**PAW IoT Cube**

**SoSe 19**

**Dokumentation Firmware ESP8266**

Stand: 07.06.2019

Version: 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Änderungen** | **Autor** |
| 1 | 07.06.19 | erstellt: Hinzufügen eines neuen Sensors | SD |
| 2 | 07.06.19 | erstellt: Aufbau der Firmware  erstellt: Hinzufügen eines neuen Aktors  geändert: Hinzufügen eines neuen Sensors | WD |
|  |  |  |  |

Inhalt

[1 Aufbau der Firmware 3](#_Toc10815434)

[1.1 Allgemeiner Aufbau der Firmware 3](#_Toc10815435)

[1.2 Genutzte Externe Bibliotheken 3](#_Toc10815436)

[2 Hinzufügen eines neuen Sensors 3](#_Toc10815437)

[3 Hinzufügen eines neuen Aktors 5](#_Toc10815438)

# Aufbau der Firmware

## Allgemeiner Aufbau der Firmware

Der Hauptteil der Firmware besteht aus den folgenden Dateien:

|  |  |
| --- | --- |
| main.cpp | WLAN-Verbindung  MQTT-Verbindung  MQTT-Handshake  Hauptroutine |
| readSensors.h/cpp | Funktionen zum auslesen der einzelnen Sensoren und Schicken der Werte vie MQTT |
| setActuators.h/cpp | Funktionen zum Ansteuern der Aktoren |
| controlLed.h/cpp | Funktionen zum Ansteuern der Status-LED |

Dazu kommt noch eine Konfigurationsdatei settings.h in der WLAN-Zugangsdaten und Broker-IP spezifiziert werden können, sowie zwischen der Ausgabe von Debug-Nachrichten über die serielle Schnittstelle oder Statusanzeigen via LED umgeschaltet werden kann.

## Genutzte Externe Bibliotheken

|  |  |
| --- | --- |
| **allgemeine Bibliotheken** | |
| **Bibliothek** | **Link** |
| PubSubClient | <https://www.arduinolibraries.info/libraries/pub-sub-client> |
| ESP8266Wifi | <https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi> |

|  |  |
| --- | --- |
| **Sensorbibliotheken** | |
| **Bibliothek** | **Link** |
|  |  |
|  |  |
|  |  |

# Hinzufügen eines neuen Sensors

Um einen neuen Sensor in die bestehende Firmware zu integrieren sind mehrere Schritte notwendig.

Änderungen in readSensorValues.h

* Festlegen eines neuen Sensornamens und eines entsprechenden #define
  + Achtung: für diesen #define muss ein Wert zwischen 0 und 19 gewählt werden, der noch nicht vorhanden ist
  + Hinweis: Die Nummerierung der DIP-Schalter ist wie folgt zu interpretieren

|  |  |
| --- | --- |
| Nr. | Wert |
| 1 | 16 |
| 2 | 8 |
| 3 | 4 |
| 4 | 2 |
| 5 | 1 |

* + - Die Summe aller Werte der aktiven DIP-Schalter ist der Wert der in den defines angegeben ist
* Gegebenenfalls vorhandene Bibliothek zum Auslesen des Sensors einbinden
* Anlegen eines Funktionsprototyps zum Auslesen des entsprechenden Sensors mit entsprechenden Übergabeparametern

Änderungen in readSensorValues.cpp

* Implementierung einer Funktion
  + Auslesen des Sensors
  + Ggf. Formatierung des Messwertes zu einem String
  + MQTT-Publish
* Orientierung bieten bereits implementierten Funktionen für andere Sensoren

Änderungen in main.cpp

* Abhängig von der Bibliothek, die zum Auslesen des Sensors verfügbar ist, ist gegebenenfalls eine Initialisierung von Variablen vor der setup() Funktion notwendig
* Änderungen in der setup() Funktion
  + Erweiterung des ersten switch-case um das zusätzlich in readSensorValues.h definierte #define
  + Gegebenenfalls Initialisierung von Variablen (abhängig von gegebener Bibliothek zum Auslesen des Sensors)
  + Generieren der discovery message
    - Achtung: Der übergebene Name muss mit dem in Nodered spezifizierten Discovery-Namen übereinstimmen, siehe hierzu die entsprechende Dokumentation zum Implementieren eines neuen Sensors in Nodered
  + Orientierung bieten die bereits implementierten Sensoren
* Änderung in der loop() Funktion
  + Erweiterung des switch-case um das zusätzlich in readSensorValues.h definierte #define
  + Generieren eines Topics pro aufzunehmenden Messwerttyp
    - Achtung: die Topics müssen hierbei mit den in Nodered spezifizierten Topics übereinstimmen. Existiert der gewünschte Messwerttyp noch nicht, so muss hierfür ein neuer Zweig in Nodered angelegt werden. Siehe hierzu die entsprechende Dokumentation
  + Kombinieren der vom Broker erhaltenen ID mit zuvor definierten Topics (Orientierung bieten die bereits implementierten Sensoren)
  + Aufrufen der Funktion, die in readSensorValues.cpp zum Auslesen des Sensors definiert wurde

# Hinzufügen eines neuen Aktors

Auch für die Implementierung eines neuen Aktors sind einige Schritte notwendig.

Änderungen in setActuators.h

* Festlegen eines neuen Aktornamens und eines entsprechenden #define
  + Achtung: für diesen #define muss ein Wert zwischen 20 und 31 gewählt werden, der noch nicht vorhanden ist
* Gegebenenfalls vorhandene Bibliothek zum Ansteuern des Aktors einbinden
* Anlegen eines Funktionsprototyps zum Ansteuern des entsprechenden Aktors mit entsprechenden Übergabeparametern

Änderungen in setActuators.cpp

* Implementierung einer Funktion zur Verarbeitung des übergebenen Wertes und Ansteuern des Aktors

Änderungen in main.cpp

* Änderungen in der setup() Funktion
  + Erweiterung des ersten **und** zweiten switch-case um das zusätzlich in setActuators.h definierte #define
  + Generieren der discovery message im ersten switch-case
    - Achtung: Der übergebene Name muss mit dem in Nodered spezifizierten Discovery-Namen übereinstimmen, siehe hierzu die entsprechende Dokumentation zum Implementieren eines neuen Sensors in Nodered
  + Generieren eines Topics für die Aktoren-Befehle im zweiten switch-case und Abonnieren dieses Topics
    - Achtung: die Topics müssen hierbei mit den in Nodered spezifizierten Topics übereinstimmen. Existiert der gewünschte Messwerttyp noch nicht, so muss hierfür ein neuer Zweig in Nodered angelegt werden. Siehe hierzu die entsprechende Dokumentation
  + Orientierung bieten die bereits Implementierten Aktoren
* Änderungen in der commandReceive() Funktion
  + Erweiterung des switch-case um das zusätzlich in readSensorValues.h definierte #define
  + Konvertierung des empfangenen Command-Strings und Aufruf der in setActuators.h/cpp neu hinzugefügten Funktion