Mit dem Arduino JSON abrufen & dekodieren mit (ArduinoJson)

Wenn du mit deinem ESP8266 oder ESP32* oder auch mit dem Arduino im Internet bist, dann sicher nicht ohne Grund. Vielleicht möchtest du Daten von einer API abrufen und in deinem Projekt weiterverwenden. **In diesem Tutorial lernst du, wie du mit dem Arduino JSON lädst und mithilfe der Bibliothek ArduinoJson dekodierst (oder parst).**

Hier verwenden wir einen ESP8266* – du kannst mit ein paar Anpassungen im Code aber auch jeden anderen Microcontroller verwenden, mit dem du ins Internet kommst. In diesem Tutorial auf pollux labs erfährst du, wie du deinen ESP8266 mit dem Internet verbindest.

Anschließend wirst du eine API kontaktieren und dir JSON-Daten herunterladen, die die aktuelle Anzahl von Menschen im Weltraum enthalten. Diese Daten wirst du lokal dekodieren (oder parsen) und in deinem Seriellen Monitor anzeigen.

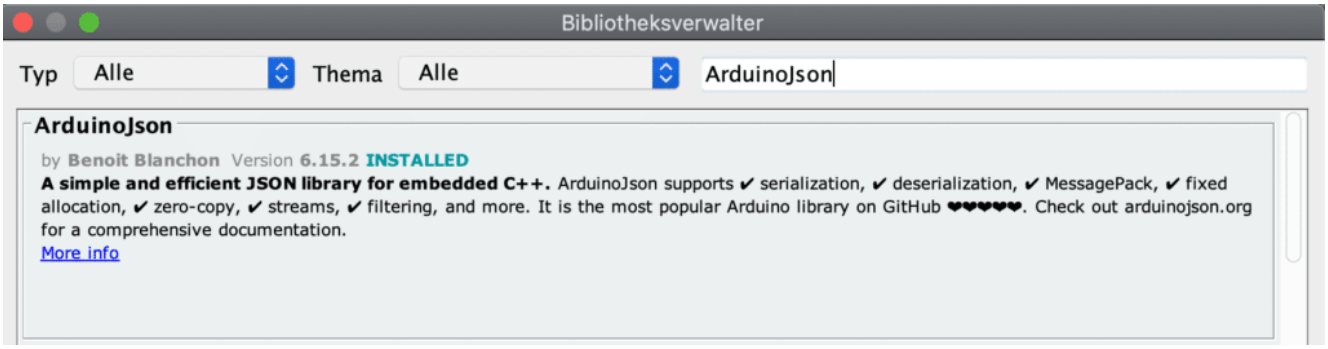
Die Bibliothek ArduinoJson

Bevor du loslegen kannst, benötigst du die aktuelle Version der Bibliothek ArduinoJson. Hierbei handelt es sich um eine wirklich praktische Erweiterung, die dir die meiste Arbeit mit

JSON abnimmt.

> *Hinweis: Dieses Tutorial bezieht sich auf die Bibliothek ArduinoJson bis zur Version 6.17 – in unserem kostenlosen ESP8266 Online-Kurs erklären wir, wie die neueste Version funktioniert.*

Öffne also deinen Bibliotheksverwalter in der Arduino IDE und suche dort nach **ArduinoJson**. Installiere dir die neueste Version.



Binde die Bibliothek nun ganz oben in deinem Sketch ein:

```
#include <ArduinoJson.h>
```

Der API Call

Um herauszufinden, wie viele Astronauten gerade im Weltall sind, fragen wir eine API von open-notify.org ab – und zwar unter folgender URL:

```
http://api.open-notify.org/astros.json
```

Wenn du diese URL kopierst und in deinem Browser öffnest, siehst du bereits die Rohdaten im JSON-Format. Recht am Anfang findest du **„number“** und dahinter die aktuelle Anzahl Astronauten im Weltraum. Heute – am 2. August 2020 – sind das 5.

Um diese Zeichenkette auf deinen ESP8266 zu bekommen, benötigst du folgenden Code:

```
HTTPClient http; //Instanz von HTTPClient starten
http.begin("http://api.open-notify.org/astros.json");  //URL für die Abfrage
int httpCode = http.GET(); //Antwort des Servers abrufen
    if (httpCode == 200) {
       String payload = http.getString(); //Daten in eine Variable speichern
      }
```

Jetzt befinden sich die Rohdaten, die du von der API geladen hast, in der Variable **payload.** Schön und gut, aber um diese Daten weiterverarbeiten zu können – z.B. die aktuelle Anzahl Astronauten auf einem Display anzuzeigen – musst du sie zunächst dekodieren. Und hier kommt wieder die Bibliothek ArduinoJson ins Spiel.

Mit dem Arduino JSON dekodieren

Mit dieser Bibliothek greifst du dir einzelne Daten aus dem JSON-String (den Rohdaten) und speicherst sie in Variablen deiner Wahl. Damit sie jedoch ihre Arbeit erledigen kann, muss sie zunächst wissen, wie groß die Rohdaten sind, um sich selbst genügend Arbeitsspeicher zu reservieren.

Hierfür gibt es einen praktischen Assistenten. Öffne zunächst die URL der API und kopiere dir mit **Strg + A** und **Strg + C** sämtliche Zeichen.

Öffne anschließend den Assistenten von ArduinoJson und füge sie dort ins linke Feld ein.

Anschließend siehst du unter dem Eingabefeld den Abschnitt **Parsing program** – hier befinden sich sämtliche Befehle und Informationen, die du nun benötigst. In der ersten Zeile steht die Speichermenge in der Konstanten **capacity**. In unserem Fall

ist das

```
const size_t capacity = JSON_ARRAY_SIZE(5) + 5*JSON_OBJECT_SIZE(2) + JSON_OBJECT_SIZE(3) + 200;
```

Beachte hierbei: Sollte der JSON-String länger werden (weil z.B. ein ganzer Haufen Astronauten in den Weltraum aufbricht), dann benötigst du auch mehr Speicher für ArduinoJson. Ansonsten würde es zu einer entsprechenden Fehlermeldung kommen.

Der nächste Befehl, den du nun anpassen musst ist dieser:

```
deserializeJson(doc, payload);
```

Hier findest du die Variable **payload**, in der die Rohdaten von der API stecken. Mit der Funktion **deserializeJson()** werden diese nun dekodiert.

Anschließend kannst du sie deinen eigenen Variablen zuweisen. Wir beschränken uns hier ja auf die Anzahl Astronauten im Weltraum, weswegen wir nur eine Variable benötigen. Der Assistent von ArduinoJson schlägt dir bereits eine vor, deren Name auf dem Key („number“) in den JSON-Daten beruht.

```
int number = doc["number"];
```

Um zu prüfen, ob alles funktioniert hat, gibst du diese Variable nun in deinem Seriellen Monitor aus:

```
Serial.println(number);
```

Wir haben hier heute eine 5 stehen – welche Zahl hast du? Hier nun der gesamte Sketch zum Kopieren und Weiterverwenden:

```
#include <ESP8266WiFi.h> //WiFI
#include <ArduinoJson.h> //JSON
#include <ESP8266HTTPClient.h> //API-Abfrage
```

```
// WIFI-Zugangsdaten
const char* ssid = "NETZWERK";
const char* password =  "PASSWORT";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password); //Internet-Verbindung starten

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  delay(1000);
  Serial.println("Hello, world!");
}

void loop() {
  if ((WiFi.status() == WL_CONNECTED)) {

    HTTPClient http; //Instanz von HTTPClient starten
      http.begin("http://api.open-notify.org/astros.json");
//Abfrage-URL
    int httpCode = http.GET(); //Antwort des Servers abrufen

    if (httpCode == 200) {
       String payload = http.getString(); //Daten in eine
Variable speichern
       const size_t capacity = JSON_ARRAY_SIZE(5) + 5 *
JSON_OBJECT_SIZE(2) + JSON_OBJECT_SIZE(3) + 200;
      DynamicJsonDocument doc(capacity);
      deserializeJson(doc, payload);
      int number = doc["number"];
      Serial.println(number);
    }
  }
  delay(10000);
}
```

Wie geht es weiter?

Jetzt wo du weißt, wie du an Daten im JSON-Format herankommst und sie weiterverwendest, stehen dir ganz neue Möglichkeiten offen. Wie wäre es z.B. mit einer LEGO ISS, die leuchtet, wenn die echte ISS über ihr fliegt?