60661-5 Eigene IoT-Loesungen mit ESP32

Seite 39:

```
001 /*
002 * LED-Blink-Sketch
003 */
004 int LEDPin = 2; // ESP32 Blue-LED, GPIO2
005 int pause = 600;
006
007 // Vorbereitungs- und Initialisierungsroutine
008 void setup() {
009 // Digital-Pin 2 als Ausgabe initialisieren
010 pinMode(LEDPin, OUTPUT);
011 }
012
```

Seite 40:

```
013 // die Schleife, die immer wieder durchlaufen wird
014 void loop() {
015 digitalWrite(LEDPin, HIGH); // schaltet die LED ein
016 delay(pause); // etwas warten
017 digitalWrite(LEDPin, LOW); // schaltet die LED aus
018 delay(pause); // etwas warten
019 }
```

Seite 41:

```
001 /*
002 * LEDs, die mittels Touch-Sensoren ein-/ausgeschaltet werden
003 * mit Ausgabe auf dem seriellen Monitor
004 */
005 int LEDPin = 2; // GPIO2, Pin G2
006 int touchPinEin = 4; // T0, GPIO4, Pin G4
007 int touchPinAus = T3; // T3, GPIO15, Pin G15
008 int pause = 600;
009 int touchEin = 0; // eingelesener Wert für ein
010 int touchAus = 0; // eingelesener Wert für aus
011
012 // Vorbereitungs- und Initialisierungsroutine
013 void setup() {
014 // Digital-Pin 2 als Ausgabe initialisieren
015 pinMode(LEDPin, OUTPUT);
016 // notwendig für korrekte Darstellung durch den seriellen Arduino-Monitor
017 Serial.begin (115200);
018 Serial.println ("Start");
019 }
020
```

Seite 42:

```
021 // die Schleife, die immer wieder durchlaufen wird
022 void loop() {
023 touchEin = touchRead(touchPinEin); // liest Touch ein
024 touchAus = touchRead(touchPinAus); // liest Touch aus
025 if (touchEin < 50) {
026 digitalWrite(LEDPin, HIGH); // schaltet die LED ein
027 Serial.println ("LED ein");
028 }
029 if (touchAus < 50) {
030 digitalWrite(LEDPin, LOW); // schaltet die LED aus
031 Serial.println ("LED aus");
032 }
033 delay(pause); // etwas warten
034 }
```

Seite 43:

```
001 /*
002 * Hall-Messung mit dem internen Sensor
003 */
004
005 void setup() {
006 Serial.begin(115200);
007 }
008
009 void loop() {
010 int messWert = 0;
011 messWert = hallRead();
012 Serial.print("Hall-Sensor-Messung: ");
013 Serial.println(messWert);
014 delay(1000);
023 }
```

Seite 44:

```
001 /*
002 * Temperaturmessung mit dem internen Sensor
003 */
004 #ifdef __cplusplus
005 extern "C" {
006 #endif
007 uint8_t temprature_sens_read();
008 #ifdef __cplusplus
009 }
010 #endif
011 uint8_t temprature_sens_read();
012
013 void setup() {
014 Serial.begin(115200);
015 }
016
```

Seite 45:

```
017 void loop() {
018 Serial.print("Temperatur: ");
019 // konvertiert Temperatur von Fahrenheit nach Celsius
020 Serial.print((temprature_sens_read() - 32) / 1.8);
021 Serial.println(" C");
022 delay(5000);
023 }
```

Seite 47:

```
001 /*
002 * ESP32-LEDC-Software-Fade
003 * Code angelehnt an Original-Arduino-Fade-Beispiel:
004 * https://www.arduino.cc/en/Tutorial/Fade
005 */
006 #define LED_PVM_KANAL 0 // der erste von 16 Kanälen wird belegt
007 #define LED_AUFLOESUNG 13 // 13-Bit-Auflösung
008 #define LED_BASE_FREQ 5000 // 5000 Hz als LEDC-Basis-Frequenz
009 #define LED_PIN 2
010
011 int helligkeit = 0; // Helligkeit der LED
012 int fadeWeite = 5; // Anzahl der Schritte pro Fade
013
014 // Simulation des Arduino-analogWrite
015 // Wert muss zwischen 0 und valueMax liegen
016 void ledcAnalogWrite(uint8_t channel, uint32_t value,
uint32_t valueMax = 255) {
```

Seite 48:

```
017 // Berechnung des EIN-Zyklus (duty), 8191 ist = 2 ^ 13 (Auflösung) - 1
018 uint32_t duty = (8191 / valueMax) * min(value, valueMax);
019 ledcWrite(channel, duty); // Wert an LED ausgeben
```

```
020 }
021
022 // Vorbereitungs- und Initialisierungsroutine
023 void setup() {
024 // Setup des PVM-Timers und Anbinden des Timers an einen LED-Pin
025 ledcSetup(LED_PVM_KANAL, LED_BASE_FREQ, LED_AUFLOESUNG);
026 ledcAttachPin(LED_PIN, LED_PVM_KANAL);
027 }
028
029 void loop() {
030 // Helligkeit an dem gewählten PVM-Kanal festlegen
031 ledcAnalogWrite(LED_PVM_KANAL, helligkeit);
032 // Helligkeit um die gewählte Schrittbreite ändern
033 helligkeit = helligkeit + fadeWeite;
034 // an den Endpunkten des Fade-Vorgangs die Richtung ändern
035 if (helligkeit <= 0 || helligkeit >= 255) {
036 fadeWeite = -fadeWeite;
037 }
038 delay(30);
039 }
```

Seite 50:

```
006 #define LED_PVM_KANAL_R 0 // Kanal 0 für Rot
#define LED_PVM_KANAL_G 1 // Kanal 1 für Grün
#define LED_PVM_KANAL_B 2 // Kanal 2 für Blau
007 #define LED_AUFLOESUNG 13 // 13-Bit-Auflösung
008 #define LED_BASE_FREQ 5000 // 5000 Hz als LEDC-Basis-Frequenz
009 #define LED_PIN_R 0
#define LED_PIN_G 2
#define LED_PIN_B 4
```

```
001 /*
002 * ESP32-LEDC-Software-RGB-Zustandsanzeige
003 */
004 #define LED_PVM_KANAL_R 0 // Kanal 0 für Rot
005 #define LED_PVM_KANAL_G 1 // Kanal 1 für Grün
006 #define LED_PVM_KANAL_B 2 // Kanal 2 für Blau
007 #define LED_AUFLOESUNG 13 // 13-Bit-Auflösung
008 #define LED_BASE_FREQ 5000 // 5000 Hz als LEDC-Basis-Frequenz
009 #define LED_PIN_R 0
010 #define LED_PIN_G 2
011 #define LED_PIN_B 4
012
013 // Simulation des Arduino-analogWrite
014 // Wert muss zwischen 0 und valueMax liegen
015 void ledcAnalogWrite(uint8_t channel, uint32_t value,
uint32_t valueMax = 255) {
016 // Berechnung des EIN-Zyklus (duty), 8191 ist = 2 ^ 13 (Auflösung) - 1
```

Seite 51:

```
017 uint32_t duty = (8191 / valueMax) * min(value, valueMax);
018 ledcWrite(channel, duty); // Wert an LED ausgeben
019 }
020
021 // Aufruf der analogWrite-Routine für jede RGB-Farbe
022 // void rgbAusgeben(uint32_t rot, uint32_t gruen, uint32_t blau) {
023 void rgbAusgeben(uint32_t rgbWerte[]) {
024 ledcAnalogWrite(LED_PVM_KANAL_R, rgbWerte[0]);
025 ledcAnalogWrite(LED_PVM_KANAL_G, rgbWerte[1]);
026 ledcAnalogWrite(LED_PVM_KANAL_B, rgbWerte[2]);
027 }
028
029 // Ermitteln der RGB-Kombination für den als Parameter übergebenen Zustand
030 void zustandAnzeigen (int zustand) {
031 uint32_t zustaende[6][3] = { 0, 255, 0, 150, 200, 0, 255, 255, 0,
032 220, 150, 0, 255, 0, 0, 255, 0, 100};
```

```
rgbAusgeben(zustaende[zustand]); //RGB-Werte übergeben
}

// Vorbereitungs- und Initialisierungsroutine
void setup() {
// Setup der PVM-Timer und Anbinden der Timer an einen LED-Pin
ledcSetup(LED_PVM_KANAL_R, LED_BASE_FREQ, LED_AUFLOESUNG);
ledcAttachPin(LED_PIN_R, LED_PVM_KANAL_R);
ledcSetup(LED_PVM_KANAL_G, LED_BASE_FREQ, LED_AUFLOESUNG);
ledcAttachPin(LED_PIN_G, LED_PVM_KANAL_G);
ledcSetup(LED_PVM_KANAL_B, LED_BASE_FREQ, LED_AUFLOESUNG);
ledcAttachPin(LED_PIN_B, LED_PVM_KANAL_B);
}

// die Schleife, die immer wieder durchlaufen wird
void loop() {
for (int i = 0; i < 6; i++) {
zustandAnzeigen (i);
delay(2000);
}
}
```

Seite 54:

```
/*
* Messen der Helligkeit mit einem Lichtsensor/Dämmerungsschalter
*/
#define LICHT_SENSOR_PIN 32 // Pin für den Lichtsensor

int tag = 3500; // Schwellenwert Tag
int nacht = 900; // Schwellenwert Nacht
int messWert = 0; // Messwert

// Lesen der Lichtmesswerte und Aktion ausführen (hier Serial Monitor)
void lesenHelligkeit() {
```

Seite 55:

```
messWert = analogRead(LICHT_SENSOR_PIN);
if (messWert < nacht) {
Serial.print("Nachtfunktion - Messwert: ");
Serial.println(messWert);
} else if (messWert > tag) {
Serial.print("Tagfunktion - Messwert: ");
Serial.println(messWert);
} else {
Serial.print("Normalbetrieb - Messwert: ");
Serial.println(messWert);
}
}

void setup() {
Serial.begin(115200);
Serial.println();
Serial.print("Start Helligkeit messen: ");
}

void loop() {
lesenHelligkeit();
delay(2000); // Pause um 2.000 ms
}
```

Seite 58:

```
/*
* Messen der Bodenfeuchtigkeit mit dem YL-69
*/
#define BODEN_SENSOR_A_PIN 33 // Pin für den Bodensensor analog
#define BODEN_SENSOR_D_PIN 17 // Pin für den Bodensensor digital
#define BODEN_SENSOR_VCC_PIN 34 // Pin für VCC-Bodensensor

```

```
int schwellenWert = 50; // anzupassender Schwellenwert

// Lesen der YL-69-Messwerte und Aktion ausführen (hier Serial Monitor)
void lesenYL69() {
int analogValue = analogRead(BODEN_SENSOR_A_PIN);
int digitalValue = digitalRead(BODEN_SENSOR_D_PIN);
int feuchtigkeit = map(analogValue, 0, 4095, 100, 0);
digitalWrite(BODEN_SENSOR_VCC_PIN, HIGH);
Serial.print("Feuchtigkeit: ");
```

Seite 59:

```
Serial.print(analogValue);
Serial.print(" ");
Serial.print(digitalValue);
Serial.print(" ");
Serial.print(feuchtigkeit);
Serial.print("%");

if (feuchtigkeit >= schwellenWert) {
Serial.print(" - Keine Bewässerung nötig");
} else {
Serial.print(" - Die Pflanzen brauchen Wasser");
}
if (digitalValue == 1) {
Serial.println(" - Alarm !Bewässerung dringend erforderlich");
} else {
Serial.println(" - alles in Ordnung ");
}
digitalWrite(BODEN_SENSOR_VCC_PIN, LOW);
}

void setup() {
pinMode(BODEN_SENSOR_VCC_PIN, OUTPUT);
pinMode(BODEN_SENSOR_D_PIN, INPUT);
digitalWrite(BODEN_SENSOR_D_PIN, LOW);
Serial.begin(115200);
}

void loop() {
lesenYL69();
delay(3000);
}
```

Seite 61:

```
/*
* Messen der Bodenfeuchte mit einem kapazitiven Sensor
*/
#define BODEN_SENSOR_A_PIN 33 // Pin für den Bodensensor analog

int schwellenWert = 400; // anzupassender Schwellenwert

// Lesen der Messwerte und Aktion ausführen (hier Serial Monitor)
void lesenMesswert() {
int analogValue = analogRead(BODEN_SENSOR_A_PIN);
int feuchtigkeit = map(analogValue, 0, 4095, 100, 0);
Serial.print("Feuchtigkeit: ");
Serial.print(analogValue);
Serial.print(" ");
Serial.print(feuchtigkeit);
Serial.print("%");

if (feuchtigkeit >= schwellenWert) {
Serial.print(" - Keine Bewässerung nötig");
} else {
Serial.print(" - Die Pflanzen brauchen Wasser");
}
}

void setup() {
Serial.begin(115200);
```

```
027 }
028
029 void loop() {
030 lesenMesswert();
031 delay(3000);
032 }
```

Seite 63:

```
001 /*
002 * Messen der Entfernung mit einem Ultraschallsensor
003 */
004 #define US_SENSOR_TRIG_PIN 33 // Trigger-Pin für den Ultraschallsensor
005 #define US_SENSOR_ECHO_PIN 34 // Eco-Pin für den Ultraschallsensor
006
007 float entfernung; // berechnete Entfernung
008 float dauer; // Zeitspanne des Echos
009
010 // Lesen der HCSR04-Messwerte und Aktion ausführen (hier Serial Monitor)
011 void lesenHCSR04() {
012 digitalWrite(US_SENSOR_TRIG_PIN, LOW); // definierten Zustand schaffen
013 delayMicroseconds(5); // Warten auf definierten PIN-Zustand
014 digitalWrite(US_SENSOR_TRIG_PIN, HIGH); // Trigger-Pin für
10 Mikrosekunden anstellen
015 delayMicroseconds(10);
016 digitalWrite(US_SENSOR_TRIG_PIN, LOW);
017 dauer = pulseIn(US_SENSOR_ECHO_PIN, HIGH); // misst die Pulslänge
in Mikrosekunden
018 entfernung = dauer * 0.034 / 2; // Entfernung berechnen
019 Serial.print("Entfernung: ");
020 Serial.println(entfernung);
021 }
022
023 void setup() {
024 pinMode(US_SENSOR_TRIG_PIN, OUTPUT);
025 pinMode(US_SENSOR_ECHO_PIN, INPUT);
026 Serial.begin(115200);
027 }
028
029 void loop() {
030 lesenHCSR04();
031 delay(2000);
047 }
```

Seite 65:

```
001 /*
002 * PIR-Bewegungsmelder
003 */
004 #define PIR_SENSOR_PIN 13 // Pin für den PIR
005
006 int statusPIR = 0; // PIR-Staus
007
008 // Lesen des PIR-Status und Aktion ausführen (hier Serial Monitor)
009 void lesenPIR() {
010 statusPIR = digitalRead(PIR_SENSOR_PIN);
011 if (statusPIR == 0) {
012 Serial.println("keine Bewegung ");
013 } else {
014 Serial.println("Bewegung erkannt");
015 }
016 }
017
018 void setup() {
019 Serial.begin(115200);
020 Serial.println();
021 pinMode(PIR_SENSOR_PIN, INPUT);
022 }
```

Seite 68:

```
/*
* Messen der Temperatur und der Luftfeuchtigkeit mit dem DHT22
*/
#include "DHTesp.h" // DHT22-Bibliothek für ESP32

#define DHT22_PIN 13 // Pin für den DHT22-Sensor
```

Seite 69:

```
DHTesp dht; // Instanz bilden

// Lesen der DHT22-Messwerte und Aktion ausführen (hier Serial Monitor)
void lesenDHT22() {
float temperatur = dht.getTemperature();
float luftFeuchte = dht.getHumidity();
if (isnan(temperatur) || isnan(luftFeuchte)) {
Serial.println("Fehler beim Lesen des DHT-Sensors!");
} else {
Serial.print("Temperatur: ");
Serial.print(temperatur);
Serial.print(" °C - Luftfeuchtigkeit: ");
Serial.print(luftFeuchte);
Serial.println(" %");
}
}

void setup() {
Serial.begin(115200);
Serial.println("Start DHT22-Sensor!");
dht.setup(DHT22_PIN);
}

void loop() {
lesenDHT22();
delay(2000);
}
```

Seite 73:

```
/**
* Senden von 433-MHz-Signalen
*/
```

Seite 74:

```
#include <RCSwitch.h> // Einbinden von rc-switch

#define TX_PIN 23 // Sender-Pin (GPIO23 - Pin D23)

RCSwitch mySwitch = RCSwitch();

void setup() {
Serial.begin(115200);
mySwitch.enableTransmit(TX_PIN); // Senden ermöglichen
}

void loop() {
if (Serial.available() > 0) { // Lesen Eingabe serieller Monitor
float codeSwitch = Serial.parseFloat();
mySwitch.send(codeSwitch, 24);
}
delay(1);
}
```

Seite 76:

```
/**
* Senden und Empfangen von 433-MHz-Signalen
*/
```

```
#include <RCSwitch.h> // Einbinden von rc-switch

#define TX_PIN 23 // Sender.Pin (GPIO23 - Pin D23)
#define RX_PIN 22 // Empfänger.Pin (GPIO22 - Pin D22)

RCSwitch mySwitch = RCSwitch();

void setup() {
Serial.begin(115200);
mySwitch.enableTransmit(TX_PIN); // Senden ermöglichen
mySwitch.enableReceive(RX_PIN); // Empfangen ermöglichen
}

```

Seite 77:

```
void loop() {
if (Serial.available() > 0) {
float codeSwitch = Serial.parseFloat();
mySwitch.send(codeSwitch, 24);
}
if (mySwitch.available()) {
int value = mySwitch.getReceivedValue();
if (value == 0) {
Serial.print("unbekanntes Protokoll");
} else {
Serial.print("empfangener Code: ");
Serial.print( mySwitch.getReceivedValue() );
Serial.print(" Bitlänge: ");
Serial.print( mySwitch.getReceivedBitlength() );
Serial.print(" Protocol: ");
Serial.println( mySwitch.getReceivedProtocol() );
}
mySwitch.resetAvailable();
}
}
```

Seite 81:

```
/**
* I2C OLED-Display SSD 1306 Ausgabe Hallo ESP32
*/
#include "SSD1306.h" // Header SSD 1306

SSD1306 display(0x3c, 21, 22); // SSD-1306-Instanz

int i = 0;

```

Seite 82:

```
// Zeichnen Grundfunktion
void zeichneHalloESP32() {
i++;
String hallo = "Hallo ESP32 " + String(i);
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 0, hallo);
}

// generelle Zeichnen-Funktion
void zeichnen() {
display.clear(); // Bildschirm löschen
zeichneHalloESP32();
display.display();
}

void setup() {
Serial.begin(115200);
Serial.println();
Serial.println();
display.init();
```

```
044 display.flipScreenVertically();
045 }
046
047 void loop() {
048 zeichnen();
049 delay(2000);
050 }
```

Seite 88:

```
005 #include "Flocke.h" // vorbereitete Bilder
010 // zeichnet Bilder
011 void zeichneBild() {
012 display.drawXbm(0, 0, flocke_width, flocke_height, flocke_bits);
013 //display.drawXbm(34, 14, WiFi_Logo_width, WiFi_Logo_height,
WiFi_Logo_bits);
014 }
020 display.clear(); // Bildschirm löschen
021 zeichneBild();
022 display.display();
```

Seite 89:

```
001 /**
002 * I2C-Scanner
003 */
004 #include <Wire.h>
005
006 #define SDA_PIN 21
007 #define SCL_PIN 22
008
009 void setup() {
010 Serial.begin(115200);
011 Wire.begin(SDA_PIN, SCL_PIN);
012 Serial.println("Start I2C-Scanner");
013 }
014
015 void loop() {
016 byte error, addresse;
017 int anz = 0;
018 for (addresse = 1; addresse < 127; addresse++ ) {
019 Wire.beginTransmission(addresse);
020 error = Wire.endTransmission();
021 if (error == 0) {
022 Serial.print("I2C-Gerät an Adresse 0x");
023 if (addresse<16)
024 Serial.print("0");
025 Serial.print(addresse,HEX);
026 Serial.println(" gefunden!");
027 anz++;
028 } else if (error==4) {
029 Serial.print("Unbekannter Fehler bei Adresse 0x");
030 if (addresse<16)
031 Serial.print("0");
032 Serial.println(addresse,HEX);
033 }
034 }
```

Seite 90:

```
035 if (anz == 0)
036 Serial.println("Keine I2C-Geräte gefunden");
037 else
038 Serial.println("Ende Scan");
039 delay(10000); // nächster Scanvorgang nach 10 Sekunden
0 }
```

Seite 94:

```
039 #define RST_PIN 22 // RST-Pin
```

```
040 #define SS_PIN 21 // SS- bzw. SDA-Pin
045 Serial.begin(115200);
```

Seite 95:

```
001 /*
002 * RFID-RC522-Auswertung der UID/Zugangskontrolle
003 */
004 #include <SPI.h>
005 #include <MFRC522.h> // für RFID-RC522
006
007 #define LED_PIN 2 // LED-Pin
008 #define RST_PIN 22 // RST-Pin
009 #define SDA_PIN 21 // SS- bzw. SDA-Pin
010
011 MFRC522 mfrc522(SDA_PIN, RST_PIN); // MFRC522-Instanz erstellen
012
013 void setup() {
014 Serial.begin(115200);
015 SPI.begin(); // SPI-Bus initialisieren
016 mfrc522.PCD_Init(); // RFID-RC522 initialisieren
017 Serial.println("Ein RFID-Tag an den Leser halten...");
018 Serial.println();
019 pinMode(LED_PIN, OUTPUT);
020 digitalWrite(LED_PIN, LOW);
021 }
022 void loop() {
023 String rfidTag= "";
```

Seite 96:

```
024 // Look for new cards
025 if (mfrc522.PICC_IsNewCardPresent() && // liegt eine neue Karte vor?
026 mfrc522.PICC_ReadCardSerial() ) { // ist die Auswahl einer Karte
erfolgreich?
027 Serial.print("RFID-Tag / Karten-ID :");
028 for (byte i = 0; i < mfrc522.uid.size; i++) { // Karten-ID behandeln
029 Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
// Monitorausgabe
030 Serial.print(mfrc522.uid.uidByte[i], HEX);
031 rfidTag.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
//Ausg.in String
032 rfidTag.concat(String(mfrc522.uid.uidByte[i], HEX));
033 }
034 Serial.println();
035 Serial.print("Aktion : ");
036 rfidTag.toUpperCase();
037 if (rfidTag.substring(1) == "99 70 13 2B") {
// hier den zuzul.RFID-Tag angeben
038 Serial.println("Zugang genehmigt \n");
039 digitalWrite(LED_PIN, HIGH);
040 delay(3000);
041 } else {
042 Serial.println("Zugang verweigert!\n");
043 digitalWrite(LED_PIN, LOW);
044 delay(3000);
045 }
046 }
047 }
```

Seite 97:

```
001 const uint8_t PERMITTED_UIDS[UID_COUNT][UID_LENGTH] = {
002 {0xB5, 0xA3, 0x2B, 0x1B },
{0x3A, 0x06, 0xE8, 0xAB } }
if (isPermitted(mfrc522.uid.uidByte)) {
```

Seite 102:

```
001 /*
```

```
002 * Timer-Interrupt
003 */
004 #define LED_PIN 2 // LED an GPIO2
005
006 volatile int ledStatus = 0;
007 volatile int interruptStatus;
008 int interruptZaehlerGesamt;
009 hw_timer_t *timer = NULL;
010 portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
011
012 // ISR
013 void IRAM_ATTR onTimer() {
014 portENTER_CRITICAL_ISR(&timerMux);
015 interruptStatus++;
016 portEXIT_CRITICAL_ISR(&timerMux);
017 if (ledStatus == 0) {
018 ledStatus = 1;
019 digitalWrite(LED_PIN, HIGH);
020 } else {
021 ledStatus = 0;
022 digitalWrite(LED_PIN, LOW);
023 }
024 }
025
026 void setup() {
027 Serial.begin(115200);
028 timer = timerBegin(0, 80, true);
029 timerAttachInterrupt(timer, &onTimer, true);
030 timerAlarmWrite(timer, 2000000, true); // 2 Sekunden
031 timerAlarmEnable(timer);
```

Seite 103:

```
032 pinMode(LED_PIN, OUTPUT);
033 }
034
035 void loop() {
036 if (interruptStatus > 0) {
037 portENTER_CRITICAL(&timerMux);
038 interruptStatus--;
039 portEXIT_CRITICAL(&timerMux);
040 interruptZaehlerGesamt++;
041 Serial.print("Timer-Interrupt: Gesamtzahl Interrupts: ");
042 Serial.println(interruptZaehlerGesamt);
043 }
044 }
```

Seite 105:

```
001 /*
002 * externer Interrupt
003 */
004 #define INTERR_PIN 4 // externer Interrupt an GPIO4
005
006 volatile int interruptStatus;
007 int interruptZaehlerGesamt;
008 portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
009
010 // ISR-Routine
011 void IRAM_ATTR isrHandler() {
012 portENTER_CRITICAL_ISR(&timerMux);
013 interruptStatus++;
014 portEXIT_CRITICAL_ISR(&timerMux);
015 }
016
017 void setup() {
018 Serial.begin(115200);
019 pinMode(INTERR_PIN, INPUT_PULLUP);
020 attachInterrupt(digitalPinToInterrupt(INTERR_PIN), isrHandler,
FALLING);
021 }
022
```

```
023 void loop() {
024 if (interruptStatus > 0) {
025 portENTER_CRITICAL(&timerMux);
026 interruptStatus--;
027 portEXIT_CRITICAL(&timerMux);
028 interruptZaehlerGesamt++;
029 Serial.print("Timer-Interrupt: Gesamtzahl Interrupts: ");
030 Serial.println(interruptZaehlerGesamt);
031 }
032 }
```

Seite 107:

```
006 volatile int interruptStatus, alteZeit = 0, debounceZeit = 100;
011 void IRAM_ATTR isrHandler() {
if ( (millis() - alteZeit) > debounceZeit) {
012 portENTER_CRITICAL_ISR(&timerMux);
013 interruptStatus++;
014 portEXIT_CRITICAL_ISR(&timerMux);
alteZeit = millis(); // letzte Zeit merken
}
015 }
```

Seite 110:

```
001 /*
002 * Bluetooth-Low-Energy-Scanner
003 */
004 #include <BLEDevice.h>
005 #include <BLEUtils.h>
006 #include <BLEScan.h>
007 #include <BLEAdvertisedDevice.h>
008
009 int scanTime = 30; // in Sekunden
010
011 class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks
{
012 void onResult(BLEAdvertisedDevice advertisedDevice) {
013 Serial.printf("Advertised Device: %s \n",
advertisedDevice.toString().c_str());
014 }
015 };
016
017 // Vorbereitungs- und Initialisierungsroutine
018 void setup() {
019 Serial.begin(115200);
020 Serial.println("Scanning...");
021 BLEDevice::init("");
022 BLEScan* pBLEScan = BLEDevice::getScan(); // neuen Scan erzeugen
023 pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCall
backs());
024 // aktiver Scan benötigt mehr Strom, erzielt aber schnellere Ergebnisse
025 pBLEScan->setActiveScan(true);
026 BLEScanResults foundDevices = pBLEScan->start(scanTime);
027 Serial.print("Anzahl gefundener Geräte: ");
028 Serial.println(foundDevices.getCount());
029 Serial.println("Scan erledigt!");
030 }
031
```

Seite 111:

```
032 // die Schleife, die immer wieder durchlaufen wird
033 void loop() {
034 // frei für weiteren Code, z. B. wiederholte Suche
035 delay(2000);
036 }
```

Seite 112:

```
001 /*
002 * Bluetooth-Daten von einer App empfangen, auswerten (LED) und
Nachricht senden
003 */
004 #include "BluetoothSerial.h" // Header Serial Bluetooth
005 #include <string.h> // Bearbeitung von Zeichenketten
006
007 #define LED_PIN 2 // GPIO2, Pin G2
008
009 BluetoothSerial ESP_BT; // Bluetooth-Instanz
010
011 void setup() {
012 Serial.begin(115200);
013 pinMode(LED_PIN, OUTPUT);
014 ESP_BT.begin("ESP32"); // Name des Bluetooth-Geräts
015 Serial.println("Bluetooth-Gerät bereit für das Pairing");
016 }
017
018 void loop() {
019 String zeile = ""; // für die eingegangenen Zeichen
020 if (ESP_BT.available()) { // gint es neue Nachrichten
via Bluetooth ?
021 while (ESP_BT.available()) {
022 char c = ESP_BT.read(); // Lesen der anstehenden Daten
023 zeile += c;
024 }
025 Serial.println("empfangene BLE-Daten: ");
026 Serial.println(zeile);
027 char *s1 = new char[zeile.length() + 1];
028 strcpy (s1, zeile.c_str() );
029 if (strcmp (s1, "ein") == 0) {
030 digitalWrite(LED_PIN, HIGH);
031 ESP_BT.println("LED eingeschaltet");
032 }
```

Seite 113:

```
033 if (strcmp (s1, "aus") == 0) {
034 digitalWrite(LED_PIN, LOW);
035 ESP_BT.println("LED ausgeschaltet");
036 }
037 }
038 delay(20);
039 }
```

Seite 121:

```
001 /*
002 * LED steuern mit Blynk
003 */
004 #include <WiFi.h>
005 #include <WiFiClient.h>
006 #include <BlynkSimpleEsp32.h>
007
008 #define BLYNK_PRINT Serial
009
010 const char* ssid = "meineSSID"; // Wert anpassen
011 const char* password = "meinPasswort"; // Wert anpassen
012 char auth []= "27e22e115cfe4631b17b41b83531b6bc";
// Blynk-App-auth-Token aus der E-Mail
013
```

Seite 122:

```
014 void setup() {
015 Serial.begin(115200);
016 Blynk.begin(auth, ssid, password);
017 }
018
019 void loop() {
020 Blynk.run();
```

```
021 }
```

Seite 123:

```
#include <RCSwitch.h> // Einbinden von rc-switch
#define TX_PIN 23 // Sender-Pin (GPIO23 - Pin D23)
RCSwitch mySwitch = RCSwitch();
BLYNK_WRITE(V1) { // Blynk-Widget sendet die Daten
int pinData = param.asInt();
if (pinData == 1) {
mySwitch.send(5260625, 24);
} else {
mySwitch.send(5260628, 24);
}
}
014 void setup() {
015 Serial.begin(115200);
016 Blynk.begin(auth, ssid, password);
017 mySwitch.enableTransmit(TX_PIN); // Senden ermöglichen
021 }
```

Seite 124:

```
001 /*
002 * DHT22 sendet Daten an Blynk
003 */
004 #include <WiFi.h>
005 #include <WiFiClient.h>
006 #include <BlynkSimpleEsp32.h>
007 #include "DHTesp.h"
008
009 #define BLYNK_PRINT Serial
010 #define DHT22_PIN 13
011
012 const char* ssid = "meineSSID"; // Wert anpassen
013 const char* password = "meinPasswort"; // Wert anpassen
014 char auth []= "27e22e115cfe4631b17b41b83531b6bc";
015
016 DHTesp dht;
017 BlynkTimer timer;
018
019 // sendet die Sensordaten
020 void sendSensor() {
021 float h = dht.getHumidity();
```

Seite 125:

```
022 float t = dht.getTemperature();
023 if (isnan(h) || isnan(t)) {
024 Serial.println("Failed to read from DHT sensor!");
025 return;
026 }
027 Blynk.virtualWrite(V5, h);
028 Blynk.virtualWrite(V6, t);
029 }
030
031 void setup() {
032 Serial.begin(115200);
033 Blynk.begin(auth, ssid, password);
034 dht.setup(DHT22_PIN);
035 timer.setInterval(1000L, sendSensor); // Timer-Intervall
in Millisekunden.
036 }
037
038 void loop() {
039 Blynk.run();
040 timer.run(); // Timer aufrufen
041 }
```

Seite 127:

```
001 /*
002 * Blynk sendet eine E-Mail
003 */
004 #include <SPI.h>
005 #include <WiFi.h>
006 #include <WiFiClient.h>
007 #include <BlynkSimpleEsp32.h>
008 #include "DHTesp.h"
009
010 #define BLYNK_PRINT Serial
011 //#define BLYNK_MAX_SENDBYTES 128 // für längere E-Mails
012 #define LED_PIN 2 // GPIO2, Pin G2
013 #define DHT22_PIN 13 // Pin für den DHT22-Sensor
014
015 const char* ssid = "meineSSID"; // Wert anpassen
016 const char* password = "meinPasswort"; // Wert anpassen
017 char auth []= "27e22e115cfe4631b17b41b83531b6bc";
018
019 DHTesp dht;
020 BlynkTimer timer;
021
022 volatile float h = 0;
023 volatile float t = 0;
024
025 // sendet die Sensordaten
026 void sendSensor() {
027 h = dht.getHumidity();
028 t = dht.getTemperature();
029 if (isnan(h) || isnan(t)) {
030 Serial.println("Failed to read from DHT sensor!");
031 return;
032 }
033 Blynk.virtualWrite(V5, h);
034 Blynk.virtualWrite(V6, t);
035 }
036
```

Seite 128:

```
037 void setup() {
038 Serial.begin(115200);
039 pinMode(LED_PIN, OUTPUT);
040 Blynk.begin(auth, ssid, password);
041 dht.setup(DHT22_PIN);
042 timer.setInterval(1000L, sendSensor);
// Timer-Intervall in Millisekunden.
043 }
044
045 void loop() {
046 Blynk.run();
047 timer.run();
048 if (t > 25) { // bei Überschreiten der max. Temperatur
049 digitalWrite(LED_PIN, HIGH);
050 Blynk.E-Mail("michael.schmitz@gmail.com", "Subject", "Test Blynk
Projekt E-Mail");
051 Delay (20000); // 20 Sekunden warten, bis heruntergekühlt
052 digitalWrite(LED_PIN, LOW);
053 }
054 }
```

Seite 131:

```
001 /*
002 * Messen der Temperatur und der Luftfeuchtigkeit mit dem DHT22
003 * Hinzufügen eines Zeitstempels und Ausgabe in einem Browser
004 */
005 #include <WiFi.h> // für Wi-Fi
006 #include <WiFiUdp.h>
007 #include "DHTesp.h" // DHT22-Bibliothek für ESP32
008 #include <NTPClient.h> // für NTP
009
011 #define DHT22_PIN 13 // Pin für den DHT22-Sensor
```

```
012 #define NTP_OFFSET 60 * 60 // in Sekunden
013 #define NTP_INTERVAL 60 * 1000 // in Millisekunden
014 #define NTP_ADDRESS "3.de.pool.ntp.org" // "ptbtime1.ptb.de"
015
016 const char* ssid = "meineSSID";
017 const char* password = "meinPasswort";
018
019 WiFiServer serverWiFi (80); // Port für Webserver
020 DHTesp dht; // Objekt bilden
021 WiFiUDP ntpUDP;
022 NTPClient timeClient(ntpUDP, NTP_ADDRESS, NTP_OFFSET, NTP_INTERVAL);
023
024 float temperatur;
025 float luftFeuchte;
026 String zeitFormattedNTP;
```

Seite 132:

```
70 // sendet eine Antwort an den Wi-Fi-Client
71 void wifiSend (WiFiClient client) {
72 lesenDHT22();
73 holenNTP();
74 client.println("HTTP/1.1 200 OK");
75 client.println("Content-type:text/html");
76 client.println();
77 // der Inhalt der HTTP-Antwort folgt auf den Header
78 client.println ("ESP32 <br>");
79 client.println("Zeit:" + zeitFormattedNTP + "<br>");
80 client.println("Temperatur:" + String(temperatur) + "<br>");
81 client.println("Luftfeuchtigkeit: " + String(luftFeuchte) + "<br>");
97 // die HTTP-Antwort endet mit einer weiteren Blank-Zeile
98 client.println(); }
99
```

Seite 133:

```
100 // bearbeitet eine Anfrage von dem Wi-Fi-Client
101 void wifiReceive (WiFiClient client) {
102 Serial.println("Neue Anfrage."); // Ausgabe einer Meldung
im seriellen Monitor
103 String currentLine = ""; // String-Variable f. eingehende Daten
104 int cnt = 0; // Zähler while client.connected
105 while (client.connected()) { // while-Schleife, solange
Clientverbindung
106 cnt++; // Schleifenzähler um 1 erhöhen
107 if (client.available()) { // Sind Bytes vom Client zu lesen?
108 char c = client.read(); // Lesen eines Bytes in Variable c
109 Serial.print(c); // Ausgabe des Bytes im seriellen Monitor
110 if (c == '\n') { // if the byte is a newline character
111 // das Ende der HTTP-Anfrage sind eine Blank-Zeile und zwei Newline-
Zeichen hintereinander
112 if (currentLine.length() == 0) { // Ende der HTTP-Anforderung
durch den Client
113 wifiSend(client);
114 break; // Beenden while client.connected Schleife
115 } else { // liegt ein Newline vor
116 currentLine = ""; // Variable currentLine löschen
117 }
118 } else if (c != '\r') { // alles andere als ein
Wagenrücklaufzeichen
119 currentLine += c; // Zeichen currentLine hinzufügen
120 }
```

Seite 134:

```
130 } // end if client.available
131 if (cnt > 10000) { // erzwingt disconnect fehlendes cli.avail
132 break;
133 }
134 } // end while client.connected
135 client.stop(); // Verbindung schließen
```

```
136 Serial.println("Client Disconnected.");
137 Serial.println();
138 }
139
```

```
150 // Lesen der DHT22-Messwerte
151 void lesenDHT22() {
152 temperatur = dht.getTemperature();
153 luftFeuchte = dht.getHumidity();
154 if (isnan(temperatur) || isnan(luftFeuchte)) {
155 Serial.println("Fehler beim Lesen des DHT-Sensors!");
156 } else {
157 Serial.print("Temperatur: ");
158 Serial.print(temperatur);
159 Serial.print(" °C - Luftfeuchtigkeit: ");
160 Serial.print(luftFeuchte);
161 Serial.println(" %");
162 }
163 }
164
165 // Holen der aktuellen Zeit
166 void holenNTP() {
167 timeClient.update(); // Aktualisieren des Zeitstempels
168 zeitFormattedNTP = timeClient.getFormattedTime(); // Ergebnis speichern
169 Serial.println(zeitFormattedNTP);
170 }
```

Seite 135:

```
180 void setup() {
181 Serial.begin(115200);
183 Serial.print("Verbindungsaufbau zu "); // Verbindungsaufbau zum
Wi-Fi-Netzwerk
184 Serial.println(ssid);
185 WiFi.begin(ssid, password);
186 while (WiFi.status() != WL_CONNECTED) {
187 delay(500);
188 Serial.print(".");
189 }
190 Serial.println(""); // Verbindung aufgebaut
191 Serial.println("WiFi connected.");
192 Serial.println("IP address: ");
193 Serial.println(WiFi.localIP());
194 Serial.println();
195 Serial.println("Start WiFi-Server!");
196 serverWiFi.begin();
197 Serial.println("Start DHT22-Sensor!");
198 dht.setup(DHT22_PIN);
199 Serial.println("Start NTP-Client!");
200 timeClient.begin();
201 }
```

```
210 void loop() {
211 WiFiClient client = serverWiFi.available();
// horcht auf Clientanfragen
```

Seite 136:

```
212 if (client) { // Fragt ein Client an?
213 wifiReceive (client); // Anfrage aufbereiten
214 }
215 delay(1);
216 }
```

Seite 137:

```
010 #define LED_PIN 2 // GPIO2, Pin G2
182 pinMode(LED_PIN, OUTPUT);
```

Seite 138:

```
82 client.print ("Klick <a href=\"/H\">hier</a>, um die LED an Pin 2
einzuschalten.<br>");
83 client.print ("Klick <a href=\"/L\">hier</a>, um die LED an Pin 2
auszuschalten.<br>");
```

```
121 // war die Clientanfrage "GET /H" or "GET /L" ?
122 if (currentLine.endsWith("GET /H")) {
123 digitalWrite(LED_PIN, HIGH); // GET /H schaltet die LED an
124 }
125 if (currentLine.endsWith("GET /L")) {
126 digitalWrite(LED_PIN, LOW); // GET /L schaltet die LED aus
127 }
```

Seite 140:

```
77 // der Inhalt der HTTP-Antwort folgt auf den Header
78 client.print ("<!DOCTYPE HTML><html><head>");
79 client.print ("<meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
80 client.print ("<title> ESP32-Schaltzentrale</title>");
81 client.println("<meta http-equiv=\"refresh\" content=\"10\"></head>");
82 client.print ("<body>");
83 aufbereitenKopf (client);
84 client.print ("<div style=\"font-size: 2.5rem;\">");
85 aufbereitenZeit (client);
86 aufbereitenTemperatur (client);
87 aufbereitenLuftF (client);
88 aufbereitenLED (client);
89 client.println("</div></body></html>");
97 // die HTTP-Antwort endet mit einer weiteren Blank-Zeile
```

```
030 // Überschrift aufbereiten
031 void aufbereitenKopf (WiFiClient client) {
032 client.print ("<div style=\"font-size: 3.5rem;\">");
033 client.print ("<p>ESP32 - Zentrale</p> <hr />");
034 client.print ("</div>");
035 }
036
037 // Zeitstempel aufbereiten
038 void aufbereitenZeit (WiFiClient client) {
039 client.print ("<p>Zeit: " + zeitFormattedNTP + "<br><br></p>");
040 }
041
042 // Temperatur aufbereiten
043 void aufbereitenTemperatur (WiFiClient client) {
044 float heatIndex = dht.computeHeatIndex(temperatur, luftFeuchte, false);
// heatIndex ist die gefühlte Temperatur
```

Seite 141:

```
045 if(temperatur>=25){
046 client.print ("<div style=\"color: #930000;\">");
047 } else if(temperatur<25 && temperatur>=5){
048 client.print ("<div style=\"color: #006601;\">");
049 } else if(temperatur<5){
050 client.print ("<div style=\"color: #009191;\">");
051 }
052 client.print ("Temperatur:        " +
String(temperatur) + "*C<br>");
053 client.print ("</div>");
054 }
055
056 // Luftfeuchtigkeit aufbereiten
057 void aufbereitenLuftF (WiFiClient client) {
058 client.print ("<p>Luftfeuchtigkeit: " + String(luftFeuchte) +
" %<br> </p>");
059 }
060
061 // LED-Auswahl aufbereiten
062 void aufbereitenLED (WiFiClient client) {
063 client.print ("<p>LED an Pin2       <a href=\"/
```

```
H\">hier</a> einschalten");
client.print (" und <a href=\"/L\">hier</a> ausschalten.<br> </p>");
}
```

Seite 144:

```
/*
* aktuelle Wetterdaten von OpenWeatherMap abrufen
*/

#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "meineSSID";
const char* password = "meinPasswort";
String stadtLand = "Berlin,de"; // Stadt und Länderkennung

const String url = "http://api.openweathermap.org/data/2.5/weather?q=";
const String key = "f90d72fb3fa99ef3cfdf6339e9b0xxxx";

void setup() {
Serial.begin(115200);
WiFi.begin(ssid, password);
```

Seite 145:

```
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.println(".");
}
Serial.println(""); // Verbindung aufgebaut
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void loop() {
if ((WiFi.status() == WL_CONNECTED)) { // besteht Verbindung noch?
HTTPClient http;
http.begin(url + stadtLand + "&units=metric&APPID=" + key);
// URL bestimmen
int httpCode = http.GET(); // Request absetzen
if (httpCode > 0) { // returning code ok? (id.R. 200)
String payload = http.getString();
Serial.println(payload);
} else {
Serial.println("Fehler beim http-Request");
}
http.end(); // Ressourcen freigeben
}
delay(60000); // 60 Sekunden warten
}
```

Seite 147:

```
/*
* aktuelle Wetterdaten von OpenWeatherMap abrufen
*/
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h> // JSON aufbereiten
#include <TimeLib.h> // Zeit aufbereiten

const char* ssid = "meineSSID";
const char* password = "meinPasswort";
String stadtLand = "Berlin,de"; // Ort, Stadt und Länderkennung

const String url = "http://api.openweathermap.org/data/2.5/
weather?q=";
const String key = "f90d72fb3fa99ef3cfdf6339e9b0xxxx";

```

```
016 // Holen der Wetterdaten über das Netz
017 int holenWetterDaten () {
018 float temperatur;
019 int sunset;
020 if ((WiFi.status() == WL_CONNECTED)) { // besteht Verbindung noch?
021 HTTPClient http;
022 http.begin(url + stadtLand + "&units=metric&APPID=" + key);
023 int httpCode = http.GET(); // Request absetzen
024 if (httpCode > 0) { // returning code ok? (id.R. 200)
025 String payload = http.getString();
026 StaticJsonBuffer<1000> jsonBuffer; // Speicher für JSON
bereitstellen
027 JsonObject& root = jsonBuffer.parseObject(payload);
// JSON aufbereiten
028 if (!root.success()) {
029 Serial.println("Fehler JSON-serialize");
030 sunset = 0;
031 } else {
032 temperatur = root ["main"] ["temp"];
033 Serial.print("Temperatur : ");
034 Serial.print(temperatur);
035 sunset = root ["sys"] ["sunset"];
```

Seite 148:

```
036 Serial.print(" Sonnenuntergang um ");
037 Serial.print(hour(sunset));
038 Serial.print(":");
039 Serial.print(minute(sunset));
040 Serial.print(" Uhr");
041 }
042 } else {
443 Serial.println("Fehler beim http-Request");
044 sunset = 0;
045 }
046 http.end(); // Ressourcen frei geben
047 }
048 return sunset;
049 }
050
001 void setup() {
052 Serial.begin(115200);
053 Serial.print("Connecting to ");
054 Serial.println(ssid);
055 WiFi.begin(ssid, password);
056 while (WiFi.status() != WL_CONNECTED) {
057 delay(500);
058 Serial.print(".");
059 }
060 // Verbindung aufgebaut
061 Serial.println("");
062 Serial.println("WiFi connected.");
063 Serial.println("IP address: ");
064 Serial.println(WiFi.localIP());
065 }
066
067 void loop() {
068 int sunset = holenWetterDaten(); // Holen der Wetterdaten
über das Netz
069 delay(60000); // 60 Sekunden warten
070 }
```

Seite 152:

```
010 #pragma region Globals
011 const char* ssid = "XXXXXXXX";
012 const char* password = "XXXXXXXX";
013 uint8_t connection_state = 0; // Connected to WIFI or not
016
017 String address[] = {"E-Mail1", "E-Mail2"};
018
73 Gsender *gsender = Gsender::Instance(); // Getting pointer to
```

```
class instance
74 String subject = "ESP32 Test!";
75 if (gsender->Subject(subject)->Send(2, address, "ESP32-Test-E-Mail!"))
76 {
```

```
020 const char* SMTP_SERVER = "smtp.gmail.com";
021 const char* E-MAILBASE64_LOGIN = "xxxxxxxxxxxxxxx";
022 const char* E-MAILBASE64_PASSWORD = "xxxxxxxxxxxxxxx";
023 const char* FROM = "xxxxxx@mail.com";
024 const char* _error = nullptr;
```

Seite 159:

```
001 /*
002 * ESP32 mit Cayenne verbinden
003 */
004 #define CAYENNE_PRINT Serial
005 #include <CayenneMQTTESP32.h> // Header Cayenne
006
007 char ssid[] = "meineSSID";
008 char wifiPassword[] = "meinPasswort";
009
010 // Cayenne-Authentifikationsangaben
011 char username[] = "8ef6b600-2f90-11e9-809d-0f8fe4c30267";
012 char password[] = "5f72ee148533c416a3aa194c311131507eecdd3a";
013 char clientID[] = "ce2fb450-32d0-11e9-a056-c5cffe7f75f9";
014
015 void setup() {
016 Serial.begin(115200);
017 Cayenne.begin(username, password, clientID, ssid, wifiPassword);
018 }
019
020 void loop() {
021 Cayenne.loop();
022 }
023
```

Seite 160:

```
024 // Defaultfunktion für das periodische Senden von Daten an Cayenne
025 CAYENNE_OUT_DEFAULT() {
026 Cayenne.virtualWrite(0, millis());
// Sendet Zeitangabe in Millisekunden über den virtuellen Kanal 0.
027 }
028
029 // Defaultfunktion für das Empfangen von Daten von Cayenne
030 CAYENNE_IN_DEFAULT() {
031 }
```

```
001 /*
002 * ESP32-Cayenne gibt DHT22-Daten aus und steuert eine LED
003 */
```

Seite 161:

```
004 #define CAYENNE_PRINT Serial
005 #include <CayenneMQTTESP32.h> // Header Cayenne
006 #include "DHTesp.h" // DHT22-Bibliothek für ESP32
007
008 #define LED_PIN 2 // GPIO2, Pin G2
009 #define DHT22_PIN 13 // Pin für den DHT22-Sensor
010
011 char ssid[] = "brunet33ext";
012 char wifiPassword[] = "friedens33";
013 char username[] = "8ef6b600-2f90-11e9-809d-0f8fe4c30267";
014 char password[] = "5f72ee148533c416a3aa194c311131507eecdd3a";
015 char clientID[] = "ce2fb450-32d0-11e9-a056-c5cffe7f75f9";
016
017 DHTesp dht; // Objekt bilden
018 float temperatur;
```

```
019 float luftFeuchte;
020 int ledStatus = 0;
021
022 // Lesen der DHT22-Messwerte
023 void lesenDHT22() {
024 temperatur = dht.getTemperature();
025 luftFeuchte = dht.getHumidity();
026 if (isnan(temperatur) || isnan(luftFeuchte)) {
027 Serial.println("Fehler beim Lesen des DHT-Sensors!");
028 }
029 }
030
031 CAYENNE_OUT_DEFAULT() {
032 lesenDHT22();
033 Cayenne.celsiusWrite(0, temperatur);
034 Cayenne.virtualWrite(1, luftFeuchte,TYPE_RELATIVE_HUMIDITY,UNIT_PERCENT);
035 Cayenne.digitalSensorWrite(2, ledStatus);
036 }
037
038 CAYENNE_IN_DEFAULT() {
039 if (request.channel == 3) {
040 ledStatus = getValue.asInt();
041 digitalWrite(LED_PIN,ledStatus);
042 }
043 }
044
```

Seite 162:

```
045 void setup() {
046 Serial.begin(115200);
047 pinMode(LED_PIN, OUTPUT);
048 digitalWrite(LED_PIN, LOW);
049 Serial.println("Start DHT22-Sensor!");
050 dht.setup(DHT22_PIN);
051 Cayenne.begin(username, password, clientID, ssid, wifiPassword);
052 }
053
054 void loop() {
055 Cayenne.loop();
056 }
```

Seite 169:

```
001 /*
002 * Deep-Sleep mit Touch-Wake-up
003 */
004 #define TOUCH_PIN0 T0 // Touch 0 an GPIO4
005 #define TOUCH_PIN3 T3 // Touch 3 an GPIO15
006
007 RTC_DATA_ATTR int bootCount = 0; // Zähler in den RTC-Speicher
legen
008 int schwellenWert = 40; // Schwellenwert
010
020 // Ausgabe der Weckquelle
021 void printWeckQuelle(){
022 esp_sleep_wakeup_cause_t weckQuelle = esp_sleep_get_wakeup_cause();
023 switch(weckQuelle) {
024 case ESP_SLEEP_WAKEUP_EXT0 :
025 Serial.println("Weckquelle ist ein externes RTC_IO-Signal");
026 break;
027 case ESP_SLEEP_WAKEUP_EXT1 :
028 Serial.println("Weckquelle ist ein externes RTC_CNTL-Signal");
029 break;
030 case ESP_SLEEP_WAKEUP_TIMER :
031 Serial.println("Weckquelle ist ein Timer");
032 break;
033 case ESP_SLEEP_WAKEUP_TOUCHPAD :
034 Serial.println("Weckquelle ist ein Touch-Sensor");
035 break;
036 case ESP_SLEEP_WAKEUP_ULP :
037 Serial.println("Weckquelle ist ein ULP-Programm");
```

```
038 break;
```

Seite 170:

```
039 default :
040 Serial.printf("Weckquelle undefiniert: %d\n", weckQuelle);
041 break;
042 }
043 }
044
050 // Ausgeben des Touch-Sensors
051 void printWeckQuelleTouchSensor(){
052 touch_pad_t touchPinWakeUp = esp_sleep_get_touchpad_wakeup_status();
053 switch(touchPinWakeUp)
054 {
055 case 0 : Serial.println("Auslöser Touch-Sensor 0 an GPIO4"); break;
056 case 3 : Serial.println("Auslöser Touch-Sensor 3 an GPIO15"); break;
057 default : Serial.println("auslösender Touch-Sensor unbekannt"); break;
058 }
059 }
060
070 void callback(){ // Platzhalter für weitere Befehle
071 }
072
073 void setup(){
074 Serial.begin(115200);
075 delay(1000); // etwas Zeit; wichtig - nicht löschen!
076 ++bootCount; // wird bei jedem Boot um 1 hochgezählt
077 Serial.println("Boot-Nummer: " + String(bootCount));
078 printWeckQuelle(); // Weckquelle ausgeben
079 printWeckQuelleTouchSensor(); // Weckquelle Touch-Sensor ausgeben
080 touchAttachInterrupt(T0, callback, schwellenWert);
// T0 als Sensor bekannt machen
081 touchAttachInterrupt(T3, callback, schwellenWert);
// T1 als Sensor bekannt machen
082 esp_sleep_enable_touchpad_wakeup();
// Touch-Sensoren als Weckquelle festlegen
083 Serial.println("Start Deep-Sleep-Modus");
084 esp_deep_sleep_start(); // in den Deep-Sleep-Modus versetzen
085 Serial.println("Diese Stelle sollte nie erreicht werden");
086 }
087
088 void loop(){ // wird nie ausgeführt
089 }
```

Seite 171:

```
001 /*
002 * Deep-Sleep mit Timer-Wake-up
003 */
004 #define TIME_TO_SLEEP 5 * 1000000
// "Schlafzeit" in Mikrosekunden => 5 Sekunden
006
```

Seite 172:

```
007 RTC_DATA_ATTR int bootCount = 0; // Zähler in den RTC-Speicher legen
010
020 // Ausgabe der Weckquelle
82 esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP);
// Timer als Weckquelle festlegen
84 Serial.println("Start Deep-Sleep-Modus");
```

```
082 esp_sleep_enable_ext0_wakeup(GPIO_NUM_33, 1);
```

Seite 173:

```
004 #define BUTTON_PIN_BITMASK 0X3000000
082 esp_sleep_enable_ext0_wakeup(BUTTON_PIN_BITMASK,
ESP_EXT1_WAKEUP_ANY_HIGH);
```

Seite 175:

```
const char* ssid = "ssid";
const char* password = "password";
```

Seite 178:

```
/*
 * Messen der Temperatur und der Luftfeuchtigkeit mit dem DHT22
 * Hinzufügen eines Zeitstempels und Ausgabe in einem Browser
 * LED über eine Browsereingabe steuern
 */
#include <WiFi.h> // für Wi-Fi
#include <WiFiUdp.h>
#include "DHTesp.h" // DHT22-Bibliothek für ESP32
#include <NTPClient.h> // für NTP

#define LED_PIN 2 // GPIO2, Pin G2
#define DHT22_PIN 13 // Pin für den DHT22-Sensor
#define NTP_OFFSET 2 * 60 * 60 // in Sekunden
#define NTP_INTERVAL 60 * 1000 // in Millisekunden
#define NTP_ADDRESS "3.de.pool.ntp.org" // "ptbtime1.ptb.de"

const char* ssid = "brunet33ext";
const char* password = "friedens33";

WiFiServer serverWiFi (80); // Port für Webserver
DHTesp dht; // Objekt bilden
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_ADDRESS, NTP_OFFSET, NTP_INTERVAL);

float temperatur;
float luftFeuchte;
String zeitFormattedNTP;

// Überschrift aufbereiten
void aufbereitenKopf (WiFiClient client) {
client.print ("<div style=\"font-size: 3.5rem;\">");
client.print ("<p>ESP32 - Zentrale</p> <hr />");
client.print ("</div>");
}

```

Seite 179:

```
// Zeitstempel aufbereiten
void aufbereitenZeit (WiFiClient client) {
client.print ("<p>Zeit: " + zeitFormattedNTP + "<br><br></p>");
}

// Temperatur aufbereiten
void aufbereitenTemperatur (WiFiClient client) {
float heatIndex = dht.computeHeatIndex(temperatur, luftFeuchte,
false);
// heatIndex ist die gefühlte Temperatur
if(temperatur>=25){
client.print ("<div style=\"color: #930000;\">");
} else if(temperatur<25 && temperatur>=5){
client.print ("<div style=\"color: #006601;\">");
} else if(temperatur<5){
client.print ("<div style=\"color: #009191;\">");
}
client.print ("Temperatur:        " +
String(temperatur) + "*C<br>");
client.print ("</div>");
}

// Luftfeuchtigkeit aufbereiten
void aufbereitenLuftF (WiFiClient client) {
client.print ("<p>Luftfeuchtigkeit: " + String(luftFeuchte) +
" %<br> </p>");
```

```
059 }
060
061 // LED-Auswahl aufbereiten
062 void aufbereitenLED (WiFiClient client) {
063 client.print ("<p>LED an Pin2       <a href=\"/
H\">hier</a> einschalten");
064 client.print (" und <a href=\"/L\">hier</a> ausschalten.<br> </p>");
065 }
066
```

Seite 180:

```
067 // sendet eine Antwort an den Wi-Fi-Client
068 void wifiSend (WiFiClient client) {
069 lesenDHT22();
070 holenNTP();
071 client.println("HTTP/1.1 200 OK");
072 client.println("Content-type:text/html");
073 client.println();
074 // der Inhalt der HTTP-Antwort folgt auf den Header
075 client.print ("<!DOCTYPE HTML><html><head>");
076 client.print ("<meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
077 client.print ("<title> ESP32-Schaltzentrale</title>");
078 client.println("<meta http-equiv=\"refresh\" content=\"10\"></head>");
079 client.print ("<body>");
080 aufbereitenKopf (client);
081 client.print ("<div style=\"font-size: 2.5rem;\">");
082 aufbereitenZeit (client);
083 aufbereitenTemperatur (client);
084 aufbereitenLuftF (client);
085 aufbereitenLED (client);
086 client.println("</div></body></html>");
087 // die HTTP-Antwort endet mit einer weiteren Blank-Zeile
088 client.println();
089 }
090
091 // bearbeitet eine Anfrage von dem Wi-Fi-Client
092 void wifiReceive (WiFiClient client) {
093 Serial.println("Neue Anfrage.");
// Ausgabe einer Meldung im seriellen Monitor
094 String currentLine = ""; // String-Variable f. eingehende Daten
095 int cnt = 0; // Zähler while client.connected
096 while (client.connected()) { // while-Schleife, solange
Clientverbindung
097 cnt++; // Schleifenzähler um 1 erhöhen
098 if (client.available()) { // Sind Bytes vom Client zu lesen?
099 char c = client.read(); // Lesen eines Bytes in Variable c
100 Serial.print(c); // Ausgabe des Bytes im seriellen Monitor
101 if (c == '\n') { // Ist das Zeichen ein Newline-Zeichen ?
102 // das Ende der HTTP-Anfrage ist eine Blank-Zeile und
zwei Newlin-Zeichen hintereinander
103 if (currentLine.length() == 0) {
// Ende der HTTP-Anforderung durch den Client
```

Seite 181:

```
104 wifiSend(client);
105 break; // Beenden der while client.connected-Schleife
106 } else { // liegt ein Newline vor
107 currentLine = ""; // Variable currentLine löschen
108 }
109 } else if (c != '\r') { // alles andere als ein Wagenrücklaufzeichen
110 currentLine += c; // Zeichen der Variablen currentLine hinzufügen
111 }
112 // war die Clientanfrage "GET /H" or "GET /L":
113 if (currentLine.endsWith("GET /H")) {
114 digitalWrite(LED_PIN, HIGH); // GET /H schaltet die LED an
115 }
116 if (currentLine.endsWith("GET /L")) {
117 digitalWrite(LED_PIN, LOW); // GET /L schaltet die LED aus
118 }
```

```
} // end if client.available
if (cnt > 10000) { // erzwingt disconnect fehlendes client.available
break;
}
} // end while client.connected
client.stop(); // Verbindung schließen
Serial.println("Client Disconnected.");
Serial.println();
}

// Lesen der DHT22-Messwerte
void lesenDHT22() {
temperatur = dht.getTemperature();
luftFeuchte = dht.getHumidity();
if (isnan(temperatur) || isnan(luftFeuchte)) {
Serial.println("Fehler beim Lesen des DHT-Sensors!");
} else {
Serial.print("Temperatur: ");
Serial.print(temperatur);
Serial.print(" °C - Luftfeuchtigkeit: ");
Serial.print(luftFeuchte);
Serial.println(" %");
}
}
```

Seite 182:

```
// Holen der aktuellen Zeit
void holenNTP() {
timeClient.update(); // Aktualisieren des Zeitstempels
zeitFormattedNTP = timeClient.getFormattedTime();
// Ergebnis speichern
// Serial.println(timeClient.getFormattedTime());
Serial.println(zeitFormattedNTP);
// unsigned long zeitNTP = timeClient.getEpochTime();
// Serial.println(zeitNTP);
}

void setup() {
Serial.begin(115200);
pinMode(LED_PIN, OUTPUT);
Serial.print("Verbindungsaufbau zu ");
// Verbindungsaufbau zum Wi-Fi-Netzwerk
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println(""); // Verbindung aufgebaut
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
Serial.println();
Serial.println("Start WiFi-Server!");
serverWiFi.begin();
Serial.println("Start DHT22-Sensor!");
dht.setup(DHT22_PIN);
Serial.println("Start NTP-Client!");
timeClient.begin();
}

void loop() {
WiFiClient client = serverWii.available();
// horcht auf Clientanfragen
if (client) { // Fragt ein Client an?
wifiReceive (client); // Anfrage aufbereiten
}
```

Seite 183:

```
182 delay(1);
183 }
```

```
001 /*
002 * Messen der Temperatur und der Luftfeuchtigkeit mit dem DHT22
003 * Weitergabe der Daten mit Zeitstempel an MQTT-Broker Mosquitto
004 * Schalten einer LED über MQTT
005 */
006 #include <WiFi.h> // für Wi-Fi
007 #include <WiFiUdp.h>
008 #include "DHTesp.h" // DHT22-Bibliothek für ESP32
009 #include <NTPClient.h> // für NTP
010 #include <PubSubClient.h> // für MQTT
011 #include <ArduinoJson.h> // JSON aufbereiten
012 #include <string.h> // Bearbeitung von Zeichenketten
013
014 #define LED_PIN 2 // Pin für die LED
015 #define DHT22_PIN 13 // Pin für den DHT22-Sensor
016 #define NTP_OFFSET 2 * 60 * 60 // in Sekunden
017 #define NTP_INTERVAL 60 * 1000 // in Millisekunden
018 #define NTP_ADDRESS "3.de.pool.ntp.org"
019
020 const char* ssid = "meineSSID";
021 const char* password = "meinPasswort";
022 const char* mqttServer = "192.168.2.210";
023 const char* mqttUser = "esp32";
024 const char* mqttPassword = "raspberry";
025
026 DHTesp dht; // Objekte bilden
027 WiFiClient wifiClient;
028 WiFiUDP ntpUDP;
029 NTPClient timeClient(ntpUDP, NTP_ADDRESS, NTP_OFFSET, NTP_INTERVAL);
030 PubSubClient mqttClient(wifiClient);
031
032 float temperatur;
033 float luftFeuchte;
```

Seite 184:

```
034 unsigned long zeitNTP;
035 unsigned long zeitNTPalt = 0;
036
037 // Lesen der DHT22-Messwerte
038 void lesenDHT22() {
039 temperatur = dht.getTemperature();
040 luftFeuchte = dht.getHumidity();
041 if (isnan(temperatur) || isnan(luftFeuchte)) {
042 temperatur = 0;
043 luftFeuchte = 0;
044 }
045 }
046
047 // Holen der aktuellen Zeit
048 void holenNTP() {
049 timeClient.update(); // Aktualisieren des Zeitstempels
050 zeitNTP = timeClient.getEpochTime();
051 }
052
053 // sendet MQTT-Daten an Mosquitto
054 void mqttSend () {
055 char mqttDaten [80], zwStr[11];
056 holenNTP();
057 if (zeitNTP > zeitNTPalt + 100) { // alle 100 Sekunden
058 zeitNTPalt = zeitNTP;
059 lesenDHT22();
060 strcpy (mqttDaten, "{\"zeit\" : \"");
061 dtostrf(zeitNTP, 10, 0, zwStr);
062 strcat (mqttDaten, zwStr);
063 strcat (mqttDaten, "\", \"temperatur\" : \"");
064 dtostrf(temperatur, 3, 1, zwStr);
065 strcat (mqttDaten, zwStr);
066 strcat (mqttDaten, "\", \"luftFeuchte\" : \"");
```

```
dtostrf(luftFeuchte, 4, 2, zwStr);
strcat (mqttDaten, zwStr);
strcat (mqttDaten, "\"}");

// Alternative
DynamicJsonBuffer jsonBuffer;
String input = "{\"zeit\":123456789,\"temperatur\":20.00,
\"luftFeuchte\":55.00}";
```

Seite 185:

```
JsonObject& root = jsonBuffer.parseObject(input);
root[String("zeit")] = zeitNTP;
root[String("temperatur")] = temperatur;
root[String("luftFeuchte")] = luftFeuchte;
String mqttString;
root.printTo(mqttString);
mqttString.toCharArray(mqttDaten, 80);

Serial.print("Daten senden; ");
Serial.println(mqttDaten);
mqttClient.publish("esp32/DHT22", mqttDaten);
}
}

// MQTT-Daten empfangen
void callback(char* topic, byte* payload, unsigned int length) {
String strTemp;
int i;
Serial.print("empfangene topic: ");
Serial.print(topic);
Serial.print(" payload: ");
for (i = 0; i < length; i++) { // Sichern der Nachricht in Variable
Serial.print((char)payload[i]);
strTemp += (char)payload[i];
}
Serial.println();

if(strcmp (topic, "esp32/LED") == 0) {
DynamicJsonBuffer jsonBuffer; // Speicher für JSON bereitstellen
JsonObject& root = jsonBuffer.parseObject(strTemp);
// JSON aufbereiten
if (!root.success()) {
Serial.println("Fehler JSON-serialize");
} else {
const char* ledStatus = root["LED"];
Serial.print("ledStatus : ");
Serial.println(ledStatus);
if (strcmp (ledStatus, "on") == 0) {
digitalWrite(LED_PIN, HIGH);
```

Seite 186:

```
} else if (strcmp (ledStatus, "off") == 0) {
digitalWrite(LED_PIN, LOW);
}
}
} else { // weitere Topics
}
}

// Verbindung zum MQTT-Broker aufbauen
void mqttConnect() {
Serial.println("Versuch, MQTT-Verbindung aufzubauen ");
while (!mqttClient.connected()) { // so lange, bis Verbindung steht
Serial.print(".");
if(mqttClient.connect("ESP32Client", mqttUser, mqttPassword )) {
Serial.println();
Serial.println("MQTT verbunden");
} else {
Serial.print("Fehler Verbindungsaufbau, rc=");
Serial.print(mqttClient.state());
```

```
131 Serial.println(" erneuter Versuch in 2 Sekunden");
132 delay(2000); // 2 Sekunden vor erneutem Versuch warten
133 }
134 }
135 mqttClient.subscribe("esp32/+");
136 }
137
138 void setup() {
139 Serial.begin(115200);
140 Serial.print("Verbindungsaufbau zu "); // Verbindungsaufbau zum
Wi-Fi-Netzwerk
141 Serial.println(ssid);
142 WiFi.begin(ssid, password);
143 while (WiFi.status() != WL_CONNECTED) {
144 delay(500);
145 Serial.print(".");
146 }
147 Serial.println("");
148 Serial.println("WiFi verbunden");
149 Serial.println("IP address: ");
150 Serial.println(WiFi.localIP());
151 Serial.println("Start DHT22-Sensor!");
```

Seite 187:

```
152 dht.setup(DHT22_PIN);
153 Serial.println("Start NTP-Client!");
154 timeClient.begin();
155 mqttClient.setServer(mqttServer, 1883);
156 mqttClient.setCallback(callback);
157 mqttConnect();
158 Serial.println();
159 pinMode(LED_PIN, OUTPUT);
160 digitalWrite(LED_PIN, LOW);
161 }
162
163 void loop() {
164 if (!mqttClient.connected()) {
165 mqttConnect();
166 }
167 mqttClient.loop();
168 mqttSend ();
169 delay(50); // 0,05 Sekunden warten
170 }
```

```
001 /*
002 * Deep-Sleep mit Timer-Wake-up
003 */
004 #define TIME_TO_SLEEP 5 * 1000000
// "Schlafzeit" in Mikrosekunden => 5 Sekunden
005
006 RTC_DATA_ATTR int bootCount = 0; // Zähler in den RTC-Speicher legen
007
008 // Ausgabe der Weckquelle
009 void printWeckQuelle(){
010 esp_sleep_wakeup_cause_t weckQuelle = esp_sleep_get_wakeup_cause();
011 switch(weckQuelle) {
012 case ESP_SLEEP_WAKEUP_EXT0 :
013 Serial.println("Weckquelle ist ein externes RTC_IO-Signal");
014 break;
015 case ESP_SLEEP_WAKEUP_EXT1 :
016 Serial.println("Weckquelle ist ein externes RTC_CNTL-Signal");
017 break;
```

Seite 188:

```
018 case ESP_SLEEP_WAKEUP_TIMER :
019 Serial.println("Weckquelle ist ein Timer");
020 break;
021 case ESP_SLEEP_WAKEUP_TOUCHPAD :
022 Serial.println("Weckquelle ist ein Touch-Sensor");
```

```
break;
case ESP_SLEEP_WAKEUP_ULP :
Serial.println("Weckquelle ist ein ULP-Programm");
break;
default :
Serial.printf("Weckquelle undefiniert: %d\n", weckQuelle);
break;
}
}

void callback(){ // Platzhalter für weitere Befehle
}

void setup(){
Serial.begin(115200);
delay(1000); // etwas Zeit; wichtig - nicht löschen!
++bootCount; // wird bei jedem Bootvorgang um 1 hochgezählt
Serial.println("Boot-Nummer: " + String(bootCount));
printWeckQuelle(); // Weckquelle ausgeben
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP);
Serial.println("Start Deep-Sleep-Modus");
esp_deep_sleep_start(); // in den Deep-Sleep-Modus versetzen
Serial.println("Diese Stelle sollte nie erreicht werden");
}

void loop(){ // wird nie ausgeführt
}
```