Bern University
of Applied Sciences

Kernmodul Embedded Systems (EMBSY)

IoT-Projekt

Hardware- und Software-Manual

Autor: Daniel Meer (med1)

Dozent: Roger Weber (wbr1)

Datum: 15. November 2016

Version: 1.0

Inhaltsverzeichnis

1. Einleitung

1.1. Allgemeines

Im Rahmen des Kernmoduls Embedded Systems (EMBSY) vertiefen die Studierenden das Erlernte mit einem Miniprojekt. Mit Hilfe eines Gateways sollen die Daten eines funkbasierten Sensorknotens in die Cloud gesendet werden. Als Hardware für das Gateway wird ein Raspberry Pi[1] verwendet. Zusätzlich soll eine Webseite erstellt werden, welche die Sensor-Daten darstellt.

1.2. Systemübersicht

Das System umfasst die in Abbildung 1.1 dargestellten Komponenten. Nachfolgende Auflistung erläutert kurz die einzelnen Komponenten:

- **BLE-Sensor:** Der BLE-Sensor bietet die Möglichkeit die Beschleunigung, die Winkelbeschleunigung und die Temperatur zu messen. Die Kommunikation mit dem Gateway erfolgt dabei über das Protokoll «Bluetooth Low Energy». Da dies nicht Bestandteil des Unterrichts ist, übernimmt die Software «sensor-hub» die Ansteuerung des Sensors.
- **Gateway:** Die Hardware des Gateways ist ein Raspberry Pi 3. Dieses muss an einem Netzwerk mit Internet-Zugang angeschlossen sein, damit es die Sensordaten in die Cloud senden kann. Als Betriebssystem wird Linux eingesetzt. Der Zugriff für die Entwicklung erfolgt über SSH. Dabei steht der benötigte Hostname auf der Unterseite des Gehäuses.

  Auf dem Gateway laufen zwei Applikationen. Die eine nennt sich «sensor-hub» und bietet eine vereinfachte Kommunikation mit den BLE-Sensoren. Die andere Applikation muss von den Studierenden erstellt werden und die Sensordaten in die Cloud weiterleiten.
- **Broker:** Den Studierenden wird ein MQTT-Broker zur Verfügung gestellt. Dort können sie ihre Daten hinsenden und wieder abholen. Die Schnittstelle ist durch das MQTT-Protokoll gegeben.
- **Webseite:** Die Webseite wird von den Studierenden erstellt und soll die Sensordaten beim Broker holen und in einem Diagramm darstellen. Sie kann entweder lokal auf dem Computer des Studierenden laufen oder auf einem Server wie in einer der Übungen während des Unterrichts.

[1] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

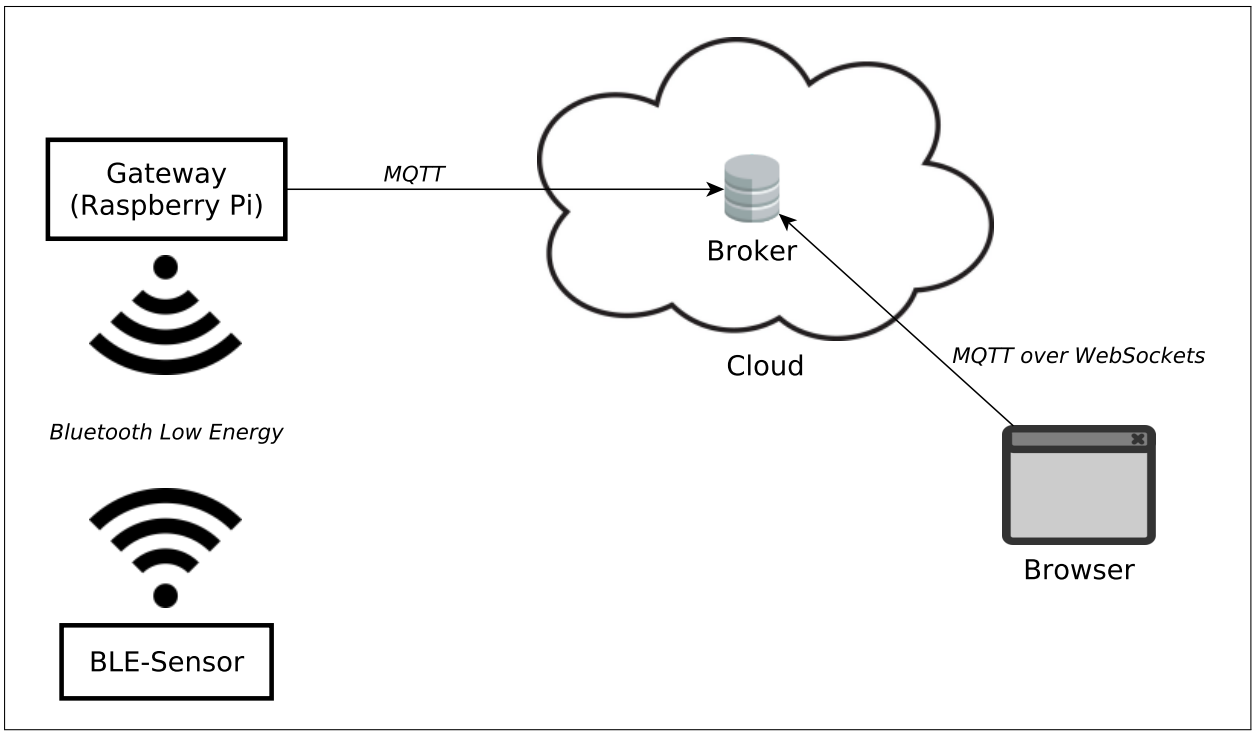

Abbildung 1.1.: Systemübersicht

2. Die Software «sensor-hub»

2.1. Einleitung

Die Software «sensor-hub» wurde entwickelt, um BLE-Sensorknoten an das Internet of Things anzubinden. Dazu läuft die Software als Daemon (Hintergrundprozess) auf einem Gateway. Eine Applikation kann nun über einen Socket mit dem «sensor-hub»-Prozess kommunizieren und die BLE-Sensorknoten steuern.

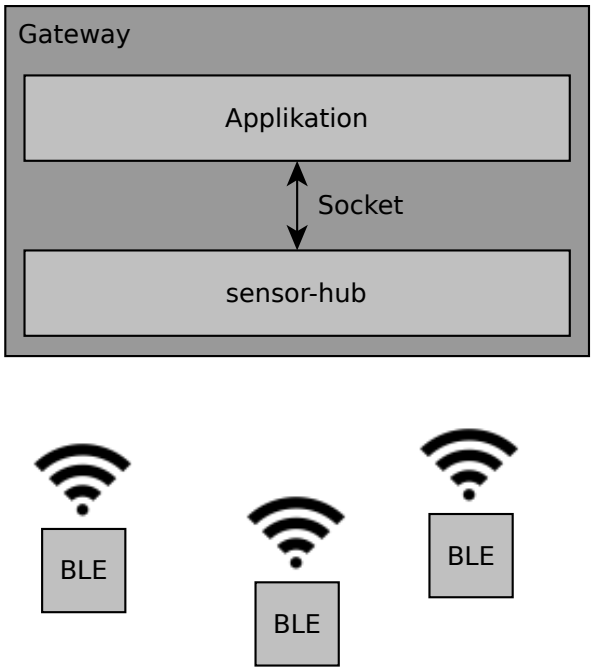

Abbildung 2.1.: Verwendung von sensor-hub

2.2. Schnittstellen-Definition

2.2.1. Allgemein

Um mit dem «sensor-hub»-Prozess zu kommunizieren wird ein UNIX-Socket verwendet. Dieser ist unter folgendem Pfad zu finden: `/tmp/sensor-hub.socket`

Beispiel:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/types.h>
#include <unistd.h>

void main(void)
{
    /* Get socket file descriptor */
    int socket_fd = socket(AF_UNIX, SOCK_STREAM, 0);

    /* Set socket address */
    struct sockaddr_un socket_addr;
    socket_addr.sun_family = AF_UNIX;
    strcpy(socket_addr.sun_path, "/tmp/sensor-hub.socket");
    int addr_len = strlen(socket_addr.sun_path) + sizeof(socket_addr.sun_family);

    /* Connect to socket */
    connect(socket_fd, (struct sockaddr *)&socket_addr, addr_len);

    /* Write string from stdin to socket */
    char buffer[100];
    fgets(buffer, 100, stdin);
    send(socket_fd, buffer, strlen(buffer), 0);

    /* Read response from socket */
    int t = recv(socket_fd, buffer, 99, 0);
    buffer[t] = '\0';
    printf("%s", buffer);

    /* Close socket */
    close(socket_fd);
}
```

Das Format der Nachrichten, welche über den Socket gesendet werden, ist JSON. Die Applikation kann den sensor-hub steuern, indem er Kommandos (**Commands**) über den Socket schickt.

Sobald ein Ereignis eintritt (neuer BLE-Sensor gefunden, Sensor erfolgreich konfiguriert, etc.) sendet der sensor-hub ein **Event** über den Socket. Auf diese Weise kommunizieren die beiden Teile asynchron miteinander.

2.2.2. Commands

- `StartBleScan`

  Startet den BLE-Scan.

  Nach einigen Sekunden wird dieser wieder gestoppt. Da kein Sensorknoten angesprochen wird, bleibt das device-Feld leer.

  z.B.:

```json
{
  "device": "",
  "command": "StartBleScan"
}
```

- **`StopBleScan`**

  Stoppt den BLE-Scan vorzeitig.

  Da kein Sensorknoten angesprochen wird, bleibt das «device»-Feld leer.

  z.B.:

```
{
  "device": "",
  "command": "StopBleScan"
}
```

- **`Connect`**

  Verbindet einen Sensor-Knoten, welcher vorher gescannt wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command":"Connect"
}
```

- **`Disconnect`**

  Trennt die Verbindung zu einem Sensor-Knoten.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "Disconnect"
}
```

- **`GetDeviceInfo`**

  Liest die Device-Informationen über das TXW51 aus.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "GetDeviceInfo"
}
```

- **`GetTemperature`**

  Liest den Temperatursensor aus.

  Damit der Temperatursensor funktioniert, muss das Gyroskop eingeschaltet sein. Ansonsten wird immer derselbe Wert zurückgegeben.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "GetTemperature"
}
```

- `ConfigAccel`

  Konfiguriert den Beschleinigungssensor.

  Es wird ein «data»-Objekt mit der entsprechenden Konfiguration benötigt.

  - **on:** `true`, wenn der Sensor eingeschaltet werden soll; `false` wenn er ausgeschaltet werden soll.
  - **fullscale:** Messbereich des Beschleunigungssensors. Nimmt folgende Werte an: 2, 4, 6, 8, 16 (wobei 2 entsprechen $\pm 2\,\mathrm{g}$)
  - **odr:** Abtastrate des Beschleunigungssensors. Nimmt folgende Werte an: 0, 3.125, 6.25, 12.5, 25, 50, 100, 400, 800, 1600 (wobei 25 entsprechen 25 Hz)

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "ConfigAccel",
  "data": {
    "on": true,
    "fullscale": 4,
    "odr": 12.5
  }
}
```

- `ConfigGyro`

  Konfiguriert das Gyroskop.

  Es wird ein «data»-Objekt mit der entsprechenden Konfiguration benötigt.

  - **on:** `true`, wenn der Sensor eingeschaltet werden soll; `false` wenn er ausgeschaltet werden soll.
  - **fullscale:** Messbereich des Gyroskops. Nimmt folgende Werte an: 250, 500, 2000 (wobei 250 entsprechen $\pm 250\,^\circ/\mathrm{s}$
  - **odr:** Abtastrate des Gyroskops. Nimmt folgende Werte an: 95, 190, 380, 760 (wobei 95 entsprechen 95 Hz)

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "ConfigGyro",
  "data": {
    "on": true,
    "fullscale": 500,
    "odr": 95
  }
}
```

- `ConfigTemp`

  Konfiguriert den Temperatursensor-Sampler.

  Der Temperatursensor-Sampler liefert periodisch den Wert des Temperatursensors. Damit muss er nicht von einem externen Programm gepollt werden. Der Temperatursensor funktioniert nur, wenn das Gyroskop eingeschaltet ist.

  Es wird ein «data»-Objekt mit der entsprechenden Konfiguration benötigt.

  - **on:** `true`, wenn der Sensor eingeschaltet werden soll; `false` wenn er ausgeschaltet werden soll.
  - **odr:** Abtastrate des Temperatursensors. Nimmt einen beliebigen Wert in Hz an.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "ConfigTemp",
  "data": {
    "on": true,
    "odr": 1
  }
}
```

- `StartMeasurement`

  Startet eine Messung.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "StartMeasurement"
}
```

- `StopMeasurement`

  Stoppt eine laufende Messung.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "command": "StopMeasurement"
}
```

2.2.3. Events

- `DeviceDiscovered`

  Wird jedesmal gesendet, wenn ein neuer BLE-Sensorknoten gefunden wurde.

  z.B.:

```
{
  "device": "",
  "event": "DeviceDiscovered",
  "data": {
    "id": "FD:55:8D:40:FB:C2",
    "name": "TXW51"
  }
}
```

- **`DeviceConnected`**

  Wird gesendet, wenn die Verbindung zu einem BLE-Sensorknoten hergestellt wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "DeviceConnected"
}
```

- **`DeviceDisconnected`**

  Wird gesendet, wenn die Verbindung zu einem BLE-Sensorknoten beendet wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "DeviceDisconnected"
}
```

- **`DeviceInfo`**

  Wird gesendet, wenn die Device-Informationen angefragt wurden.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "DeviceInfo",
  "data": {
    "manufacturer": "BFH",
    "model": "TXW51",
    "serial": "002",
    "hw": "1.0",
    "fw": "1.0",
    "name": "John"
  }
}
```

- **`Temperature`**

  Wird gesendet, wenn die Temperatur gemessen wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "Temperature",
  "data": {
    "value": 23,
    "unit": "°C"
  }
}
```

- `AccelData`

  Wird gesendet, wenn neue Beschleunigungsdaten vorhanden sind.

  Im Moment werden die Daten noch roh als 16-Bit-Werte gesendet. Für die Umrechnung siehe Kapitel 2.2.4.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "AccelData",
  "data": {
    "x": -558,
    "y": -4917,
    "z": 15626,
    "unit": ""
  }
}
```

- `GyroData`

  Wird gesendet, wenn neue Gyroskopdaten vorhanden sind.

  Im Moment werden die Daten noch roh als 16-Bit-Werte gesendet. Für die Umrechnung siehe Kapitel 2.2.4.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "GyroData",
  "data": {
    "x": 638,
    "y": 321,
    "z": 67,
    "unit": ""
  }
}
```

- `MeasurementStopped`

  Wird gesendet, wenn eine Messung gestoppt wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "MeasurementStopped"
}
```

- `AccelConfigured`

  Wird gesendet, wenn der Beschleunigungssensor konfiguriert wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "AccelConfigured"
}
```

- `GyroConfigured`

  Wird gesendet, wenn das Gyroskop konfiguriert wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "GyroConfigured"
}
```

- `TempConfigured`

  Wird gesendet, wenn der Temperatursensor konfiguriert wurde.

  z.B.:

```
{
  "device": "FD:55:8D:40:FB:C2",
  "event": "TempConfigured"
}
```

2.2.4. Umrechnung der Messwerte

In den Events `AccelData` und `GyroData` werden die Rohdaten des LSM330 Sensors gesendet. Die Daten sind im Zweierkomplement codiert. Je nach gewähltem Messbereich ist die Sensitivität anders. Aus diesem Grund sind die Werte in der Tabelle 2.1 aufgelistet.

Tabelle 2.1.: Umrechnung der Beschleunigungs- und Gyro-Daten [1]

Sensor	**Messbereich**	**Sensitivität**
Accelerometer	±2 g	0.061 mg/digit
	±4 g	0.122 mg/digit
	±6 g	0.183 mg/digit
	±8 g	0.244 mg/digit
	±16 g	0.732 mg/digit
Gyroskop	±250 dps	8.75 mdps/digit
	±500 dps	17.50 mdps/digit
	±2000 dps	70 mdps/digit

3. Arbeiten mit dem MQTT-Protokoll

3.1. Allgemeines

MQTT ist ein Protokoll, welches sich wegen des einfachen Designs und des geringen Overheads, besonders gut für Anwendungen im Internet of Things eignet. Es funktioniert nach dem Publisher-Subscriber-Prinzip und ist ein OASIS Standard[4].

Um das MQTT-Protokoll zu verwenden, setzen wir die Libraries vom Paho-Projekt[2] ein. Diese sind weit verbreitet und für viele Sprachen verfügbar.

3.2. Broker

Wir verwenden den MQTT Broker der BFH. Nachfolgend die Zugangsdaten:

- **Adresse:** iot.i3s.bfh.ch
- **Port über eine TCP-Verbindung:** 1883
- **Port über eine WebSocket-Verbindung:** 9001
- **Username:** *Wird Ihnen im Unterricht übergeben.*
- **Passwort:** *Wird Ihnen im Unterricht übergeben.*

Achtung:
Bei einer Verbindung über WebSockets im Browser funktioniert die Namensauflösung für den Broker nicht. Aus diesem Grund müsse Sie die IP-Adresse direkt eingeben. Ansonsten kann sich der Browser nicht mit dem Broker verbinden.

3.3. Verwendung der Library in C

Nachfolgend die Dokumentation der Library:

- Git: `https://github.com/eclipse/paho.mqtt.c`
- Dokumentation: `http://www.eclipse.org/paho/files/mqttdoc/Cclient/index.html`

Die Library können Sie über Moodle herunterladen. In Listing 3.1 ist als Beispiel eine pro-Datei zu sehen, wo die Paho-Library eingebunden wird. Passen Sie die Pfade Ihrem Projekt entsprechend an. Aufgrund der einfacheren Handhabung werden wir eine statische Version der Library verwenden.

Listing 3.1: Einbinden der MQTT-Library in das Projekt

```
TEMPLATE = app

CONFIG += \
console \
static

CONFIG -= \
app_bundle \
qt

QMAKE_CFLAGS += -std=gnu99

LIBS += -lpthread -lrt
LIBS += -L$$PWD/lib/paho/lib/static/ -lpaho-mqtt3a_static

INCLUDEPATH += lib/paho/include
DEPENDPATH += lib/paho/include

target.path = /home/pi
INSTALLS += target

SOURCES += src/main.c \
src/mqtt.c \
src/unixsocket.c

HEADERS += \
src/mqtt.h \
src/unixsocket.h
```

Im Ordner «include» befinden sich drei Header-Dateien. Denn die Library kann auf drei verschiedene Arten verwendet werden:

- **MQTTAsync.h:** Diese Version der Library funktioniert komplett asynchron. D.h. keine der Funktionen blockiert den Programmverlauf. Statt dessen werden Callbacks verwendet. Es wird empfohlen diese Version zu nehmen.
- **MQTTClient.h:** Dies ist die ältere bzw. originale Version der API. Manche der Funktionen blockieren.
- **MQTTClientPersistence.h:** Diese Version bietet noch zusätzlich persistente Speicherung der Daten. Oftmals wird dies jedoch nicht benötigt.

Auf Moodle befindet sich ebenfalls eine zip-Datei mit offiziellen Beispielen.

Hier ein Überblick über die wichtigsten Funktionen.

- **`MQTTAsync_create()`:** Erzeugt einen MQTT Client.
- **`MQTTAsync_setCallbacks()`:** Registriert einige Callbacks des MQTT Clients.
- **`MQTTAsync_destroy()`:** Zerstört einen MQTT Client.
- **`MQTTAsync_connect()`:** Setzt einen Verbindungsaufbau in Gang.
- **`MQTTAsync_sendMessage()`:** Sendet eine Nachricht.
- **`MQTTAsync_subscribe()`:** Abonniert ein MQTT Topic.

3.4. Verwendung der Library in JavaScript

Nachfolgend die Dokumentation der Library:

- Git: https://github.com/eclipse/paho.mqtt.javascript
- Dokumentation: http://www.eclipse.org/paho/files/jsdoc/index.html

Die Library können Sie über Moodle herunterladen.

Hier ein Überblick über die wichtigsten Funktionen.

- **`new Paho.MQTT.Client()`:** Erzeugt einen MQTT Client.
- **`connect()`:** Setzt einen Verbindungsaufbau in Gang.
- **`subscribe()`:** Abonniert ein MQTT Topic.
- **`send()`:** Sendet eine Nachricht.
- **`new Paho.MQTT.Message()`:** Erzeugt eine Nachricht, welche versendet werden kann.

In JavaScript werden zwei Methoden zur Verfügung gestellt, um JSON-Strings in JavaScript-Objekte zu wandeln und umgekehrt:

- **`JSON.parse(jsonString)`:** Parst einen JSON-String und erzeugt daraus ein JavaScript-Objekt.
- **`JSON.stringify(jsObject)`:** Generiert aus einem JavaScript-Objekt einen JSON-String.

4. Verwendung von Google Charts

Es gibt sehr viele verschiedene Libraries in JavaScript um Diagramme zu zeichnen. Wir verwenden hier Google Charts, welche weit verbreitet sind und sich durch ihre einfache Bedienbarkeit auszeichnen.

Nachfolgend die Dokumentation der Library:

- Git: *Die Library ist zwar gratis, aber nicht Open-Source. Somit ist keine Repo verfügbar.*
- Dokumentation: https://developers.google.com/chart/interactive/docs/

Sie müssen die Library nicht herunterladen, sondern können direkt eine Internet-Adresse im Script-Tag angeben (siehe Listing 4.1).

Listing 4.1: Einbinden der «Google Charts»-Library in das Projekt

```
<script src="https://www.gstatic.com/charts/loader.js" type="text/javascript"></script>

<script type="text/javascript">
google.charts.load('current', {packages: ['line']});
google.charts.setOnLoadCallback(initChart);

function initChart() {
...
}
```

Hier ein Überblick über die wichtigsten Funktionen.

- **`google.charts.load()`:** Lädt das entsprechende Charts-Package.
- **`google.charts.setOnLoadCallback()`:** Registriert einen Callback für wenn das Package geladen wurde.
- **`new google.visualization.DataTable()`:** Erzeugt eine Tabelle für Datenwerte.
- **`addColumn()`:** Erweitert die Daten-Tabelle um eine Spalte.
- **`addRow()`:** Fügt eine Reihe (einen Datensatz) der Daten-Tabelle hinzu.
- **`new google.charts.Line()`:** Erzeugt einen Line-Graphen.
- **`draw()`:** Stellt den aktuelle Datensatz im Diagramm dar.

Literaturverzeichnis

[1] *TXW51 Sensor Node – Bedienungsanleitung*, 1st ed., Berner Fachhochschule, Februar 2015.

[2] eclipse.org, "Paho – Open Source messaging for M2M," Stand: November 2016. [Online]. Available: https://eclipse.org/paho/

[3] json.org, "JSON," Stand: August 2013. [Online]. Available: http://www.json.org/

[4] knolleary, "MQTT v3.1.1 now an OASIS Standard," Stand: November 2016. [Online]. Available: http://mqtt.org/2014/11/mqtt-v3-1-1-now-an-oasis-standard

[5] Wikipedia, "JavaScript Object Notation," Stand: August 2013. [Online]. Available: https://de.wikipedia.org/wiki/JavaScript_Object_Notation

Anhang A.

JSON

A.1. Allgemeines

Die «JavaScript Object Notation»[1], kurz JSON, ist ein kompaktes Datenformat in für Mensch und Maschine einfach lesbarer Textform zum Zweck des Datenaustauschs zwischen Anwendungen. Parser für JSON existieren in praktisch allen verbreiteten Sprachen. In Verbindung mit JavaScript wird JSON für Ajax oder WebSockets zur Übertragung von Daten zwischen Client und Server verwendet [5].

- JSON steht für **J**ava**S**cript **O**bject **N**otation
- JSON ist ein kompaktes textbasiertes Datenformat
- JSON ist unabhängig von der Programmiersprache. JSON benutzt JavaScript-Syntax um die Datenobjekte zu beschreiben.
- JSON ist «selbst beschreibend» (lesbar für den Menschen) und einfach zu verstehen.

A.2. JSON Syntax

A.2.1. Allgemeines

- Daten werden in Key-Value Paaren angeordnet
- Daten werden durch Kommas separiert
- Geschweifte Klammern beinhalten ein Objekt
- Eckige Klammern beinhalten ein Array.

[1] www.json.org

A.2.2. JSON Key-Value Paare

Daten werden in Key-Value Paaren angeordnet. Ein Key-Value Paar umfasst einen Key in Anführungszeichen gefolgt von einem Dopplepunkt und dem Value. Ein Beispiel eines JSON-Objekts ist im Listing A.1 dargestellt. Das Objekt besteht aus zwei Key-Value Paaren (`"cmd":"set"` sowie `"value":58`). Einer der Values ist ein String und der andere eine Zahl (siehe Abschnitt A.2.3).

Listing A.1: Beispiel eines JSON-Objekts

```
{
  "cmd": "set",
  "value": 58
}
```

A.2.3. JSON Values

Folgende Values sind möglich (siehe auch Abbildung A.1):

- Ein String in Anführungszeichen
- Eine Zahl (Integer oder Floating point)
- Ein Objekt umfasst mit geschweiften Klammern
- Ein Array umfasst mit eckigen Klammern
- Ein Boolean (`true` oder `false`)
- Der `null` Value

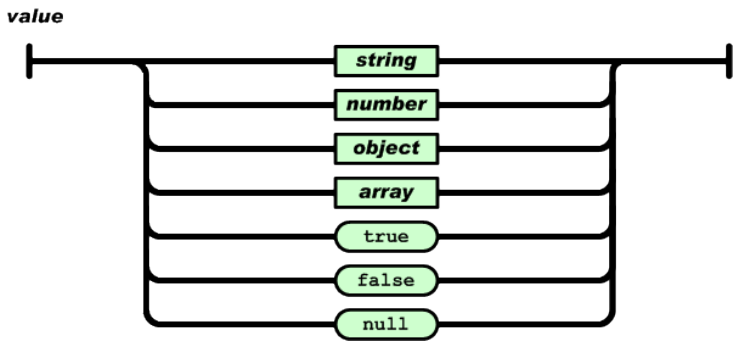

Abbildung A.1.: JSON Values [3]

A.2.4. JSON Objekte

JSON-Objekte werden durch geschweifte Klammern umfasst. Ein Objekt kann mehrere Key-Value Paare beinhalten. Abbildung A.2 zeigt den grundlegenden Aufbau von JSON-Objekten.

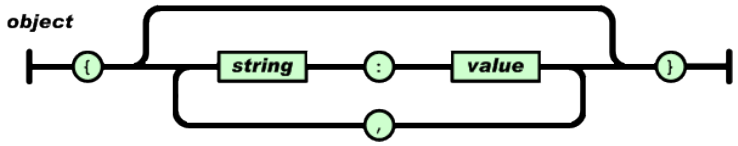

Abbildung A.2.: JSON Objekte [3]

A.3. JSON und JavaScript

A.3.1. Allgemeines

Mit einfachen Beispielen soll dieser Abschnitt einen Einstieg in die Benutzung von JSON mit JavaScript geben. Alle Beispiele sind eigenständig und sollten in einem Browser geöffnet werden können.

A.3.2. JSON-Objekt erstellen

Im Beispiel von Listing A.2 wird ein JSON-Objekt erstellt mit den beiden Key-Value Paaren `"temp_ist":23` und `"temp_soll":30`. Die beiden Values werden anschliessend im entsprechenden Textfeld eingefügt.

Listing A.2: Beispiel JSON-Objekt erstellen

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Create JSON-Object</title>
        <script type="text/javascript">

            var data = {"temp_ist":23,"temp_soll":30};

            window.onload=function(){
                document.getElementById('txt_temp_ist').value = data.temp_ist;
                document.getElementById('txt_temp_soll').value = data.temp_soll;
            }
        </script>
    </head>
    <body>
        <div class="input">
            <div class="label">Temperature Ist:</div>
            <input type="text" id="txt_temp_ist" readonly="readonly" />
        </div>
        <div class="input">
            <div class="label">Temperature Soll:</div>
            <input type="text" id="txt_temp_soll" />
        </div>
    </body>
</html>
```

A.3.3. JSON Value verändern

Listing A.3 ergänzt das Beispiel von Abschnitt A.3.2. Es wird zusätzlich eine Funktion hinzugefügt, welche bei Änderungen im Textfeld «Temperature Soll» aufgerufen wird. Die Änderung wird im JSON-Objekt übernommen und der neue Wert wird in einer Dialogbox dargestellt.

Listing A.3: Beispiel JSON-Objekt erstellen

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Edit JSON-Value</title>
        <script type="text/javascript">

            var data = {"temp_ist":23,"temp_soll":30};

            window.onload=function(){
                document.getElementById('txt_temp_ist').value = data.temp_ist;
                document.getElementById('txt_temp_soll').value = data.temp_soll;
            }

            function changeTempSoll() {
                data.temp_soll = document.getElementById('txt_temp_soll').value;
                alert("New Soll Temperature: " + data.temp_soll);
            }
        </script>
    </head>
    <body>
        <div class="input">
            <div class="label">Temperature Ist:</div>
            <input type="text" id="txt_temp_ist" readonly="readonly" />
        </div>
        <div class="input">
            <div class="label">Temperature Soll:</div>
            <input type="text" id="txt_temp_soll"  onchange="changeTempSoll()"/>
        </div>
    </body>
</html>
```

A.3.4. JSON-Objekt senden und empfangen

Listing A.4 simuliert das Senden und Empfangen eines JSON-Objekts. Das zu sendende JSON-Objekt wird in einem Textfeld dargestellt und kann verändert werden. Um ein JSON-Objekt in eine Textarea einfügen zu können, muss es mit Hilfe von `JSON.stringify()` in ein String umgewandelt werden. Wird eine Änderung im Textfeld vorgenommen, wird die Funktion `send()` aufgerufen. Der String kann unverändert weitergeleitet werden (im Beispiel an die Funktion `receive()`). Dort wird durch Aufruf von `JSON.parse()` der empfangene String wieder in ein JSON-Objekt umgewandelt. In einem Loop werden zum Schluss alle verfügbaren Key-Value Paare in einer Dialogbox dargestellt.

Listing A.4: Beispiel JSON-Objekt senden und empfangen

```
<html>
    <head>
        <meta charset="utf-8">
        <title>Send/receive JSON-Value</title>
        <script type="text/javascript">

            window.onload=function(){
                var TestData = {"temp_ist":23,"temp_soll":30};
                document.getElementById('txt_send').value = JSON.stringify(TestData);
            }

            function send() {
                var jsonTxt = document.getElementById('txt_send').value;
                receive(jsonTxt);
            }

            function receive(data) {
                var jsonObject = JSON.parse(data);
                for (var key in jsonObject) {
                    alert("Key: " + key + " and Value: " + jsonObject[key]);
                }
            }
        </script>
    </head>
    <body>
        <div class="input">
            <div class="label">Zu sendender JSON-String:</div>
            <input type="text" id="txt_send"  onchange="send()" />
        </div>
    </body>
</html>
```

A.4. JSON und C

A.4.1. Allgemeines

Damit die ankommenden und ausgehenden Daten von der Applikation verarbeitet werden können, stehen die im Abschnitt A.4.2 beschriebenen Funktionen zur Verfügung. Es stehen nur Funktionen zur Verfügung um String-Values einzufügen oder auszulesen.

A.4.2. JSON API

json_createFromBuffer

```
json_t * json_createFromBuffer(char *pcMsg, uint32_t u32Length);
```

Summary

Erzeut ein neues JSON-Objekt aus einem Charakter-Buffer. Das zurückgegebene JSON-Objekt muss am Ende mit `json_cleanup()` freigegeben werden.

Parameters

`pcMsg`	Der Charakter-Buffer in welchem sich die zu dekodierende JSON-Nachricht befindet.
`u32Length`	Länge der JSON-Nachricht innerhalb des Buffers.

Return Values

Gibt das erzeugte JSON-Objekt zurück oder `NULL` falls ein Fehler aufgetreten ist.

json_createFromString

```
json_t * json_createFromString(char *pcMsg);
```

Summary

Erzeut ein neues JSON-Objekt aus einem String (Folge von Charaktern welche mit einem $'\backslash 0'$-Charakter abgeschlossen sind). Das zurückgegebene JSON-Objekt muss am Ende mit `json_cleanup()` freigegeben werden.

Parameters

`pcMsg`	Der String in welchem sich die zu dekodierende JSON-Nachricht befindet.

Return Values

Gibt das erzeugte JSON-Objekt zurück oder NULL falls ein Fehler aufgetreten ist.

json_createEmpty

```
json_t * json_createEmpty(void);
```

Summary

Erzeugt eine neues leeres JSON-Objekt. Die String-Repräsentation des neu erzeugten Objektes hat die Form: "{}". Das zurückgegebene JSON-Objekt muss am Ende mit `json_cleanup()` freigegeben werden.

Return Values

Gibt das erzeugte JSON-Objekt zurück oder NULL falls ein Fehler aufgetreten ist.

json_cleanup

```
Err json_cleanup(json_t *pJson);
```

Summary

Zurückgeben aller Ressourcen welche durch das JSON-Objekt belegt werden.

Parameters

`pJson`	JSON-Objekt welches freigegeben werden soll.

Return Values

`NONE`	Funktion erfolgreich ausgeführt.
`ERR_PARAM`	Parameter Fehler.

json_getString

```
char * json_getString(json_t * pJson)
```

Summary

Gibt die String-Repräsentation eines JSON-Objektes zurück. Die durch den String belegten Ressourcen, müssen mit `json_freeString()` freigegeben werden.

Parameters

`pJson`	JSON-Objekt welches enkodiert wird.

Return Values

Die String-Repräsentation des JSON-Objektes oder `NULL` falls ein Fehler aufgetreten ist.

json_freeString

```
void json_freeString(char * pcJsonMsg)
```

Summary

Gibt die Ressourcen zurück welche durch den Aufruf der Funktion `json_getString()` erzeugt wurden.

Parameters

`pcJsonMsg`	Die Ressourcen welche durch `son_getString()` erzeugt wurden und freizugeben sind.

json_getStringValue

```
char * json_getStringValue(json_t * pJson, char * pcKey);
```

Summary

Gibt den zum Key dazugehörigen String-Value aus dem JSON-Objekt zurück. Der zurückgegebene Value ist ein String und darf nicht verändert werden.

Parameters

`pJson`	JSON-Objekt
`pcKey`	Key-String

Return Values

Der zum Key dazugehörige String-Value oder `NULL` falls dieser nicht verfügbar ist.

json_setKeyValue

```
Err json_setKeyValue(json_t * pJson, char * pcKey, char * pcValue);
```

Summary

Setzt innerhalb eines JSON-Objektes ein neues Key-Value Paar. Falls der Key innerhalb des JSON-Objektes bereits vorhanden ist, wird der alte Value mit dem Neuen überschrieben.

Parameters

`pJson`	JSON-Objekt
`pcKey`	Key-String
`pcValue`	Value-String

Return Values

`NONE`	Funktion erfolgreich ausgeführt.
`ERR_PARAM`	Parameter Fehler.
`ERR_JSON_PACK`	Key-Value Paar konnte nicht eingefügt werden.

Example

Listing A.5 zeigt ein Beispiel für die Benutzung der JSON-API. Im ersten Teil wird aus einem String ein JSON-Objekt erzeugt und der Value zum Key `"Hello"` wird ausgegeben. Der zweite Teil erzeugt ein leeres JSON-Objekt und fügt diesem das Key-Value Paar `"Hello":"World"` hinzu. Anschliessend wird die String-Repräsentation des JSON-Objektes ausgegeben.

Listing A.5: Beispiel zur Benutzung der JSON-API

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#include "json.h"

int main(int argc, char **argv) {

    char *   pcString = "{\"Hello\":\"World\"}";
    char *   pcValue = NULL;
    char *   pcMsg = NULL;
    json_t * jsonMsg = NULL;

    // 1.)
    // Create a Json object out of a string
    jsonMsg = json_createFromString(pcString);
    // jsonMsg = json_createFromBuffer(pcString, 17); // would achieve the same!
    if(jsonMsg != NULL) {
```

```
        // Get the value which corresponds to the key Hello
        pcValue = json_getStringValue(jsonMsg, "Hello");
        if(pcValue != NULL)
            printf("%s\n", pcValue);

        // Don't forget to free Json object!
        json_cleanup(jsonMsg);
    }

    // 2.)
    // Create an empty Json object
    jsonMsg = json_createEmpty();
    if(jsonMsg != NULL) {

        // Add a key-value pair
        json_setKeyValue(jsonMsg, "Hello", "World");

        // Get the string representation of the Json object
        pcMsg = json_getString(jsonMsg);
        if(pcMsg != NULL) {
            printf("%s\n", pcMsg);

            // Don't forget to free string resource!
            json_freeString(pcMsg);
        }
        // Don't forget to free Json object!
        json_cleanup(jsonMsg);
    }
    return EXIT_SUCCESS;
}
```