, den wir 'Wetter-Routing' nennen. Dieser schickt die anliegende Nachricht, abhängig von ihrem 'msg.payload', an unterschiedliche Ausgänge. Wir nutzen die 'contains'-Option im Konfigurationsdialog, um die eintreffenden Nachrichten in Abhängigkeit der Wetterlage voneinander zu trennen. Wenn 'msg.payload' den String 'Sun' enthält, wird die Nachricht an den ersten Ausgang gesendet, wenn 'msg.payload' den String 'Cloud' enthält, wird die Nachricht an den zweiten Ausgang gesendet und so weiter. Wir führen diese Konfiguration für die Strings 'Sun', 'Cloud', 'Breez' (für 'Breezy' und 'Breeze'), 'Rain', 'Shower' und 'Thunder' durch. Für die letzte Regel nutzen wir die Option 'otherwise', um nicht zutreffende Fälle zu berücksichtigen:

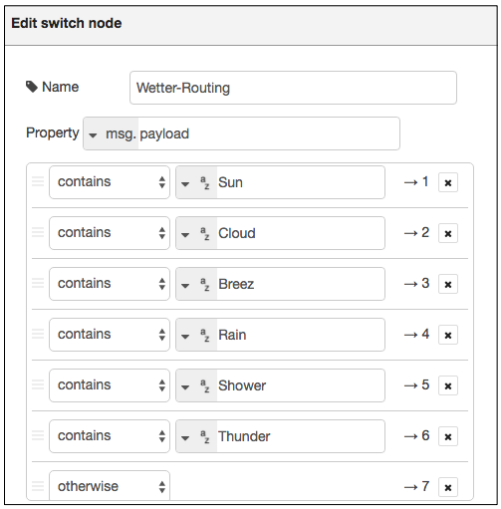


Abbildung 51: Konfiguration des zweiten Switch Nodes

Die Twitter-API verbietet identische Tweets innerhalb eines bestimmten Zeitfensters. Deshalb verfassen wir im nächsten Schritt verschiedene Sätze, die eine bestimmte Wetterlage beschreiben, um sie später auf Twitter zu veröffentlichen. Dies geschieht in sechs Function Nodes. Im ersten Function Node werden sonnige Tweets erstellt, im zweiten werden wolkige Tweets erstellt, im dritten windige Tweets und so weiter. Wir verbinden jeden Ausgang des vorherigen Switch Nodes mit dem jeweils passenden Function Node: Wenn es sonnig ist, leitet der Switch Node die Nachricht an seinen ersten Ausgang. Entsprechend verbinden wir diesen ersten Ausgang mit dem Eingang des Function Nodes, in dem sonnige Tweets erstellt werden und so weiter. Der Code in den sechs Function Nodes unterscheidet sich nur in den Sätzen, der Rest ist jeweils identisch. Nur der letzte Function Node 'Mal sehen wie's wird-Tweets' arbeitet nicht mit der Höchsttemperatur.

In den Zeilen eins bis fünf formulieren wir fünf Sätze zum Wetter und speichern sie in fünf Variablen 'tweet0', 'tweet1',...,'tweet4'. Dann fertigen wir einen weiteren Satz an, den wir in der Variable 'add' speichern. Dieser Satz enthält die Höchsttemperatur, auf die wir mittels der Eigenschaft 'msg.temp' des anliegenden Nachrichten-Objektes zugreifen (Zeile sechs). Anschließend bilden wir ein Array 'tweetbuffer', in dem die Sätze abgelegt und aneinandergefügt werden (Zeile sieben). Nun fragen wir mit der Methode 'get()' den momentanen Wert der globalen Variable 'save' ab und speichern ihn in der Variable 'zaehle' (Zeile 9). Abhängig von 'zaehle' wird ein Satzpaar aus dem Array 'tweetbuffer' geladen. Das anliegende 'msg.payload' wird dann mit diesem Satzpaar überschrieben (Zeile zehn). Abschließend stellen wir das aktualisierte Nachrichten-Objekt am Ausgang des Function Nodes bereit (Zeile elf).
So haben wir erreicht, dass bei jedem Druck auf den Taster ein anderes Satzpaar in 'msg.payload' gespeichert wird:

```
//Code für den Function Node 'Sonnige Tweets'
1 var tweet0 = "In Berlin wird es heute sonnig.";
2 var tweet1 = "Heute scheint die Sonne in Berlin.";
3 var tweet2 = "In Berlin wird heut' die Sonne scheinen.";
4 var tweet3 = "In Berlin lacht heut' die Sonne.";
5 var tweet4 = "Berlin zeigt sich heut' von seiner sonnigen Seite.";
6 var add = " Die Höchsttemperatur liegt bei " + msg.temp +" °C.";
7 var tweetbuffer = [tweet0+add, tweet1+add, tweet2+add, tweet3+add, tweet4+add];
8
9 var zaehle = flow.get('save');
10 msg.payload = tweetbuffer[zaehle];
11 return msg;
```

```
//Code für den Function Node 'Wolkige Tweets'
var tweet0 = "Berlin wird heute von Wolken verdeckt.";
var tweet1 = "Heute wird es wolkig in Berlin.";
var tweet2 = "Berlin ist heute voll von Wolken.";
var tweet3 = "In Berlin ist es heute bedeckt.";
var tweet4 = "Berlin ist heute wolkenbedeckt.";
var add = " Die Höchsttemperatur liegt bei " + msg.temp +" °C.";
var tweetbuffer = [tweet0+add, tweet1+add, tweet2+add, tweet3+add, tweet4+add];

var zaehle = flow.get('save');
msg.payload = tweetbuffer[zaehle];
return msg;
```

```
//Code für den Function Node 'Windige Tweets'
var tweet0 = "In Berlin weht heut' der Wind.";
var tweet1 = "Heute wird's windig in Berlin.";
var tweet2 = "In Berlin kann man heute segeln gehen.";
var tweet3 = "Gutes Wetter zum Segeln in Berlin.";
var tweet4 = "In Berlin weht heut' ne steife Brise.";
var add = " Die Höchsttemperatur liegt bei " + msg.temp +" °C.";
var tweetbuffer = [tweet0+add, tweet1+add, tweet2+add, tweet3+add, tweet4+add];

var zaehle = flow.get('save');
msg.payload = tweetbuffer[zaehle];
return msg;
```

```
//Code für den Function Node 'Regnerische Tweets'
var tweet0 = "Heute wird's nass in Berlin.";
var tweet1 = "In Berlin wird es heute regnen.";
var tweet2 = "In Berlin wird es heute regnerisch.";
var tweet3 = "Heute lieber Bahn fahren, es wird nass.";
var tweet4 = "In Berlin gibt's heute Regen.";
var add = " Die Höchsttemperatur liegt bei " + msg.temp +" °C.";
var tweetbuffer = [tweet0+add, tweet1+add, tweet2+add, tweet3+add, tweet4+add];

var zaehle = flow.get('save');
msg.payload = tweetbuffer[zaehle];
return msg;
```

```
//Code für den Function Node 'Unwetter-Tweets'
var tweet0 = "In Berlin wird's heute gewittern.";
var tweet1 = "Heute gibt's Blitz und Donner in Berlin.";
var tweet2 = "In Berlin gibt's heut' ein Unwetter.";
var tweet3 = "Heute wird's in Berlin gewittern.";
var tweet4 = "In Berlin wird's heute ungemütlich.";
var add = " Die Höchsttemperatur liegt bei " + msg.temp +" °C.";
var tweetbuffer = [tweet0+add, tweet1+add, tweet2+add, tweet3+add, tweet4+add];

var zaehle = flow.get('save');
msg.payload = tweetbuffer[zaehle];
return msg;
```

```
//Code für den Function Node 'Mal sehen wie's wird-Tweets'
var tweet0 = "Mal sehen wie das Wetter heute wird.";
var tweet1 = "Wie wird das Wetter heute?";
var tweet2 = "Auf die Wettervorhersage ist (n)immer verlass.";
var tweet3 = "In Berlin ist's immer schön.";
var tweet4 = "Es gibt kein schlechtes Wetter, nur falsche Kleidung.";
var tweetbuffer = [tweet0, tweet1, tweet2, tweet3, tweet4];

var zaehle = flow.get('save');
msg.payload = tweetbuffer[zaehle];
return msg;
```

Nachdem wir unser finales Nachrichten-Objekt mit einem Debug Node überprüft haben, verbinden wir alle sechs Function Nodes ebenso mit einem Twitter Out Node. Dieser twittert den Inhalt des empfangenen 'msg.payload' und somit bei einem Druck auf den Taster das heutige Wetter in Berlin:

Abbildung 52: Wettermeldung auf Twitter

5) Weiteres

5.1) Debug Tab-Länge verändern

Manche Nachrichten-Objekte sind so umfangreich, dass man sie im Debug-Bereich nicht vollständig lesen kann. Um dies zu verhindern, ändert man einen Eintrag in der Datei 'settings.js'. Dafür zunächst in das Node-RED-Verzeichnis wechseln:

Aus dem home-Verzeichnis:

```
cd \.node-red
```

Aus einem höheren Verzeichnis:

```
cd home/pi/.node-red
```

Datei 'settings.js' öffnen:

```
sudo nano settings.js
```

Im oberen Viertel die Anzahl angezeigter Zeichen erhöhen:

```
>> debugMaxLength: 6000; <<
```

Änderungen sichern:

ctrl → x → y → Enter

Danach ist ein Neustart von Node-RED notwendig [14].

5.2) Autostart bei Boot

Die folgenden Befehle verhelfen zum automatischen Start von Node-RED während des Boot-Vorgangs.

5.2.1) Raspbian Jessie

Node-RED-Version 0.12.1 (Raspbian Jessie November 2015):

```
sudo update-rc.d nodered defaults
```

Node-RED-Version 0.12.5 und höher:

```
sudo systemctl enable nodered.service
```

Autostart deaktivieren:

```
sudo systemctl disable nodered.service
```

Welche Version installiert ist, kann man nach dem Start von Node-RED im Terminal ablesen. Alternativ wird die installierte Version auch im Dropdown-Menü oben rechts im Flow-Editor angezeigt [10].

5.2.2) Raspbian Wheezy

Für den Autostart von Node-RED ist unter Raspbian Wheezy ein Skript im init.d-Verzeichnis erforderlich. Hier stehen Skripte, die verschiedene Dienste starten, stoppen oder neu starten. Das Skript erfordert die Installation von 'screen':

```
sudo apt-get install screen
```

Erzeuge eine neue Datei 'node_red' im init.d-Verzeichnis:

```
sudo nano /etc/init.d/node_red
```

Kopiere das folgende Skript und füge es in die gerade erzeugte Datei ein. Dieses Skript wird Node-RED als Benutzer 'pi' ausführen:

```
#! /bin/sh

# Starts and stops Node-RED

# /etc/init.d/node_red

### BEGIN INIT INFO

# Provides:     node_red

# Required-Start:       $syslog

# Required-Stop:        $syslog

# Default-Start:        2 3 4 5

# Default-Stop:         0 1 6

# Short-Description:    Node-RED initialisation

### END INIT INFO

# Note: this runs as the user called pi
```

```
PIDFILE=/var/run/nodered.pid

#Load up node red when called
case "$1" in

start)
        echo "Starting Node-Red.."
        su -l pi -c "screen -dmS red node-red-pi --max-old-space-size=64 -v"
        echo `screen -ls red | sed -n '2p' | cut -f1 -d.` > $PIDFILE
;;

stop)
        echo "Stopping Node-Red.."
        su -l pi -c "screen -S red -X quit"
;;

restart)
        echo "Restarting Node-Red.."
        $0 stop
        $0 start
;;
*)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac
```

Speichere das Skript:

ctrl → x → y → Enter

Mache das Skript ausführbar:

```
sudo chmod +x /etc/init.d/node_red
```

Füge das Skript in die korrekten Runlevels ein:

```
sudo update-rc.d node_red defaults
```

Node-RED wird nun gestartet, wenn der Raspberry Pi hochfährt und beendet, sobald der Raspberry Pi herunterfährt. Für das manuelle Starten und Stoppen von Node-RED sind nun folgende Befehle vorhanden:

```
sudo service node_red start
sudo service node_red stop
sudo service node_red restart
```

Der Befehl

```
sudo update-rc.d -f node_red remove
```

entfernt das Skript aus dem Autostart und löscht sämtliche Verknüpfungen zu /etc/init.d_node_red aus den Runlevels [11] [15] [16].

5.3) Subflows

Ein Subflow fasst mehrere Nodes zu einem Node zusammen und macht komplexe Flows übersichtlicher. Subflow-Nodes sind dann dauerhaft links in der Node-Palette verfügbar. Es gibt zwei Möglichkeiten, einen Subflow zu erstellen:

Möglichkeit 1: Einen leeren Subflow erstellen

In dem Flow-Editor oben rechts auf das Dropdown-Menü klicken und 'Sublows → Create Subflow' wählen. Es öffnet sich ein neuer Tab mit dem Namen 'Subflow 1'. Hier kann nun wie gewohnt per Drag and Drop ein Subflow erstellt werden. Die Anzahl der Ein- und Ausgänge des neuen Subflow-Nodes stellt man per Mausklick ein, ebenso lässt sich der Subflow einfach wieder löschen:

Abbildung 53: Erstellen eines leeren Subflows

Gleichzeitig erscheint in der Node-Palette eine neue Kategorie 'subflows', in welcher neue Subflow-Nodes zu finden sind:

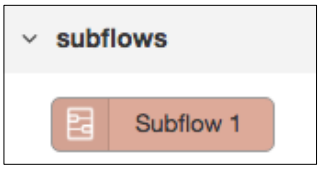


Abbildung 54: Kategorie für Subflow-Nodes

Möglichkeit 2: Einen Subflow aus vorhandenen Nodes erstellen

Dafür zuerst Nodes markieren und im Dropdown-Menü 'Subflows → Selection to Subflow' wählen.
Dies fasst die markierten Nodes zu einem Subflow zusammen. Wie zuvor erscheint ein neuer Subflow-Node. Nach einem Doppelklick hierauf kann man die in Abbildung 53 gezeigten Einstellungen vornehmen.

5.4) Installation und Test des MQTT-Brokers 'Mosquitto'

Message Queue Telemetry Transport (MQTT) ist als Protokoll für das Internet der Dinge standardisiert [17]. Hierbei kommunizieren Clients mit einem Broker. Clients können entweder Publisher oder Subscriber sein. Ein Publisher veröffentlicht Nachrichten auf einem Topic. Der Broker verteilt dann diese Nachrichten an alle Subscriber, die jenes Topic abonniert haben [18]. Die folgenden Befehle installieren den MQTT-Broker 'Mosquitto' auf dem Raspberry Pi.

Wechsle in das Home-Verzeichnis:

```
cd /home/pi
```

Lade den Signaturschlüssel für die Paketquelle (englisch *Repository*):

```
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
rm mosquitto-repo.gpg.key
```

Lade die Paketquelle und stelle sie dem Advanced Package Tool (apt) zur Verfügung:

```
cd /etc/apt/sources.list.d/
```

Für Jessie:

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.list
```

Für Wheezy:

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-wheezy.list
```

Lese alle Paketlisten neu ein:

```
sudo apt-get update
```

Stelle fest, welche Mosquitto-Pakete verfügbar sind:

```
apt-cache search mosquitto
```

Installiere den Broker, Clients für die Kommandozeile und Python Language Bindings:

```
sudo apt-get install mosquitto mosquitto-clients python-mosquitto
```

Der Broker wird automatisch gestartet. Prüfe den Service-Status, Prozess und Default Port 1883:

```
service mosquitto status
ps -ef | grep mosq
netstat -tln | grep 1883
```

Es gibt drei Möglichkeiten für das manuelle Starten und Stoppen des Brokers:

Möglichkeit 1:

```
sudo service mosquitto start
sudo service mosquitto stop
```

Möglichkeit 2:

```
sudo /etc/init.d/mosquitto start
sudo /etc/init.d/mosquitto stop
```

Möglichkeit 3:

```
mosquitto
```

Die Tastenkombination 'ctrl → c' stoppt hier den Broker.

Der folgende Befehl deinstalliert den Broker, die Clients für die Kommandozeile und die Python Language Bindings wieder:

```
sudo apt-get purge mosquitto mosquitto-clients python-mosquitto
```

Folgender Befehl tut dasselbe, löscht aber zusätzlich alle zugehörigen Konfigurationsdateien:

```
sudo apt-get --purge remove mosquitto mosquitto-clients python-mosquitto
```

Testen des Brokers mit den Kommandozeilen-Clients:

Für einen ersten Test des Brokers öffnen wir zwei neue Terminal-Fenster A und B.

Schritt 1: Mit dem Terminal A als Subscriber das Topic 'jakob12/affe' abonnieren:

```
mosquitto_sub -d -t jakob12/affe
```

Schritt 2: Mit dem Terminal B als Publisher auf dem Topic 'jakob12/affe' veröffentlichen:

```
mosquitto_pub -d -t jakob12/affe -m "Test-Nachricht"
```

Im Terminal-Fenster A erscheint 'Test-Nachricht'. Der Broker leitet also wie gewünscht die Nachricht von Publisher zu Subscriber [19] [20] [21] [22] [23] [24] [25] [26] [27] [28]:

```
pi@raspberrypi:~ $ mosquitto_sub -d -t jakob12/affe
Client mosqsub/1271-raspberryp sending CONNECT
Client mosqsub/1271-raspberryp received CONNACK
Client mosqsub/1271-raspberryp sending SUBSCRIBE (Mid: 1, Topic: jakob12/affe, QoS: 0)
Client mosqsub/1271-raspberryp received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/1271-raspberryp received PUBLISH (d0, q0, r0, m0, 'jakob12/affe', ... (14 bytes))
Test-Nachricht
```

Abbildung 55: Terminal A als MQTT-Subscriber

```
pi@raspberrypi:~ $ mosquitto_pub -d -t jakob12/affe -m "Test-Nachricht"
Client mosqpub/1272-raspberryp sending CONNECT
Client mosqpub/1272-raspberryp received CONNACK
Client mosqpub/1272-raspberryp sending PUBLISH (d0, q0, r0, m1, 'jakob12/affe', ... (14 bytes))
Client mosqpub/1272-raspberryp sending DISCONNECT
pi@raspberrypi:~ $
```

Abbildung 56: Terminal B als MQTT-Publisher

Testen des Brokers mit Node-RED:

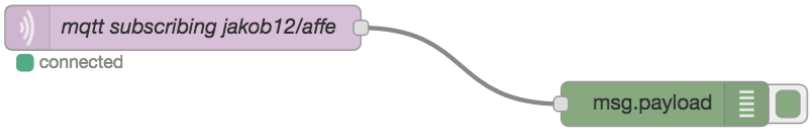


Abbildung 57: Flow als MQTT-Subscriber

Mit einem MQTT In Node wollen wir nun das gleiche Topic abonnieren. Der Broker läuft in diesem Fall auf dem Raspberry Pi selbst, also konfigurieren wir als Server 'localhost' und den per Default verwendeten Port '1883':

Abbildung 58: Konfiguration des MQTT In Nodes

Nach dem Ausführen des Flows zeigt der MQTT In Node mit 'connected' eine bestehende Verbindung an. Zum Testen veröffentlichen wir eine neue Nachricht vom Terminal B aus:

```
pi@raspberrypi:~ $ mosquitto_pub -d -t jakob12/affe -m "Nachricht an Node-RED"
Client mosqpub/1310-raspberryp sending CONNECT
Client mosqpub/1310-raspberryp received CONNACK
Client mosqpub/1310-raspberryp sending PUBLISH (d0, q0, r0, m1, 'jakob12/affe', ... (21 bytes))
Client mosqpub/1310-raspberryp sending DISCONNECT
pi@raspberrypi:~ $
```

Abbildung 59: Terminal B als MQTT-Publisher

Die Nachricht erscheint im Debug-Bereich des Flow-Editors:

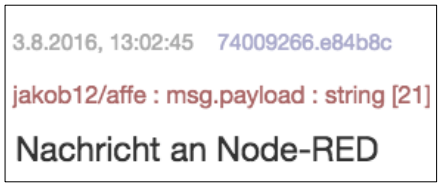


Abbildung 60: Empfangene MQTT-Nachricht in Node-RED

Die Nachricht gelangt ebenfalls an den Kommandozeilen-Client auf dem Raspberry Pi:

```
Client mosqsub/1271-raspberryp received PUBLISH (d0, q0, r0, m0, 'jakob12/affe', ... (21 bytes))
Nachricht an Node-RED
```

Abbildung 61: Terminal A als MQTT-Subscriber

In einem weiteren Schritt wollen wir von Node-RED aus eine MQTT-Nachricht veröffentlichen. Dazu erweitern wir den Flow um einen MQTT Out Node und einen Inject Node:

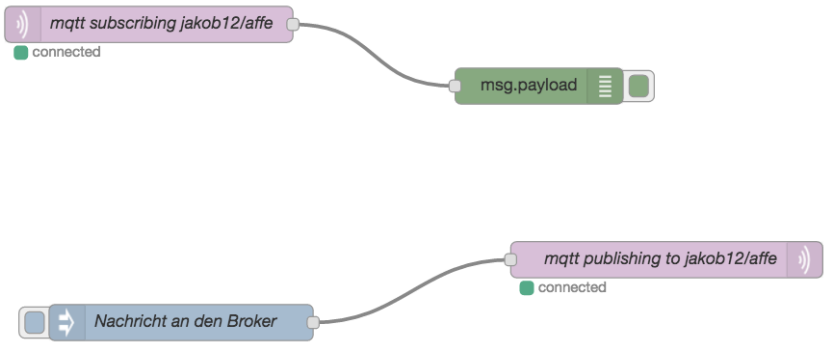


Abbildung 62: Erweiterter MQTT-Flow

Statt 'localhost' geben wir bei der Konfiguration des MQTT Out Nodes diesmal explizit die IP-Adresse des Raspberry Pis an. Mit dem Inject Node generieren wir eine 'Nachricht an den Broker':

Abbildung 63: Konfiguration des MQTT Out Nodes

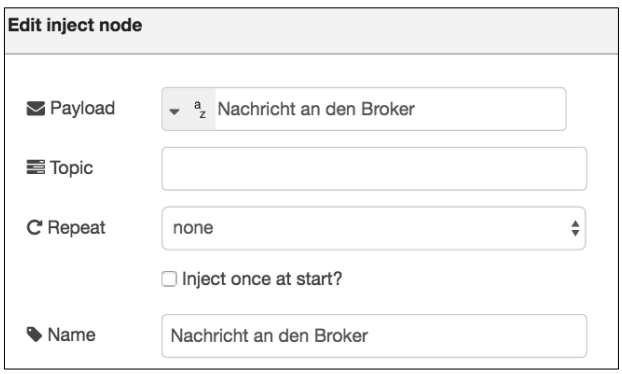


Abbildung 64: Konfiguration des Inject Nodes

Nun die neuen Nodes verbinden und den Flow noch einmal ausführen. Mit einem Klick auf den Inject Node gelangt die Nachricht an den Kommandozeilen-Client:

```
Client mosqsub/1271-raspberryp received PUBLISH (d0, q0, r0, m0, 'jakob12/affe', ... (23 bytes))
Nachricht an den Broker
```

Abbildung 65: Terminal A als MQTT-Subscriber

Und auch Node-RED empfängt die Nachricht wieder über den MQTT In Node [29]:

Abbildung 66: Empfangene MQTT-Nachricht in Node-RED

6) Fazit und Ausblick

Diese Einführung zeigt, wie man in Node-RED auf einfache Art und Weise Hardware mit Webdiensten und Programmierschnittstellen verbinden kann. Dabei sind nur geringe Programmierkenntnisse erforderlich. Somit sind auch für Einsteiger IoT-Anwendungen realisierbar. Man könnte beispielsweise mit dem Mikrocontroller ESP8266 eine Lampe in das WLAN integrieren. Node-RED und der Raspberry Pi ermöglichen die Kommunikation mit der Lampe über MQTT. In nächsten Schritten können weitere MQTT-Clients eingebunden werden. Ein Beispiel wäre Owntracks, eine App für das Smartphone, welche auf einem bestimmten Topic fortlaufend Koordinaten sendet. In Verbindung mit dem Geofence Node und einer Konfiguration des Heim-Routers wäre eine Standorterkennung denkbar. Dies könnte man zu einem Szenario verbinden, sodass sich die Lampe einschaltet, wenn man abends nach Hause kommt.

Literaturverzeichnis

1: https://de.wikipedia.org/wiki/Computerwurm, Zugriff: 27. September 2016
2: http://noderedguide.com/index.php/2015/10/01/nr-lecture-1/, Zugriff: Juli 2016
3: http://nodered.org/docs/hardware/arduino, Zugriff: 9. August 2016
4: http://nodered.org/docs/hardware/beagleboneblack, Zugriff: 9. August 2016
5: http://noderedguide.com/index.php/2015/11/06/node-red-lecture-5-the-node-red-programming-model/#ftnt1, Zugriff: Juni 2016
6: http://noderedguide.com/index.php/2015/10/28/node-red-lecture-2-building-your-first-flows-15/, Zugriff: Juni 2016
7: http://noderedguide.com/index.php/2015/10/05/node-red-lecture-4-a-tour-of-the-core-nodes/, Zugriff: Juni 2016
8: http://nodered.org/docs/writing-functions, Zugriff: Juni 2016
9: https://github.com/node-red/node-red/wiki/Node-msg-Conventions, Zugriff: Juni 2016
10: http://nodered.org/docs/hardware/raspberrypi , Zugriff: Juni 2016
11: https://swabbster.wordpress.com/2015/04/16/raspberry-pi2-node-red-setup/, Zugriff: Juni 2016
12: https://www.npmjs.com/package/node-red-contrib-ui, Zugriff: Juli 2016
13: http://www.computerhope.com/unix/ucd.htm, Zugriff: Juli 2016
14: http://stackoverflow.com/questions/31274637/see-full-msg-debug-with-node-red, Zugriff: 29. Juli 2016
15: https://learn.adafruit.com/raspberry-pi-hosting-node-red/managing-node-red, Zugriff: 28. Juli 2016
16: https://wiki.ubuntuusers.de/SysVinit/, Zugriff: 28. Juli 2016
17: http://dennisseidel.de/mqtt-eine-einfuhrung/, Zugriff: Juli 2016
18: http://mosquitto.org/man/mqtt-7.html, Zugriff: Juli 2016
19: https://www.youtube.com/watch?v=Ptk4A85EgF8, Zugriff: 2. August 2016
20: https://iotbytes.wordpress.com/mosquitto-mqtt-broker-on-raspberry-pi/, Zugriff: 2. August 2016
21: http://logbuch.dmaertens.de/openhab/mqtt/mosquitto-auf-einem-raspberry-pi-installieren, Zugriff: 2. August 2016
22: https://mosquitto.org/2013/01/mosquitto-debian-repository/, Zugriff: 2. August 2016
23: http://www.switchdoc.com/2016/02/tutorial-installing-and-testing-mosquitto-mqtt-on-raspberry-pi/, Zugriff: 2. August 2016
24: http://jpmens.net/2013/09/01/installing-mosquitto-on-a-raspberry-pi/, Zugriff: 2. August 2016
25: https://wiki.ubuntuusers.de/apt/apt-get/, Zugriff: 21. August 2016
26: https://wiki.ubuntuusers.de/apt/apt-key/, Zugriff: 21. August 2016
27: https://wiki.ubuntuusers.de/sources.list/, Zugriff: 21. August 2016
28: https://de.wikipedia.org/wiki/Repository, Zugriff: 21. August 2016
29: http://www.rs-online.com/designspark/electronics/deu/blog/building-distributed-node-red-applications-with-mqtt, Zugriff: 3. August 2016

Abbildungsverzeichnis

Quellenangaben verwendeter Bilder

Titelbild: http://core0.staticworld.net/images/article/2016/05/raspberry-pi-2-node-red-100663608-primary.idge.png, Zugriff: 5. August 2016

Aus Abbildung 1, Zugriff jeweils 8. August 2016:

Node-RED-Logo: https://iotfuse.com/wp-content/uploads/2015/12/node-red-new.jpg

Flow-Editor: https://developer.ibm.com/bluemix/wp-content/uploads/sites/20/2015/02/nodered.png

Smartphone-Piktrogramm: http://www.clipartsfree.de/images/joomgallery/originals/schwarz_weiss_34/smartphone_icon_20140820_1177113301.png

Wolke: http://images.clipartpanda.com/cloud-clipart-best-cloud-clipart.jpeg

IBM Watson-Logo: https://developer.ibm.com/watson/wpcontent/uploads/sites/19/2015/01/IBM_Watson_avatar_pos.png

MongoDB-Logo: https://webassets.mongodb.com/_com_assets/cms/mongodb-logo-rgb-j6w271g1xn.jpg

Twitter-Logo: http://logos- download.com/wp- content/uploads/2016/02/Twitter_logo_bird_transparent_png.png

GroveStreams-Logo: http://www.programmableweb.com/sites/default/files/styles/facebook_scale_height_200/public/GS_Logo2014.jpg?itok=2pLQXGTU

Sofia2-Logo: http://sofia2.com/img/logo_sofia2.png

Facebook-Logo: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c2/F_icon.svg/2000px- F_icon.svg.png

DHT11: https://img.conrad.de/medias/global/ce/2000_2999/2200/2270/2271/1405544_LB_00_FB.EPS.jpg

ESP8266: http://statics3.seeedstudio.com/images/product/WiFi%20Serial%20Transceiver%20Module.jpg

DS18B20: https://tushev.org/images/electronics/arduino/ds18x20/DS18B20.jpg

PIR: https://www.parallax.com/sites/default/files/styles/full-size-product/public/555-28027_1.png?itok=j5y-TYlRT

Aus den Abbildungen 1, 17, 30, 46, Zugriff jeweils Juni 2016:

Raspberry Pi-Logo: https://upload.wikimedia.org/wikipedia/en/thumb/c/cb/Raspberry_Pi_Logo.svg/810px-Raspberry_Pi_Logo.svg.png

Raspberry Pi-Pinout: http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/#prettyPhoto/0/