Met de vos in het IoT (3)

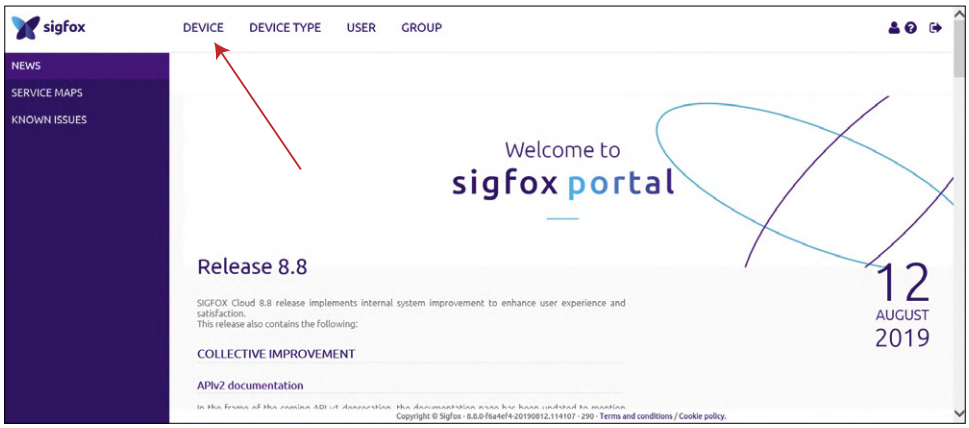
eerste stappen op het net

Frank Schleking en **Bernd vom Berg** (Duitsland)

Het MKR FOX 1200-board is geregistreerd in het Sigfox-netwerk, zodat niets meer in de weg staat voor de eerste poging tot communicatie. Ons Sigfox-device moet nu de beroemde boodschap “Hello World” naar de cloud sturen.

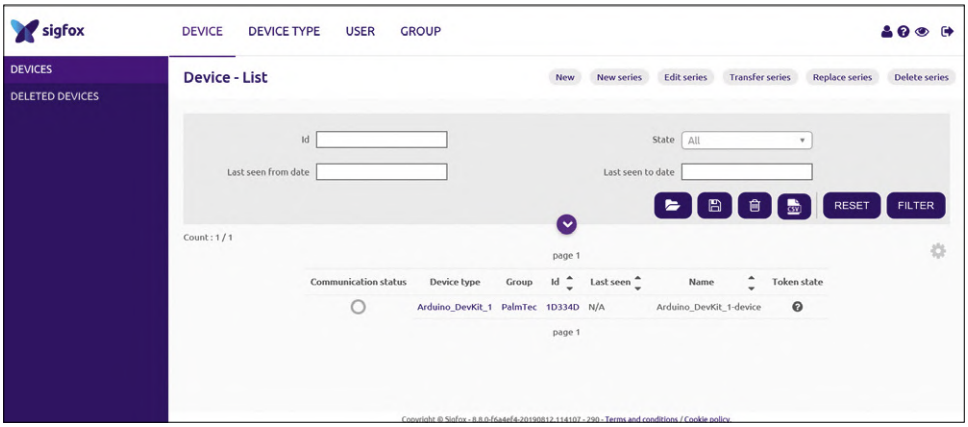
Om te beginnen kijken we hoe ons station er uitziet in de Sigfox-cloud. We melden ons op de gebruikelijke manier aan met email-adres en wachtwoord bij het Sigfox-backend [1] en komen dan op de startpagina van het Sigfox-portaal (**figuur 1**). Klik nu op de tab *Device*. U krijgt dan een overzicht van de Sigfox-apparaten die u hebt aangemeld en geactiveerd (**figuur 2**). Als het goed is ziet u dus (alleen) het station MKR FOX 1200. De lijst bevat de volgende gegevens:

- **Group**: de groepsnaam is door Sigfox automatisch aangemaakt op basis van uw gegevens en kan niet meer worden veranderd. Al uw Sigfox-apparaten bevinden zich in deze groep. Als u op de groepsnaam klikt, krijgt u gedetailleerde informatie over de groep (die ook niet meer kan worden veranderd).
- **Device type**: binnen de groep kunnen stations van hetzelfde type (met dezelfde opbouw en functionaliteit, dus identieke stations) worden gesorteerd en ingedeeld, bijvoorbeeld stations voor temperatuurmeting, stations



Figuur 1. Het Sigfox-portaal.

voor analoge signalen en stations voor binaire signalen. Wij hebben natuurlijk maar één type, de *Arduino_DevKit_1*. U kunt die naam later nog veranderen en aanpassen aan uw eigen wensen.



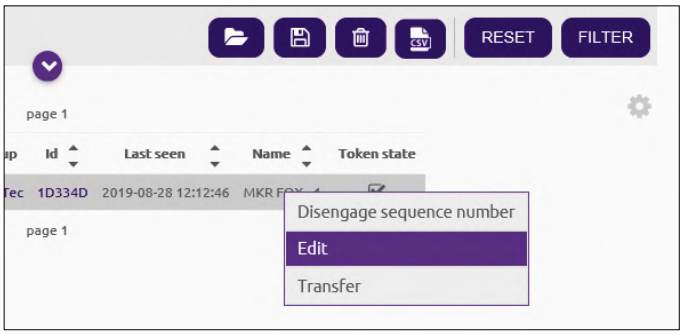
Figuur 2. De aangemelde Sigfox-devices.

- **Id** (= Id-nummer): hier verschijnt het unieke Id-nummer van het station.

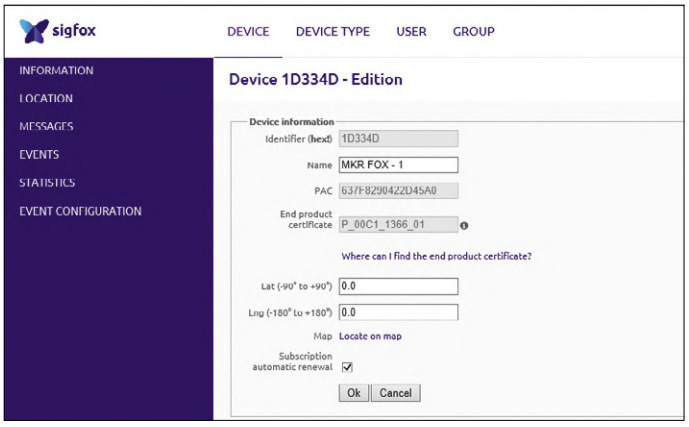
- **Last seen:** het backend vult hier de datum en tijd in waarop het station zich voor het laatst heeft gemeld, dus wanneer het laatste telegram van het station is ontvangen. Als er N/A staat, betekent dat, dat er nog geen enkel telegram van het station is ontvangen.

- **Name:** de naam van het station.

- **Token state: e**en token is een vergunning voor een station om het Sigfox-netwerk te gebruiken. Als u een overeenkomst aangaat met Sigfox (en betaalt voor het gebruik van het netwerk), krijgt u een bepaald aantal tokens tot uw beschikking. Als een station actief moet worden, kent u een token toe aan dat station. Daarna kan het gebruik maken van het netwerk. Bij de aankoop van een MKR FOX 1200-board krijgt u van Sigfox gratis een token voor één jaar. Het MKR FOX 1200-board kan dus een jaar lang in het netwerk meedoen (elk volgend MKR FOX 1200-board krijgt van Sigfox weer een eigen, uniek token). De *Token state* geeft aan, of het station het token al heeft gebruikt, dus of het al ooit data naar het Sigfox-netwerk heeft gestuurd. Bij ons is dat nog niet het geval, en daarom staat er een vraagteken, dat geeft aan dat het station nog geen gebruik heeft gemaakt van zijn recht om te zenden. Zodra het station een eerste telegram heeft verstuurd, verschijnt hier een vinkje. Pas dan is het token definitief toegewezen.



Figuur 3. Het pull-down-menu van het *Name*-veld.



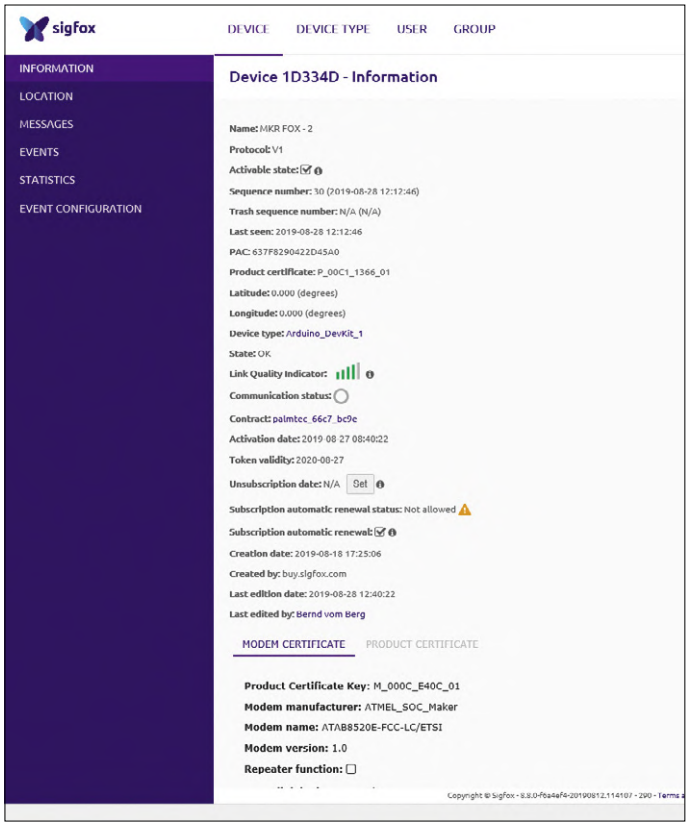
Figuur 4. Het Editor-venster van het device.

Klik nu met de muis op het veld *Name*. Er verschijnt dan een klein pull-down menu (**figuur 3**). Kies daar voor *Edit*. U komt dan in een *Edit*-venster (**figuur 4**), waar u de instellingen voor het *Device* kunt bewerken. Hier kunt u (bijvoorbeeld) de naam van het Sigfox-station veranderen in "MKR FOX – 1". Laat alle andere velden onveranderd en klik op *Ok* om de verandering vast te leggen. U komt dan in het venster *Device Information* (**figuur 5**), waarin alle gegevens over de Sigfox-module worden weergegeven. Hier geldt: alleen kijken, niet aankomen (veranderen). Als u linksboven op de tab *Device* klikt, komt u weer terug in het Device-overzicht, waar de nieuwe naam nu te zien moet zijn. Tenslotte kunt u het Sigfox-backend weer sluiten.

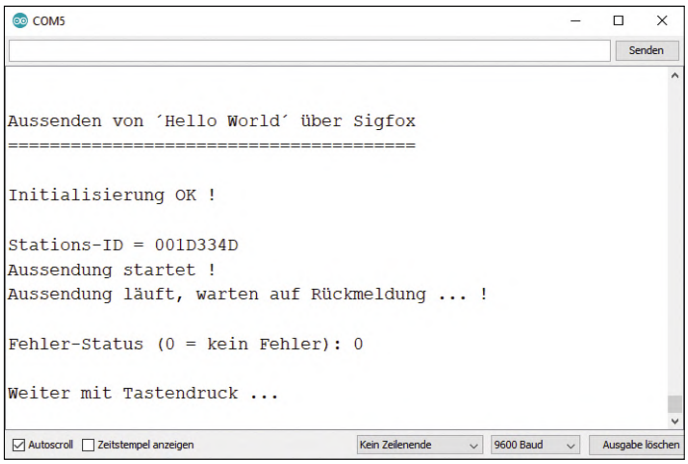
Hello World!

Om de activering definitief af te ronden, zodat we in het Sigfox-netwerk kunnen werken, moeten we eerst nog een geldig bericht versturen. Vanaf dat moment gaat het gratis Sigfox-lidmaatschap (*Subscription*) voor één jaar in. U hebt recht op 140 upload-berichten en 4 download-berichten per dag.
Als eerste bericht willen we de bekende standaardtekst "Hello World!" naar het Sigfox-backend sturen. Deze string is precies twaalf ASCII-tekens (= bytes) lang en past dus exact in een Sigfox-payload. In het vorige deel van deze serie hebben we de Sigfox-functies voor het zenden al beschreven, dus de kern van de zendroutine in **listing 1** zou gemakkelijk te begrijpen moeten zijn.

- `SigFox.begin`: dit initialiseert de Sigfox-library en het Sigfox-modem; als dat niet lukt wordt een foutmelding gegeven.



Figuur 5. Het informatievenster van het device.



Figuur 6. Zo ziet de test-string eruit.

Listing 1. Verzenden van de tekst “Hello World!”.

```
// enable Sigfox modem and check for errors
if (!SigFox.begin())        // error occurred?
{
  Serial.println("Error in Sigfox module!");

  while (1);  // after error, drop into infinite loop
}

// enable Sigfox modem debug mode
SigFox.debug();

// now we will send the string "Hello World!"

// prepare to transmit a packet
SigFox.beginPacket();

// assemble string (sequence of characters) into a Sigfox message
SigFox.print("Hello World!");

// final step in packet assembly and transmission; check for errors
// return code in variable ret
int ret = SigFox.endPacket();

Serial.print("\nError status (0 = no error): ");
Serial.println(ret);

// close Sigfox library and shut module down
SigFox.end();
// close Sigfox library and shut module down
SigFox.end();
```

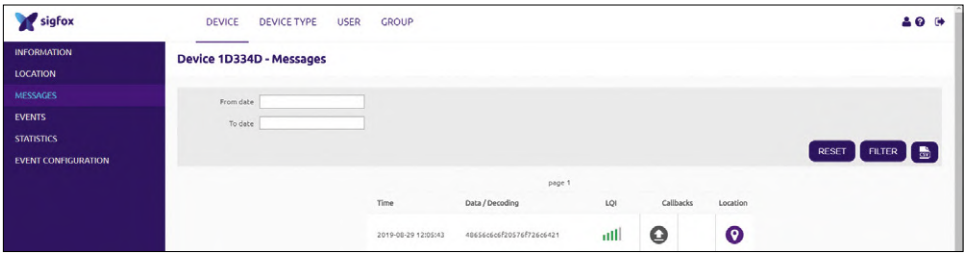
- `SigFox.debug`: activeert de debug-modus. De energiespaarfuncties van de microcontroller en het modem worden uitgeschakeld en de gele LED van het MKR FOX 1200-board knippert ten teken dat er data-overdracht plaatsvindt.
- `SigFox.beginPacket`: deze functie leidt de zendoperatie van het Sigfox-pakket (= Sigfox-payload) in.
- `SigFox.print`: de gegeven waarde of string wordt in de payload geladen en verzonden. Ook numerieke waarden worden als ASCII-strings verzonden. Bij binaire data (‘echte’ getallen, geen ASCII-weergave) wordt de functie SigFox.write gebruikt (later meer daarover).
- `int ret = SigFox.endPacket`: sluit het verzenden van het Sigfox-pakket (Sigfox-payload) af en geeft de foutstatus terug: 0 (geen fout), 1 (fout).
- `SigFox.end`: sluit de Sigfox-library en het Sigfox-modem.

Zo eenvoudig kan de complexe datacommunicatie worden uitgevoerd, als we een krachtige bibliotheek tot onze beschikking hebben! Laad de Sigfox-sketch inp het MKR FOX 1200-board, open de seriële monitor en start de sketch. Nu verschijnt in het monitorvenster het hoofdmenu van de sketch. Kies nu menupunt 2, verzenden van “*Hello World!*”. Dan worden de beschreven instructies uitgevoerd, waarbij in het monitorvenster wat toestandsinformatie wordt weergegeven. Intussen

laat de knipperende gele LED zien, dat er data wordt verzonden. Het hele proces moet zonder foutmeldingen worden afgewikkeld (**figuur 6**).

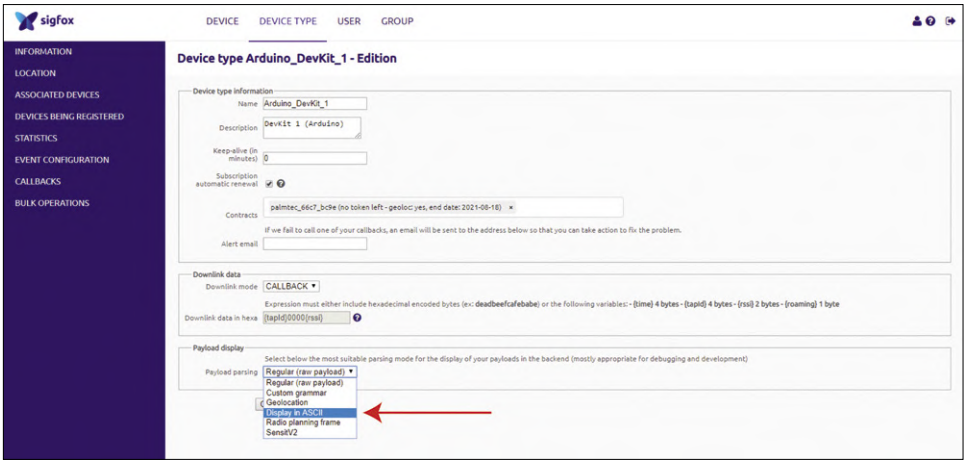
Terug in het backend

Als u in een gebied woont waar Sigfox al actief is, komt de data al heel snel aan in het Sigfox-backend. U kunt dat controleren door naar het Sigfox-backend te gaan, in te loggen en op de tab *Device* te klikken. Klik in de lijst van actieve Sigfox-devices (precies) op het veld *Id* van het station, zodat de informatiepagina van het device verschijnt. Druk dan links op *Messages*.
In het volgende venster (**figuur 7**) ziet u alle telegrammen, die het Sigfox-backend van het station heeft ontvangen. Het veld *Time* spreekt voor zich, in het veld *LQI* (Link Quality Indicator) wordt de veldsterkte weergegeven, waarmee het Sigfox-telegram is ontvangen door de basisstations. Als u met de muis naar dit veld toe gaat, ziet u een beschrijving van de ontvangstkwaliteit en wordt aangegeven via welke lokale provider het telegram is ontvangen. De provider hoeft niet per se Sigfox te heten, het kan (bijvoorbeeld in het buitenland) ook een samenwerkingspartner zijn. Als u op *Location* klikt, krijgt u een kaart te zien, waarop (ruwweg) is aangegeven, waar het Sigfox-station zich ongeveer bevindt.



Figuur 7. Het Message-venster van het device.

Figuur 8. Het informatieveld van het device-type.

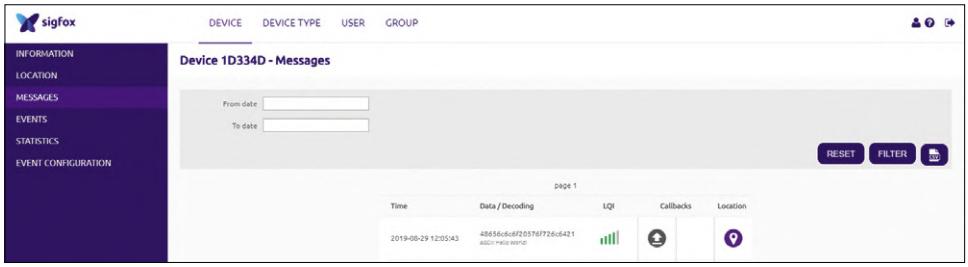


Figuur 9. Het edit-venster van het device-type.

En dan komen we bij de belangrijke informatie in het veld *Data/Decoding*, waar de inhoud van de payload wordt weergegeven, als ruwe data en in gedecodeerde vorm. Het backend weet nog niet, hoe het de ruwe data moet interpreteren, dat moeten we nog instellen. Laten we het data-veld eens goed bekijken:

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21
H  e  l  l  o     W  o  r  l  d  !
```

We zien dat de string bestaat uit de ASCII-codes van de tekst “Hello World!” (met ASCII 20 voor de spatie). De manier van decoderen is in te stellen vanuit het venster *Device-List* (figuur 2). Klik precies op de naam *Arduino_DevKit_1* onder *Device type*, dan komt u bij het *Information*-veld voor dit *Device type* (**figuur 8**), en door op *Edit* te klikken bij het bijbehorende *Edit*-venster (**figuur 9**). Verander (alleen) in het veld *Payload display* de *Payload parsing* in *Display in ASCII* en bevestig de verandering. Terug in het venster *Messages* ziet u nu bij het station een extra regel met de weergave van de Payload-inhoud in ASCII-tekens (**figuur 10**).
Voor het overdragen van numerieke waarden in de payload zijn twee voorbereidende stappen nodig. Om te beginnen moet u vastleggen welke datatypen er in de twaalf byte grote payload moeten komen, want ieder datatype heeft een verschillend aantal bytes nodig. De belangrijkste datatypen zijn te zien in **tabel 1**. De over te dragen datatypen kunnen in de payload worden gerangschikt zoals in **tabel 2**. Van de twaalf beschikbare bytes zijn er hier tien gebruikt.
Ten tweede moet een speciaal datatype worden gedefinieerd, dat al deze data samenvat in één record, dat gemakkelijk kan worden opgeslagen en gemanipuleerd. Dat ziet er in ons voorbeeld uit als in **listing 2**. Om deze constructie precies te begrijpen is wat kennis van C/C++ nodig, maar het is gemakkelijk uit te leggen.



Figuur 10. Decodering van de payload in ASCII-tekens.

Listing 2. Opbouw van het Sigfox-datatype Sigi_Dat_1.

```
// structure and data type 1 for our example:
// transmission of fixed numeric values
 typedef struct __attribute__ ((packed)) sigfox_message {
  unsigned char  value_1; // first value in payload: 1 byte long
  unsigned int   value_2; // second value in payload: 4 bytes long
  float          value_3; // third value in payload: 4 bytes long
  unsigned char  value_4; // fourth value in payload: 1 byte long
} Sigi_Dat_1;
```

Listing 3. Zenden van vaste getalwaarden.

```
 // definition of four fixed numeric values
  unsigned char   AIN_1 = 0x1f ;          // first value, 1 byte long
  unsigned int    pressure = 0x12345678; // second value, 4 bytes long
  float           temp = 1233.56;        // third value, 4 bytes long
  unsigned char   AIN_2 = 0x55;          // fourth value, 1 byte long

  // enable Sigfox modem and check for errors
  if (!SigFox.begin())        // error occurred?
  {
    Serial.println("Error initializing Sigfox module. RESET to continue");
    while (1);  // after error, drop into infinite loop
  }
  else
  {
      Serial.println("Sigfox modem OK\n");
  }

  // enable debug LED and disable power-saving modes
  SigFox.debug();

  // deal with all pending interrupts
  SigFox.status();
  delay(1);

  // now we write the current values that are to be transmitted into
  // the SF_send structure variable
  // this process assembles the payload contents
  SF_send.value_1 = AIN_1;        // unsigned char value:  1 byte
  SF_send.value_2 = pressure;     // unsigned int value:   4 bytes
  SF_send.value_3 = temp;         // float value:          4 bytes
  SF_send.value_4 = AIN_2;        // unsigned char value:  1 byte
```

Vervolg op de volgende pagina ...

Tabel 1. Aantal bytes van verschillende datatypen.

Datatype	Benodigde bytes
char / unsigned char	1
int8_t / uint8_t	1
int16_t / uint16_t	2
int / unsigned int	4
int64_t / uint64_t	8
float	4

Met de instructie `struct ..... sigfox_message` en de instructies die er op volgen tussen accolades stellen we een zogenaamde structuur samen, die naam `sigfox_message` krijgt. zo'n *struct* is niets anders dan een groep data die bij elkaar hoort en als groep een gezamenlijke (hoofd-) naam krijgt. U kent dat wel van een array, maar in een array staat altijd alleen maar data van één type. In een struct kunnen we data van verschillende typen samenvoegen, in dit geval twee `unsigned char`-, één `unsigned int`- en één `float`-waarde. De datatypen van de elementen in de struct moeten exact overeenkomen met de indeling van de payload en ook in dezelfde volgorde staan.
We kunnen gemakkelijk refereren aan een bepaald element in de struct:
`naam van de struct`**.**`naam van het element`

We kunnen dus bijvoorbeeld met

```
sigfox_message.waarde_3 =25.78;
```

het interne float-element `waarde_3` op de waarde `25.78` instellen.

En nu gaan we nog een stap verder en genereren uit deze struct een heel

Tabel 2: Een mogelijke indeling van verschillende datatypen in de payload.

Waarde in de Payload	Datatype	Bytes
1	unsigned char	1
2	unsigned int	4
3	float	4
4	unsigned char	1

nieuw datatype. U kent natuurlijk de standaard datatypen in C, zoals `unsigned char`, `int` of `float`.

Met de extra instructie `typedef` aan het begin van de struct-declaratie maken we een nieuw datatype, das precies de opbouw heeft van de struct die erachter staat. Dit nieuwe datatype moet natuurlijk ook een eigen naam hebben, en die geven we aan het eind van de declaratie, in dit geval `Sigi_Dat_1`. De code `__attribute__ ((packed)` zorgt ervoor dat er geen extra bytes voor opvulling (padding bytes) in de struct komen te staan, zodat we zeker weten dat alleen de gespecificeerde bytes in de payload terechtkomen.

Deze hele struct wordt nu meteen bij het begin van de sketch gedefinieerd als een globale variabele, zodat we hem overal in de sketch kunnen gebruiken. Als we nu met dit datatype willen werken, moeten we natuurlijk variabelen van dat type defini-

... vervolg van de vorige pagina:

```
// next we use the Sigfox library to transmit the data we have
// assembled in the structure variable SF_send, which will become the
// payload contents

// prepare to transmit a packet
SigFox.beginPacket();

// pass structure variable to the Sigfox library
SigFox.write((char*)&SF_send, sizeof(SF_send));

// error check: if endPacket() returns 1, report an error
int ret = SigFox.endPacket();
if (ret > 0)
{
  Serial.println("Error: transmission failed. RESET to continue");
  while(1);     // infinite loop
}
else
{
  Serial.println("Sigfox transmission OK");
}

// close Sigfox library and shut module down
SigFox.end();
```

Tabel 3. Binnenkomende payload.			
1f	78563412	ec319a44	55
(A)	(B)	(C)	(D)

eren. Dat gebeurt precies zoals we dat gewend zijn, met

```
datatype variabelenaam;
```

De regel `Sigi_Dat_1 SF_send;` maakt een variabele aan met de naam `SF_send`, die exact de vastgelegde structuur heeft (het is handig om er meteen een globale variabele van te maken). We kunnen aan de individuele waarden in deze variabele refereren met bijvoorbeeld `SF_send.waarde_1 = 12;`.

Listing 4. Een nieuw datatype met de naam "Sigi_Dat_2" voor het verzenden van meetwaarden.

```
// structure and data type 2 for our example:
// transmission of sensor readings
  typedef struct __attribute__ ((packed)) sigfox_message_2 {
    float      value_1;     // first value in payload:   4 bytes long
    float      value_2;     // second value in payload:  4 bytes long
    uint16_t   value_3;     // third value in payload:   2 bytes long

    // this structure therefore occupies a total of ten bytes

} Sigi_Dat_2;
```

Listing 5. Invullen van de meetwaarden in de struct-variabele.

```
    // now we write the current values that are to be transmitted into
    // the SF_send_mw structure variable
    // this process assembles the payload contents
    SF_send_mw.value_1 =   temp;        // float value:        4 bytes
    SF_send_mw.value_2 =   pressure;    // float value:        4 bytes
    SF_send_mw.value_3 =   LDR;         // uint16_t value:     2 bytes
```

Zenden

Om te zien hoe de numerieke waarden verpakt worden in de Sigfox-payload, schrijven we een functie die vaste getallen via het Sigfox-netwerk verstuurt (**listing 3**). Deze functie wordt aangeroepen met menupunt 3 in de demo-sketch. Er valt nu niet veel meer uit te leggen, behalve dat we nu de functie `SigFox.write` gebruiken, omdat we geen tekst, maar numerieke waarden willen verzenden. Als alles naar behoren werkt, verschijnt de payload in het backend zoals in **tabel 3**. Tenslotte ontwikkelen we nog een functie, waarmee echte meetwaarden worden verstuurd (menupunt 4 in de voorbeeld-sketch), om precies te zijn de gemeten temperatuurwaarde van de BMP280-sensor (float, 4 bytes), de gemeten luchtdruk van de BMP280-sensor (float, 4 bytes) en de geregistreerde LDR-meetwaarde (zet daartoe jumper JP5 op het moederboard op positie 1-2) via de analoge ingang A2 (uint16_t, 2 bytes). Daarmee worden tien van de twaalf bytes in de payload gebruikt.

Voor deze gegevens definiëren we nog een Sigfox-datatype (**listing 4**) en een variabele met de naam `SF_send_mw`:

```
// Variable zum Datentyp ´Sigi_Dat_2´:
Sigi_Dat_2 SF_send_mw;
```

In een nieuwe functie in de sketch worden drie meetwaarden gelezen, in de struct-variabele geschreven en verzonden (**listing 5**). Extra print-statements via de seriële monitor en op het ePaper-display laten zowel de actuele meetgegevens zien als het verloop van de Sigfox-communicatie. Meer uitleg is te vinden in de goed gedocumenteerde programmalisting van de sketch.

Vooruitblik

Tot zo ver onze kleine inleiding in het 0G-IoT-netwerk Sigfox, die we hebben gedaan met behulp van het Arduino Maker-board MKR FOX 1200. Onze programma's en functies bieden u een goed uitgangspunt voor het realiseren van eigen toepassingen met het Sigfox-netwerk.

Wat nu nog ontbreekt, is het visualiseren van de data aan de kant van de gebruiker, dus op een gebruikers-PC/laptop/smartphone met een vrij configureerbaar dashboard. Dat is het thema van het volgende en laatste deel van deze serie.

(190281-C-04)

Weblinks

[1] Inloggen op het backend: https://backend.sigfox.com/auth/login