

#01 Blink

Was in anderen Programmiersprachen das “Hello World” Beispiel ist, ist für die Hardware-Welt das Blink Beispiel. Das einfachste Zeichen, dass die Programmierung funktioniert ist es nämlich, eine LED an- und auszuschalten.

Nachdem der Photon konfiguriert ist, loggt man sich zum flashen unter build.particle.io ein.

Datei

01-BLINK

Material

- Particle Photon

Neue Code-Elemente

- setup()
- loop()
- pinMode()
- digitalWrite()
- delay()

```
int ledPin = D7; // In der Variablen led_pin wird der Name des Pins gespeichert, an dem die
                // interne LED hängt: D7.

// Die Setup Funktion wird einmal aufgerufen, wenn der Photon startet. Also wenn er wieder mit
// Strom versorgt wird oder der Reset Button gedrückt wurde.
// Wir benutzen die Funktion deshalb hauptsächlich für Initialisierungen.
void setup() {
    pinMode(ledPin, OUTPUT); // Wir legen fest, dass wir den Pin als Output benutzen wollen.
}

// Die loop Funktion wird ständig in einer Schleife ausgeführt.
// Hier kommt deshalb aller Code hinein, der wiederholt ausgeführt werden soll.
void loop() {
    digitalWrite(ledPin, HIGH); // Wir schalten die LED an, indem wir den Pin auf HIGH setzen.
    delay(1000);                // Warten für 1000 Millisekunden, also eine Sekunde.
    digitalWrite(ledPin, LOW);  // Die LED wird wieder ausgeschaltet.
    delay(1000);                // Bevor der loop wieder von vorne beginnt und die LED wieder
                                // angeschaltet wird, warten wir nochmal 1 Sekunde.
}
```

#02 Button

Dieses Beispiel zeigt, wie ein Button an den Photon angeschlossen und ausgelesen wird. Das geht mit dem Grove System ganz einfach. Dafür wird zunächst der Photon in das Grove Base Shield eingesetzt und der Button mit dem Anschluss D2 verbunden.

Mit dem folgenden Code wird die interne LED für eine Sekunde angeschaltet, wenn der Button gedrückt wird. Darüber hinaus wird mit `Particle.publish()` eine Nachricht in die Cloud Console geschrieben.

Filename

02-BUTTON

Material

- Particle Photon
- Grove Base Shield
- Button an D2

Neue Code-Elemente


- `if()` .. `else()`
- `Particle.publish()`

```
int ledPin = D7;
int buttonPin = D2;           // Der Button wird an den Anschluss D2 angeschlossen.

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT); // Der Pin, an den der Button angeschlossen ist, soll als Input
                             // verwendet werden.
}

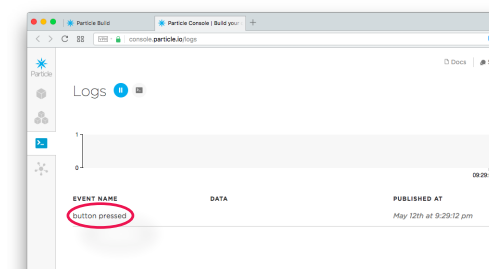
void loop() {
  // Lese den Status des Buttons aus und schreibe den Zustand in eine Variable
  bool buttonIsPressed = digitalRead(buttonPin);

  if(buttonIsPressed) {
    digitalWrite(ledPin, HIGH); // LED anschalten
    // Sende eine Nachricht an die Cloud Console, dass der Button gedrückt wurde.
    Particle.publish("awesome-button-pressed");
    delay(1000); // warte eine sekunde, bevor wir weitermachen
  } else {
    digitalWrite(ledPin, LOW); // LED ausschalten, wenn der Button nicht gedrückt ist.
  }
}
```

Um die Nachricht in der Cloud Console zu sehen, klickt man im Menü auf der linken Seite auf Console 



Im sich neu öffnenden Tab wird kurze Zeit nach dem ersten Tastendruck eine entsprechende Nachricht erscheinen.



#03 Ein Event abonnieren

Im vorherigen Beispiel wurde mit `Particle.publish()` eine Nachricht über ein Event an die Cloud gesendet. Dieses kann dann aber nicht nur in der Console angesehen werden, sondern auch von anderen Photons abonniert werden, die darauf reagieren können. In diesem Beispiel wird die LED auf dem anderen Photon an- und ausgeschaltet.

Um das auszuprobieren, sollte sich nun jeweils zwei Personen zusammentun und auf einen ihrer Photons den folgenden Code flashen. Der andere behält den Code `02-BUTTON` aus dem vorherigen Beispiel.

Vorher sollte der Name des Events von `awesome-button-pressed` noch auf beiden Photons in einen einmaligen Namen geändert werden.

Filename

03-SUBSCRIBE-EVENT

Material

- Particle Photon
- Grove Base Shield

Neue Code-Elemente

- `Particle.subscribe()`

```
int ledPin = D7;

void setup() {
  pinMode(ledPin, OUTPUT);
  Particle.subscribe("awesome-button-pressed", eventHandler); // das Event abonnieren
}

void loop() {
  // hier gibt es nichts zu tun
}

// diese Funktion wird aufgerufen, wenn der andere Photon das Event auslöst
void eventHandler(const char *event, const char *data){
  digitalWrite(ledPin, HIGH); // LED anschalten
  delay(1000);                // kurz warten
  digitalWrite(ledPin, LOW);  // LED ausschalten
}
```

#04 Potentiometer

Im Unterschied zum Button, der nur zwei Zustände kennt (gedrückt oder nicht) und damit "digital" ist, kann ein Drehregler – in der Fachsprache Potentiometer oder kurz Poti genannt – ganz viele Zustände annehmen. Die aktuelle Position wird deshalb analog ausgelesen mit der Funktion `analogRead()`. Diese gibt eine Zahl zwischen 0 und 4095* zurück.

Der aktuelle Wert wird in der Cloud veröffentlicht, indem wir in `setup()` die Funktion `Particle.variable()` verwenden.

Darüber hinaus wird die Funktion `analogWrite()` verwendet, um eine LED entsprechend der Stellung des Potis zu dimmen.

Filename

04-POTENTIOMETER

Material

- Particle Photon auf Grove Base Shield
- Rotary Angle Sensor an A0
- LED Socket Kit an D2

Neue Code-Elemente

- `Particle.variable()`
- `analogRead()`
- `analogWrite()`


```
int externalLedPin = D2; // Die externe LED hängt an D2
int potiPin = A0;        // Das Potentiometer ist am analogen Input A0 angeschlossen
int potiValue = 0;       // Variable, die den aktuellen Wert des Potentiometers enthält

void setup() {
  pinMode(externalLedPin, OUTPUT);
  pinMode(potiPin, INPUT);

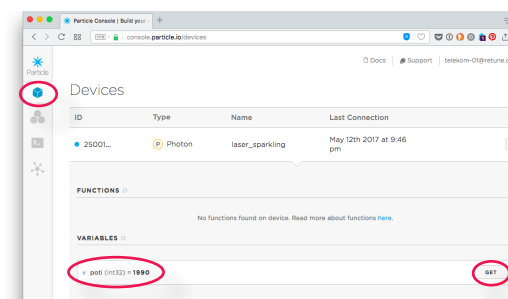
  Particle.variable("poti", potiValue); // Veröffentlichung der Variablen unter dem Namen
                                        // "poti" in der Cloud
}

void loop() {
  // Lese die Position des Potentiometers analog aus und speichere die Zahl zwischen 0 und 4095
  // in der Variablen. Das Updaten in der cloud wird automatisch erledigt.
  potiValue = analogRead(potiPin);

  // Dimme die LED entsprechend des Poti-Werts. Wir müssen den Wert durch 16 teilen, weil die
  // Funktion eine Zahl zwischen 0 und 255 erwartet.
  analogWrite(externalLedPin, potiValue / 16);
}
```

Um den aktuellen Wert sehen zu können, öffnet man wieder die Console  und klickt anschließend auf My Devices .

Nach einem Klick auf das Device steht unter VARIABLES die veröffentlichte Variable „poti“ und kann mit einem Klick auf GET aktualisiert werden.



*) Nerd Info: Es ist ein Analog-Digital-Wandler (ADC) mit 12bit Auflösung im Photon verbaut. Deshalb gibt es $2^{12} = 4096$ mögliche Werte.

#05 Funktionen veröffentlichen

Nachdem wir im letzten Beispiel gesehen haben, wie man Variablen online zugänglich macht, wollen wir nun den umgekehrten Weg gehen und schauen, wie man aus der Cloud heraus Ereignisse auf dem Photon triggern kann. Hierfür verwenden wir die `Particle.function()` um eine Funktion über die API zugänglich zu machen.

Filename

05-EXPOSE-FUNCTION

Material

- Particle Photon auf Grove Base Shield
- 4-Digit Display an I2C_1
- Vibration Motor an D2

Neue Code-Elemente

- `Particle.function()`
- TM1637Display Library

```
#include <TM1637Display.h> // Bibliothek für die Ansteuerung des Displays einbinden
int vibrationPin = D2;      // Der Vibrationsmotor ist an D2 angeschlossen
int counter = 0;           // In dieser Variablen wird gezählt, wie oft die Funktion schon
                           // aufgerufen wurde.

TM1637Display display(D1,D0); // Das Display vom Typ TM1637Display wird initialisiert. Es ist mit
                           // den Pins D1 und D0 verbunden.

void setup() {
    pinMode(vibrationPin, OUTPUT);

    // Die Funktion triggerFunction unter dem Namen „trigger“ veröffentlichen
    Particle.function(„trigger“, triggerFunction);

    // Helligkeit des Displays auf Maximum setzen (15 bei diesem Modul)
    display.setBrightness(15);



    // Den aktuellen Counter Wert aufs Display schreiben. Der ist hier noch Null.
    display.showNumberDec(counter);
}

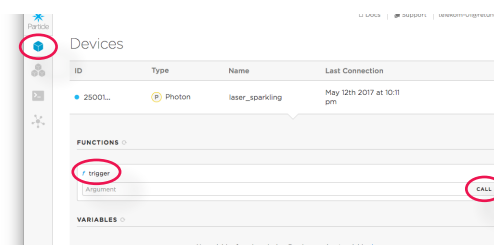
void loop() {
    // Weil wir alles eventbasiert in der triggerFunction erledigen, ist hier nichts zu tun.
}

// Diese Funktion kann über das Internet aufgerufen werden.
int triggerFunction(String command){
    // Den Vibrationsmotor kurz an- und wieder ausschalten.
    digitalWrite(vibrationPin, HIGH);
    delay(100);
    digitalWrite(vibrationPin, LOW);

    // Das Display aktualisieren
    counter++; // Den Counter um eins erhöhen.
    display.showNumberDec(counter); // Die Anzeige des Displays entsprechend aktualisieren.

    // Wir geben den aktuellen Counter-Wert zurück. Er wird damit auch in der Cloud sichtbar sein.
    return counter;
}
```

Die Funktion wird nun in der Console  unter My Devices  angezeigt und kann durch einen Klick auf CALL ausgeführt werden.



#06 Über eine eigene Website mit dem Photon interagieren

Auf veröffentlichte Funktionen und Variablen kann man aber natürlich nicht nur in der Console zugreifen, sondern diese auch über die Particle Cloud API in Webseiten oder auch Apps einbinden. Dafür gibt es eine Javascript Bibliothek und SDKs für Android, iOS und Windows.

Filename

06-WEBSITE

Material

- Particle Photon auf Grove Base Shield
- 4-Digit Display an I2C_1
- Vibration Motor an D2
- Light Sensor an A0

```
#include <TM1637Display.h> // Bibliothek für die Ansteuerung des Displays einbinden
int vibrationPin = D2;      // Der Vibrationsmotor ist an D2 angeschlossen
int lightSensorPin = A0;    // Lichtsensor an A0
int lightValue = 0;         // Variable für den aktuellen Helligkeitswert

TM1637Display display(D1,D0); // Das Display initialisieren

void setup() {
    pinMode(vibrationPin, OUTPUT);

    // Die Funktion triggerFunction unter dem Namen „trigger“ veröffentlichen.
    Particle.function(„trigger“, triggerFunction);

    // Die Variable für den Helligkeitswert unter dem Namen „light“ ebenfalls veröffentlichen.
    Particle.variable(„light“, lightValue);

    // Helligkeit des Displays auf Maximum setzen (15 bei diesem Modul)
    display.setBrightness(15);
}

void loop() {
    // Den aktuellen Messwert des Lichtsensors auslesen und auf das Display schreiben
    lightValue = analogRead(lightSensorPin);
    display.showNumberDec(lightValue);
}

// Diese Funktion kann über das Internet aufgerufen werden.
int triggerFunction(String command){
    // Den Vibrationsmotor kurz an- und wieder ausschalten.
    digitalWrite(vibrationPin, HIGH);
    delay(100);
    digitalWrite(vibrationPin, LOW);
}
```

Um den Zugriff auf die veröffentlichte Funktion „trigger“ als auch die veröffentlichte Variable „light“ in eine Website zu integrieren, verwenden wir für dieses Beispiel Javascript Bibliothek. Dazu wird der unten stehende Code in einen Texteditor kopiert und als HTML Datei abgespeichert.

```

<html>
<head>
  <title>myPhoton</title>
  <!-- Die Particle Javascript Bibliothek einbinden -->
  <script type="text/javascript" src="http://cdn.jsdelivr.net/particle-api-js/5/particle.min.js"></script>
</head>

<body>
  <!-- Eine sehr simple HTML Seite mit zwei Buttons -->
  <center>
    <span id="lightvalue">Lichtsensord noch nicht abgefragt</span>
    <br/>
    <button type="button" onclick="trigger()">Trigger</button>
    <button type="button" onclick="update()">Lichtsensord abfragen</button>
  </center>

  <!-- Hier kommt der javascript code -->
  <script>
    var token = 'ACCESS_TOKEN'; // Hier das Access Token eintragen. Ist unter Settings zu finden.
    var device = 'DEVICE_ID'; // Hier die Device ID eintragen. Ist unter Devices zu finden.
    var particle = new Particle(); // Die Bibliothek initialisieren.

    // Die trigger funktion
    function trigger(){
      particle.callFunction({ deviceId: device, name: 'trigger', auth: token });
    }

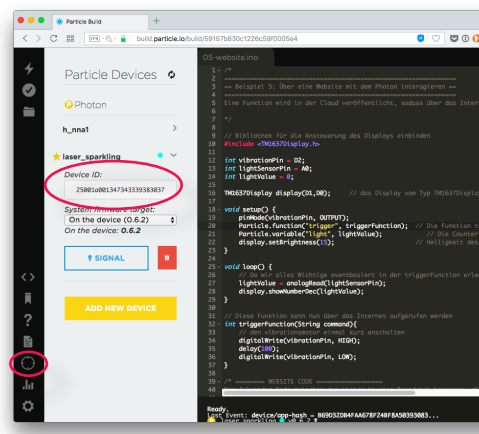
    // Diese Funktion updated den Helligkeitswert.
    function update() {
      particle.getVariable({ deviceId: device, name: 'light', auth: token }).then(function(data){
        // Wert in den HTML Code schreiben.
        document.getElementById("lightvalue").innerHTML = data.body.result;
      });
    }


    // Um sich das klicken auf den Button zum Auslesen des aktuellen wertes zu ersparen, kann
    // man die Funktion automatisch jede Sekunde aufrufen:
    // setInterval("update()", 1000);
  </script>
</body>
</html>

```

Bevor das funktioniert, muss im Code noch das Access Token und die Device ID eingetragen werden.

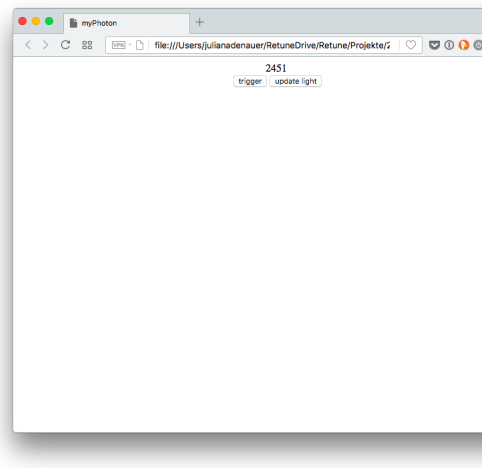
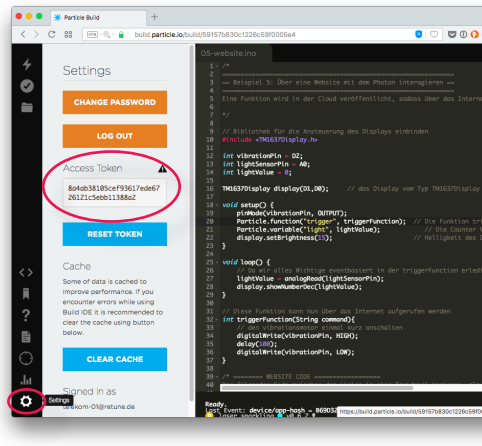
Die Device ID des verwendeten Photon ist unter Devices zu finden.



Zusätzlich benötigt man ein Access Token für den Account, das wie ein Passwort funktioniert und mithilfe dessen wir nachweisen können, dass wir berechtigt sind, auf diese Informationen zuzugreifen. Das Token ist unter Settings  zu finden.

HINWEIS: Das Access Token ist wie ein offengelegtes Passwort und sollte niemals im öffentlich zugänglichen Teil einer Website eingebunden werden!

Sind beide Informationen im Code ergänzt, kann man nach dem Abspeichern die HTML Datei im Browser öffnen. Nach einem Klick auf „Lichtsensor abfragen“ wird der aktuelle Messwert dargestellt und ein Klick auf „Trigger“ lässt den Vibrationsmotor am Photon kurz vibrieren.



#07 Visualisierung mit Ubidots

Häufig sammeln IoT Devices Daten über ihre Umgebung und Ereignisse. Es gibt einige Cloud-Services, die darauf spezialisiert sind, diese Daten zu sammeln, aufzubereiten und Aktionen abzuleiten. Einer dieser Services ist Ubidots. Wie die Anbindung an diesen Service funktioniert, zeigt dieses Beispiel.

Zunächst einmal wird wieder ein Access Token benötigt. Dieses Mal von Ubidots. Dazu legt man einen Account bei ubidots.com an und loggt sich anschließend ein. Anschließend klickt man oben rechts auf den Benutzernamen und wählt API Credentials.

Das benötigte Token findet sich dann auf der rechten Seite. Dieses wird an die entsprechende Stelle in den folgenden Code kopiert (Anführungszeichen nicht vergessen!).

Filename

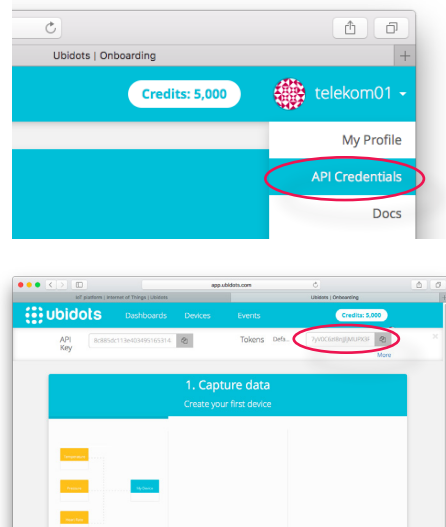
07-UBIDOTS

Material

- Particle Photon auf Grove Base Shield
- Temperature Sensor an A0

Neue Code-Elemente

- Ubidots und math Library



```
#include <Ubidots.h> // Die Ubidots Bibliothek einbinden
#include <math.h>     // Standard-Mathematik Bibliothek einbinden

Ubidots ubidots („UBIDOTS_TOKEN“); // hier muss das Ubidots Access Token eingefügt werden

int temperaturePin = A0; // Der Temperatursensor ist an A0 angeschlossen

void setup() {
    pinMode(temperaturePin, INPUT);
}

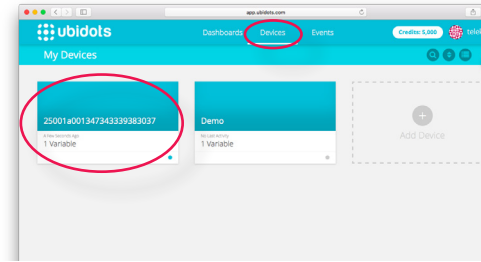
void loop() {
    // Temperatursensor analog auslesen
    int a=analogRead(temperaturePin);

    // Ein bisschen magischer Code, der den ausgelesenen Wert in eine Temperatur wandelt
    float widerstand = (4096.0/a-1.0);
    float temperature = 1.0/(log(widerstand)/4275+1/298.15)-273.15;

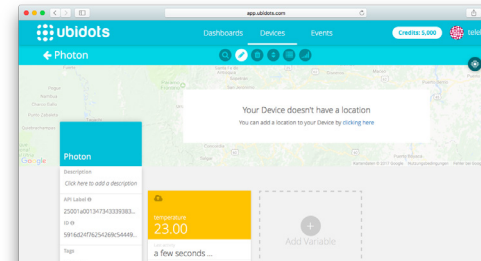
    // Die berechnete Temperatur nun an Ubidots senden
    ubidots.add („temperature“, temperature); // zum Paket hinzufügen
    ubidots.sendAll();                       // und wegschicken

    // Anschließend 5 Sekunden warten
    delay(5000);
}
```

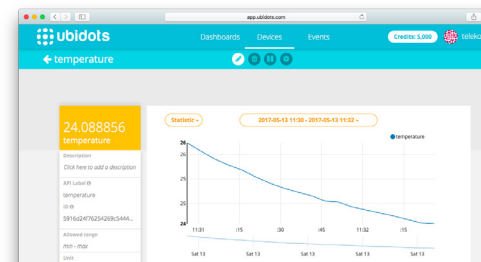
Sobald der Code geflashed ist, erscheint in Ubidots unter Devices ein neues Device mit kryptischem Namen.



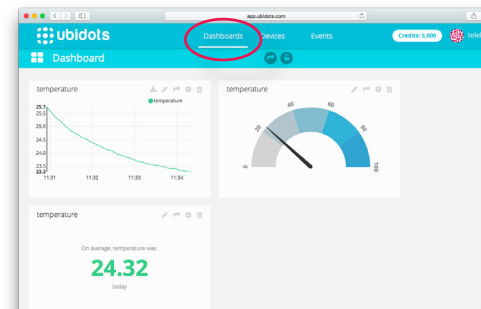
Klickt man darauf, bekommt man eine Übersicht über das Device und die registrierten Variablen. Das ist in diesem Fall nur die Temperatur. Nach einem Doppelklick auf den Device Namen lässt sich der Name in etwas weniger kryptisches ändern.



Nach einem Klick auf die Variable wird deren zeitlicher Verlauf gezeigt. Darüber hinaus können auch Mittel-, Max- und Min-Werte in einem wählbaren Zeitfenster angezeigt werden.



Unter Dashboard können aus Variablen benutzerdefinierte Widgets generiert werden.



Und schließlich kann man unter Events beispielsweise definieren, dass eine E-Mail verschickt wird, wenn eine gewisse Temperatur überschritten wird.

