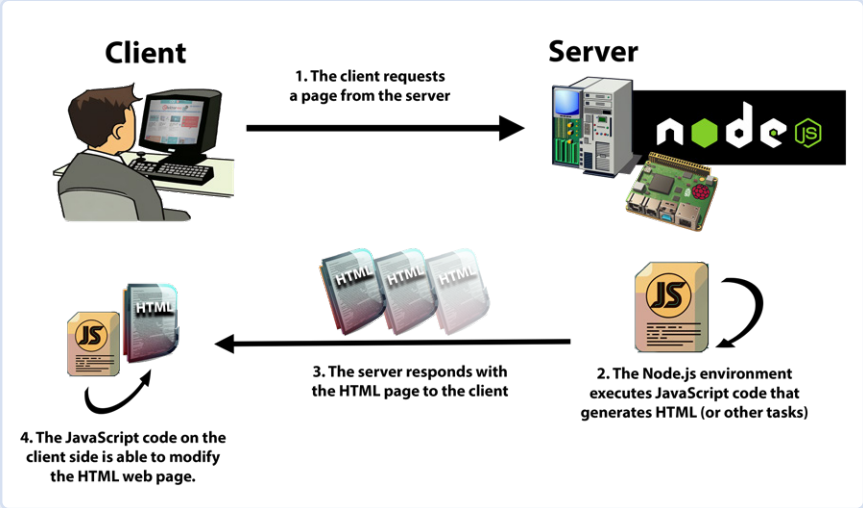
I²C-communicatie met Node.js en een Raspberry Pi

bekijk sensordata in een browser

Franck Bigrat (Frankrijk)

De I²C-bus bestaat al sinds de jaren '80 en wordt goed ondersteund op de huidige populaire platforms, waaronder de Raspberry Pi-serie. Node.js maakt het mogelijk om een webserver in JavaScript te programmeren die gegevens naar een lokaal netwerk of het internet stuurt. Dit artikel laat zien hoe eenvoudig het is om een I²C-sensor aan te sluiten op een Raspberry Pi en de resultaten weer te geven in een webbrowser.



Figuur 1. Node.js server-communicatieflow (bron: sdz.tdct.org).

In dit project gebruiken we een simpele Raspberry Pi Zero en de DS1621 I²C-temperatuursensor. Het meetresultaat wordt weergegeven op een webpagina en is zichtbaar op een tablet of smartphone die op hetzelfde WiFi-netwerk is aangesloten.
Node.js heeft modules zoals *http* en *express*, waarmee de programmeur heel eenvoudig webservers kan bouwen, zodat we de alomtegenwoordige Apache-server niet nodig hebben.
Hoewel er een module bestaat voor de DS1621-sensor, is het interessanter om de *i2c-bus* module van Node.js te gebruiken, waarmee we het project kunnen uitbreiden naar een groot aantal andere I²C-apparaten.

Wat is Node.js?

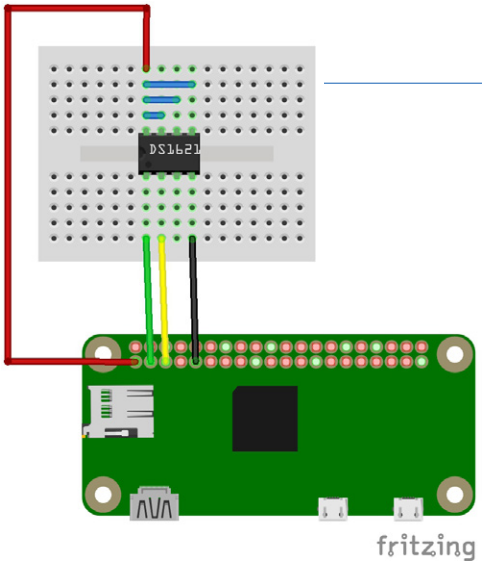
Node.js in een paar woorden beschrijven is niet zo eenvoudig.
Er zijn veel boeken en websites gewijd aan Node.js; hier moet je gewoon begrijpen dat Node.js *JavaScript is dat wordt uitgevoerd door de webserver, dus niet door een webbrowser die op een client draait* (maar de browser kan natuurlijk ook JavaScript-code uitvoeren als dat nodig is). Het is gebouwd op de JavaScript-engine van Google Chrome (V8). Eenvoudig gezegd schrijft de programmeur JavaScript-toepassingen, maar deze draaien op de server. In **figuur 1** is een en ander aanschouwelijk gemaakt.
Om (bijna) alles over Node.js te leren, raad ik de lezer sterk aan de websites op [1] en [2] te bezoeken.

De lezer kan ook deze boeken raadplegen:

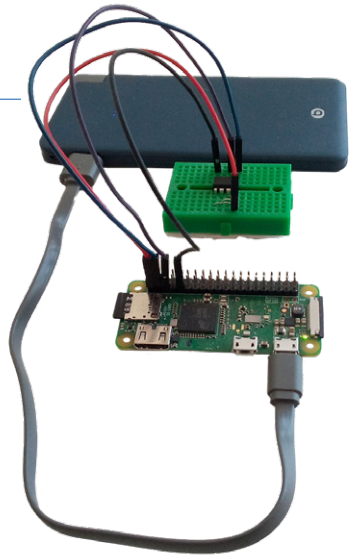
- *Node.js in Practice* door Alex Young en Marc Harter (Manning);
- *Beginning Node.js* door Basarat Ali Sayed (Apress);
- *The Node Beginner Book* door Manuel Kiessling;
- *Instant Node.js Starter* door Pedro Teixeira (Packt Publishing).

I²C en SMBus

In deze paragraaf ga ik ervan uit dat de lezer enigszins bekend is met het I²C-protocol (Inter-Integrated Circuit). Het belangrijkste is dat I²C werd ontwikkeld door Philips in de jaren '80 om het aantal verbindingen tussen een microprocessor en verschillende andere geïntegreerde schakelingen te minimaliseren. Het is een serieel, tweedraads (data en klok) synchroon datatransmissie-protocol. Verschillende schakelingen



Figuur 2. Zo wordt de Raspberry Pi Zero W op de DS1621 aangesloten.



Figuur 3. De praktische uitvoering, inclusief batterij.

kunnen tegelijkertijd op dezelfde data- en kloklijnen worden aangesloten.
Details over de functionaliteit van de i2c-bus Node.js-module zijn te vinden in [3].
Als je zorgvuldig leest, zie je dat sommige functies een smbus-prefix hebben of deel uitmaken van de SMBus-sectie. SMBus maakt communicatie mogelijk tussen de processor van een computer en verschillende periferie, zoals apparaten voor energiebeheer, temperatuursensoren of ventilatoren.
De Raspberry Pi ondersteunt dit protocol in hardware: de System Management Bus is afgeleid van de I²C-bus, en de twee protocollen zijn redelijk compatibel. Maar er zijn enkele verschillen tussen SMBus en I²C; deze worden uitgelegd in het Texas Instruments-document op [4].

De componenten aansluiten en configureren

Nu wordt het tijd om de Raspberry Pi aan te sluiten op de DS1621 temperatuursensor volgens de layout van **figuur 2**.
Vergeet niet de pinnen 5, 6 en 7 (A0, A1, A2) van de DS1621 aan VCC te leggen om het I²C-adres van de component in te stellen op 4F (hexadecimaal). De praktische uitvoering is te zien in **figuur 3**.

De Raspberry Pi configureren

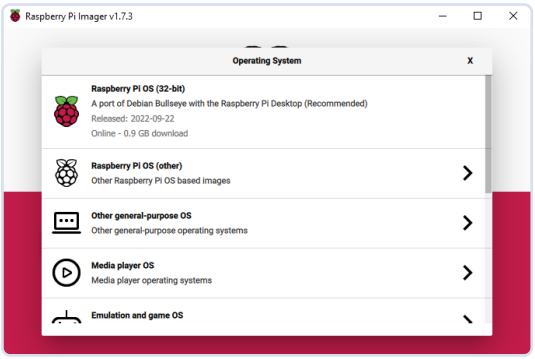
1. Download en installeer Raspberry Pi Imager van: *https://raspberrypi.com/software*

2. Start Raspberry Pi Imager (**figuur 4a**), klik op *CHOOSE OS* en vervolgens op *Raspberry Pi OS (Other)* in de lijst die dan verschijnt (**figuur 4b**), en dan *Raspberry Pi OS Lite (32-bit)* in het submenu.

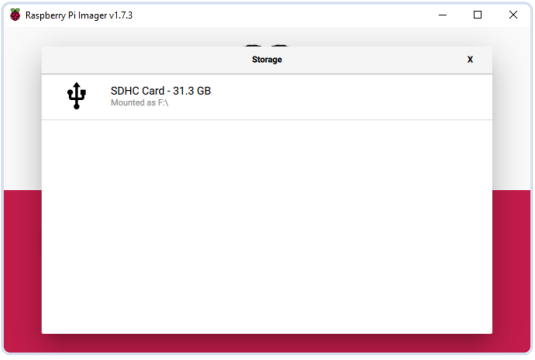
3. Klik op *CHOOSE STORAGE* en selecteer vervolgens je microSD-kaart in het *Storage*-dialoogvenster (**figuur 4c**).

4. Klik op het *Settings*-pictogram (het kleine tandwiel) rechts van de toepassing en stel de initiële configuratie-opties van de Raspberry Pi in:
   - vink *Set hostname:* aan en voer de gewenste hostnaam in; *raspberry* voldoet prima;
   - vink *Enable SSH* aan en selecteer het keuzerondje *Use password authentication*;

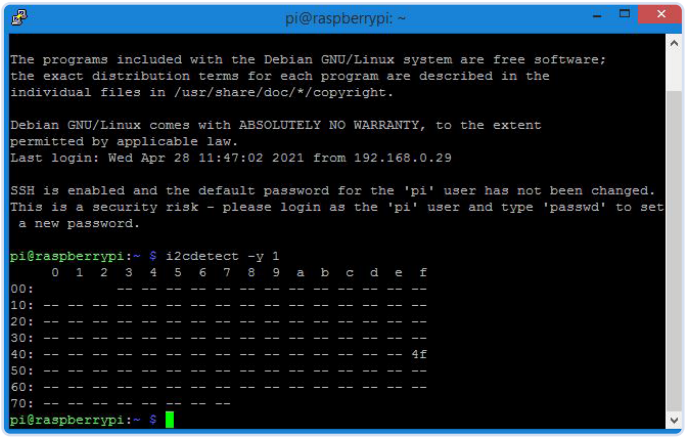
Figuur 4a. Raspberry Pi Imager.



Figuur 4b. Keuze van het besturingssysteem.



Figuur 4c. Keuze van het opslagmedium.



Figuur 5. De uitvoer van het i2cdetect-programma.

- vink *Set username and password* aan en vul de *Username* (*pi* is prima) en een *Password* in (kies er een die je makkelijk kunt onthouden, bijvoorbeeld *raspberry*);
- vink *Configure wireless LAN* aan en voer de SSID en het wachtwoord van je WiFi-netwerk in;
- vink *Set local settings* aan en pas je *Time zone* en *Keyboard layout* indien nodig aan, of laat de standaardinstellingen zoals ze zijn;
- klik op de knop *SAVE*.

5. Installeer het OS door op *WRITE* te klikken (bevestig met *YES*).

6. Als het OS geschreven is is, kun je de microSD-kaart verwijderen.

7. Steek de kaart in de microSD-slot van de Raspberry Pi en schakel die in.

8. Zoek het IP-adres van de Raspberry Pi op het netwerk met een tool zoals Advanced IP Scanner, of (nog beter) haal het uit de DHCP lease-instellingen in je WiFi-router.

9. Gebruik een SSH-client zoals Putty om verbinding te maken met de Raspberry Pi via zijn IP-adres en gebruikersnaam/wachtwoord (in ons voorbeeld respectievelijk *pi* en *raspberry*).

10. Om de volledige opslagruimte van de microSD-kaart te benutten, moet je het bestandssysteem uitbreiden:
    - open het *raspi-config* tool: `sudo raspi-config`;
    - kies *Advanced Options* en vervolgens de optie *Expand Filesystem*;
    - bevestig met de Enter-toets.
    - ga naar <*Finish*> en bevestig met Enter, bevestig dan de herstart door naar <*Yes*> te gaan en op Enter te drukken.

11. Om de I²C-bus te activeren:
    - open het *raspi-config* tool: `sudo raspi-config`;
    - selecteer *Interfacing Options* en dan *I2C* (*Enable/disable automatic loading of I2C kernel module*);
    - bevestig met de Enter-toets;
    - bevestig nogmaals door naar de knop <*Yes*> te gaan en nogmaals op Enter te drukken;
    - sluit af met <*OK*> en dan <*Finish*>.

12. Sluit de Raspberry Pi aan op de DS1621 volgens het schema van figuur 2.

13. Controleer de I²C-communicatie:
    - voer eerst een globale update uit: `sudo apt update`;
    - installeer de I²C-diagnosetools: `sudo apt install i2c-tools` (misschien moet je de Raspberry Pi eerst opnieuw opstarten);
    - voer deze opdracht uit: `i2cdetect -y 1`
    - in de lijst die verschijnt, moet de sensor staan met het adres (4f) dat wij 'hard' hebben ingesteld (**figuur 5**).

14. Installeer nu Node.js en NPM (Node Package Manager): `sudo apt install -y nodejs npm` (dit kan even duren).

15. Controleer de versies met `node -v` gevolgd door `npm -v`.

16. Installeer de *express*, *i2c-bus* en *sleep* modules:
    `npm install express`
    `npm installeer i2c-bus`
    `npm install sleep` (kan waarschuwingen genereren)

17. Maak een map met de naam *Documents* en een submap met de naam *NodeJS* aan.

18. Kopieer met een FTP-client zoals FileZilla de bestanden *DS1621_V3.js* en *DS1621_V4.html* naar de map *NodeJS*.

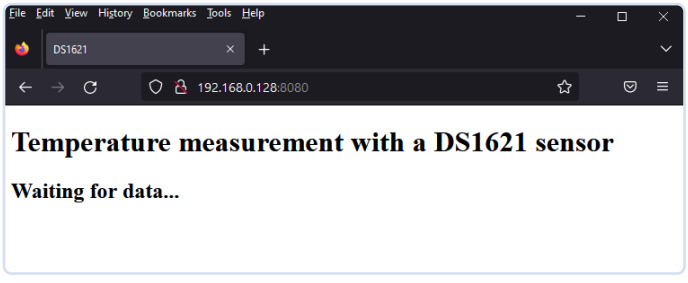
Proberen we het uit!

Zodra alle elementen zijn aangesloten, de Raspberry Pi is geconfigureerd, Node.js en de verschillende modules zijn geïnstalleerd, en de *.js* en *.html* bestanden naar de Raspberry Pi zijn gekopieerd, open je de map *Documents/NodeJS* en voer je het script uit met `node DS1621_V3.js`.
Sluit gewoon een PC, een tablet of een smartphone aan op je WiFi-netwerk, open je favoriete browser en voer dit adres in:

[je_Raspberry_Pi_IP_adres]:8080

Als de vorige stappen correct zijn uitgevoerd, moet je nu iets zien als **figuur 6**.

Na enkele seconden verschijnt de temperatuur (**figuur 7**). Vanaf dan wordt de temperatuur elke vijf seconden ververst.



Figuur 6. Wachten op gegevens.

Figuur 7. De gemeten temperatuur wordt weergegeven.

De Node.js- en HTML-scripts nader bekeken

Belangrijk maar lastig bij Node.js is het begrijpen van de uitvoering van het script. In Node.js is de uitvoering van een script niet sequentieel. Zie **figuur 8**: de instructies in de functie `Server.get('/',...)` worden niet uitgevoerd vóór de instructies in de functie `Server.get('/temp',...)`, die niet worden uitgevoerd vóór de instructies onderaan. De instructies in de functie `Server.get('/',...)` worden alleen uitgevoerd als de server de *[Rpi_IP_Address]:8080*-request van de client ontvangt. De instructies in de functie `Server.get('/temp',...)` worden alleen uitgevoerd als de server de request *[Rpi_IP_Address]:8080/temp* van de client ontvangt. Dit wordt *event-driven programming* genoemd.

In de eerste vijf regels van het Node.js-script (links in figuur 8) importeren we de vereiste Node.js-modules en maken we verschillende objecten aan:

- We maken een `I2C`-object met behulp van de `i2c-bus` module:

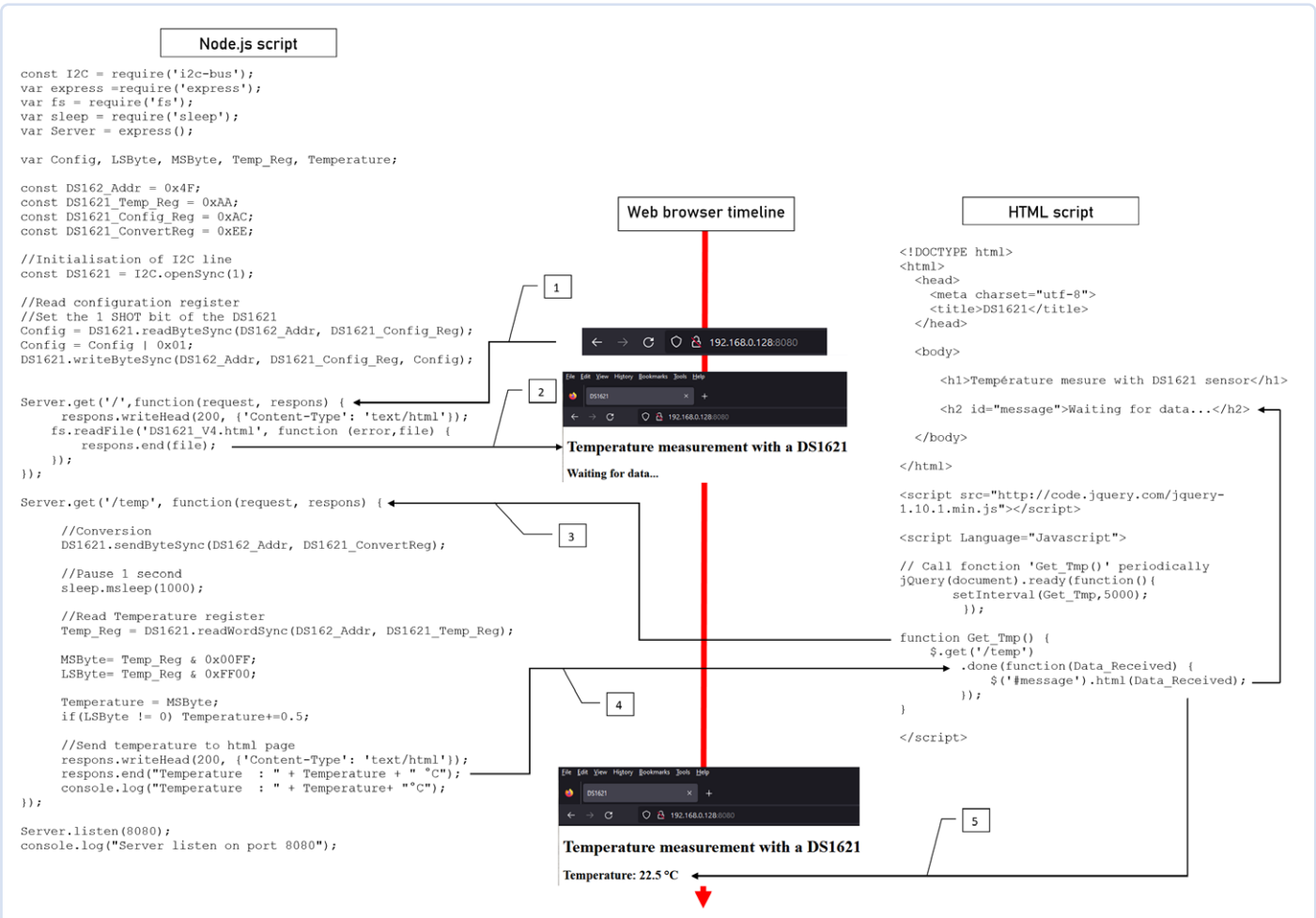
```
const I2C = require('i2c-bus');
```

- We maken een `express`-object aan met behulp van de `express`-module. Deze module is een framework dat is ontworpen voor het bouwen van webapplicaties met Node.js:

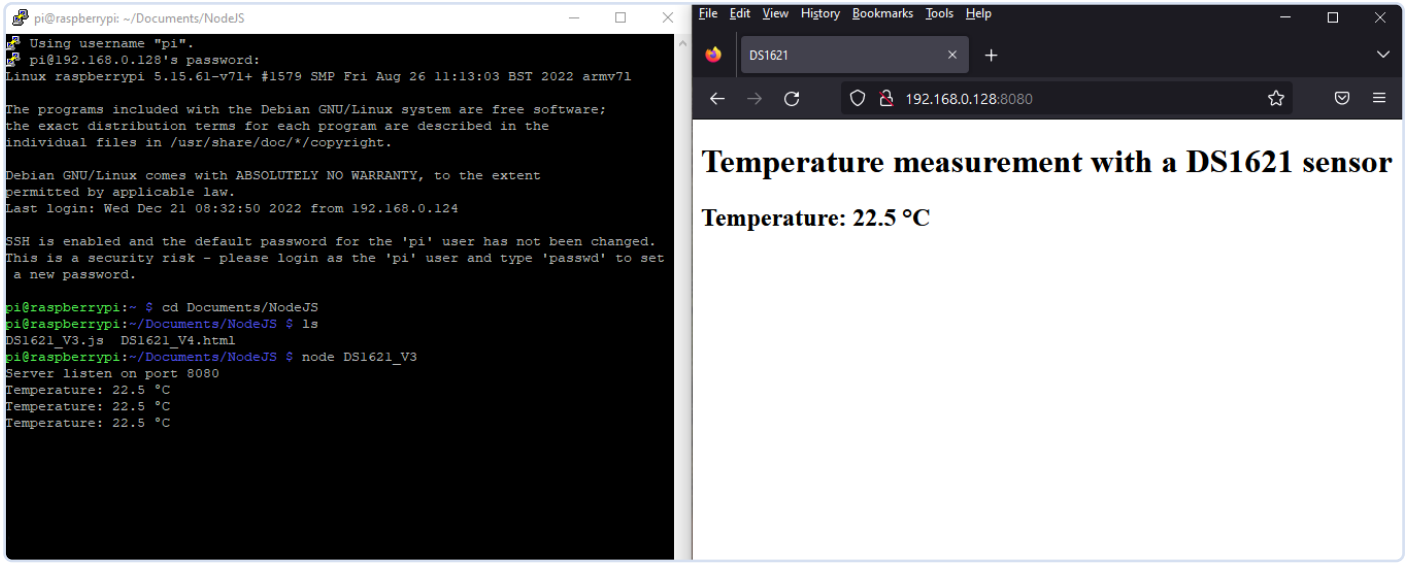
```
var express = require('express');
```

- We maken een `fs`-object aan met de `fs`-module (file system). Hiermee kunnen we bestanden op de microSD-kaart lezen en schrijven:

```
var fs = require('fs');
```



Figuur 8. Tijdslijn van de gebeurtenissen.



Figuur 9. De Linux-terminal en de webbrowser.

› We maken een `sleep`-object met behulp van de `sleep`-module. Met deze module kunnen we vertragingen creëren:

```
var sleep = require('sleep');
```

› Tenslotte maken we een `Server`-object aan:

```
var Server = express();
```

In de volgende vijf regels definiëren we vijf variabelen en vier constanten. Deze constanten bevatten de respectieve adressen van de DS1621-registers.

```
var Config, LSByte, MSByte, Temp_Reg, Temperatuur;

const DS162_Addr = 0x4F;
const DS1621_Temp_Reg = 0xAA;
const DS1621_Config_Reg = 0xAC;
const DS1621_ConvertReg = 0xEE;
```

Vervolgens openen we de I²C-bus en maken we een object met de naam `DS1621`. Dit object zal worden gebruikt om met de sensor te communiceren via I²C.

```
const DS1621 = I2C.openSync(1);
```

De volgende drie regels stellen het *1SHOT*-bit van de DS1621 in. De temperatuur wordt dus één keer (en niet meer dan één keer) gemeten en geconverteerd als de I²C Master (de Raspberry Pi) een conversie nodig heeft.

```
Config = DS1621.readByteSync(DS162_Addr,
DS1621_Config_Reg);
Config = Config | 0x01;
DS1621.writeByteSync(DS162_Addr,
DS1621_Config_Reg, Config);
```

Laten we nu eens kijken wat er gebeurt als de browser een request stuurt naar de server (**figuur 9**).
Wanneer het Node.js-script wordt uitgevoerd, verschijnt na enkele seconden het bericht “Server listen on port 8080” in de Linux-terminal.

Open een webbrowser op je PC of tablet die verbonden is met het WiFi-netwerk. Typ in de adresbalk *http://* en het IP-adres van de Raspberry Pi en de poort waarop de server luistert; bijvoorbeeld *http://192.168.0.128:8080*. Poort 8080 is gekozen omdat die anders is dan de standaard gebruikt door de Apache webserver (poort 80).

1. Wanneer je het IP-adres opgeeft, stuurt de browser een *get* HTTP-request naar de server. Wat gebeurt er?
   a. De server detecteert het verzoek met `Server.get('/'...` en voert de anonieme callback-functie `function(request, respons)` uit.
   b. Het `request`-object bevat het verzoek van de browser.
   c. In het `respons`-object sturen we twee headers mee als parameters:
      i. Een *200* http-code, wat betekent dat het verzoek succesvol was.
      ii. De string “'Content-Type': 'text/html'”, hetgeen betekent dat de browser voorbereid moet zijn om tekstgegevens van het type HTML te ontvangen.
   d. De functie-aanroep `respons.writeHead()` stuurt de parametergegevens terug naar de browser.

2. De functie `fs.readFile()` leest het bestand *DS1621_V4.html*. De anonieme callback-functie `function(error, file)` wordt uitgevoerd en, in het geval dat deze succesvol is (daar gaan we van uit), wordt het bestand gelezen en in het `file`-object geplaatst. De functie-aanroep `respons.end(file)` stuurt de HTML naar de browser, die de webpagina weergeeft. In geval van een fout wordt een foutcode geplaatst in het `error`-object.

De browser heeft nu de webpagina in HTML-formaat. Wat gebeurt er nu? Laten we eens kijken naar het HTML-script (rechts in figuur 8). Het eerste wat we zien is de regel `<h2 id="message">Waiting for data...</h2>`. Dit is een plaatshouder voor de kop waar de temperatuur zal worden geschreven. We zullen later zien hoe.
Het tweede wat we zien is het blok dat volgt. Het is een script geschreven in JavaScript, maar het is client-side, wat betekent dat het wordt uitgevoerd door de browser:

```
<script src="http://code.jquery.com/jquery-1.10.1.min.
js"></script>
```

```
<script Language="Javascript">
  // Call function Get_Tmp() periodically:
  jQuery(document).ready(function() {
         setInterval(Get_Tmp, 5000);
          });

  function Get_Tmp() {
      $.get('/temp')
          .done(function(Data_Received) {
              $('#message').html(Data_Received);
          });
  }
</script>
```

Eerst importeren we vanaf het web de jQuery-bibliotheek, die ons enkele interessante functies biedt. Dankzij jQuery kunnen we een timer instellen die elke 5 seconden een functie genaamd `Get_Temp()` aanroept:

```
  jQuery(document).ready(function(){
       setInterval(Get_Tmp, 5000);
       });
```

Tot slot hebben we de functie `Get_Tmp()`:

```
function Get_Tmp() {
$.get('/temp')
     .done(function(Data_Received) {
          $('#message').html(Data_Received);
     });
}
```

En dat is het interessante deel van het script!
Laten we eens kijken hoe het samenwerkt met het Node.js-script aan de serverkant (zie tijdlijn in het midden van figuur 8):

1. Wanneer de functie `Get_Tmp()` wordt aangeroepen, stuurt hij een *get*-verzoek naar de server. Wat gebeurt er dan?
   a. De server detecteert het verzoek met `Server.get('/temp')` en voert de anonieme callback-functie `function(request, respons)` uit.
   b. De I²C-master (Raspberry Pi) communiceert met de DS1621 via de I²C-bus, vraagt om een dataconversie, leest de data en zet de data om in °C.
   c. Het `request`-object bevat het verzoek van de browser.
   d. In het `respons`-object slaan we weer onze twee parameters op:
      i. De waarde *200*, een code die betekent dat het verzoek succesvol is.
      ii. De string *"'Content-Type': 'text/html'"* betekent dat de browser tekst ontvangt die als HTML moet worden geïnterpreteerd.
   e. De functie `respons.writeHead()` retourneert deze gegevens naar de browser.

2. De functie-aanroep `respons.end("Temperature: " + Temperature + " °C")` stuurt het bericht *"Temperatuur: XX °C"* (een ASCII-string) naar de browser. XX is de door de DS1621 gemeten temperatuur.

3. Deze string wordt in het object `Data_Received` geplaatst.

4. Herinner je je de header-sectie `<h2 id="message">Waiting for data...</h2>`? We zien dat deze header-tag de `id` van `message` heeft, dus wanneer de string wordt ontvangen, wordt dankzij de regel `$('#message').html(Data_Received);` de door de server verzonden string in de header-sectie geplaatst en zal de webpagina de temperatuur tonen.

Ten slotte nog een paar woorden over de twee laatste regels van het Node.js-script:

```
Server.listen(8080);
console.log("Server listen on port 8080");
```

De regel `Server.listen(8080);` start de server, en de regel `console.log("Server listen on port 8080");` schrijft het bericht *"Server listen on port 8080"* naar de Linux-console (shell).
Bestudeer figuur 8 zorgvuldig; ik denk dat alles beter te begrijpen is aan de hand van een tijdlijn.

210367-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de redactie van Elektor via redactie@elektor.com.

Gerelateerde producten

- **Vincent Himpe, *Mastering the I²C Bus* (SKU 15385)**
  www.elektor.nl/15385
- **Raspberry Pi Zero W Starter Kit (SKU 18328)**
  www.elektor.nl/18328

WEBLINKS

[1] Introduction to Node.js: A Beginner's Guide to Node.js and NPM: https://elektor.link/udaynode
[2] What is Node.js: A Comprehensive Guide: https://simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs
[3] 'i2c-bus' module: https://npmjs.com/package/i2c-bus
[4] SMBus Compatibility With an I²C Device: https://ti.com/lit/an/sloa132/sloa132.pdf