



Thomas
Brühlmann
5. Auflage

Arduino Praxiseinstieg

Behandelt Arduino UNO R4 und R3



UNO R4
MINIMA

komplett
in Farbe

Zusatz zu Kapitel 10

Wifi mit ESP8266 (nur Arduino UNO R3)

Ethernet-Anwendungen mit dem Ethernet Shield benötigen immer eine physische Verbindung zum Netzwerk via Ethernet-Kabel. Dadurch ist man immer darauf angewiesen, dass ein Ethernet-Switch oder Router in der Nähe ist.

Bei Anwendungen, die ohne direkten Zugang zu einem physischen Ethernet-Anschluss auskommen sollen, empfiehlt sich der Einsatz einer drahtlosen Ethernet-Verbindung.

In der heutigen Zeit sind in den meisten Haushalten bereits drahtlose Zugangspunkte vorhanden, über die man seine Rechner, Tablets oder Smartphones drahtlos mit dem Internet verbindet.

Für drahtlose Ethernet-Anwendungen mit dem Arduino gab es auf dem Markt verschiedene Shields und Boards für den WiFi-Einsatz. Die meisten Module sind aber wieder aus dem Handel verschwunden, da vor einigen Jahren ein sehr kompaktes WiFi-Modul auf den Markt gekommen ist, das WiFi-Transceiver-Modul ESP8266 des chinesischen Herstellers Espressif Systems.

ESP8266-WiFi-Transceiver

Mit den kompakten Abmessungen von 15x25 mm ist das WiFi-Modul ESP8266 ESP-01 momentan eines der kleinsten Ethernet-Module auf dem Markt. Neben der Elektronik ist auf der Leiterplatte eine Chipantenne vorhanden.

Kurz nach Vorstellung des Models ESP8266 ESP-01 hat der Hersteller auch noch zwei weitere Typen, ESP-02 und ESP-03, auf den Markt gebracht. Die Modelle ESP-02 und ESP-03 werden mittels Lötpad auf die Leiterplatte gelötet.

Alle nachfolgend beschriebenen Informationen und Beispiele beziehen sich auf das Model ESP-01.

Der ESP8266-Transceiver unterstützt 802.11-b/g/n-Netzwerke und kann als Station (Client) oder als Access-Point genutzt werden.

Die Verbindung des ESP8266 ESP-01 mit dem Arduino-Board erfolgt über eine 10-polige Stiftleiste (Abbildung 1).

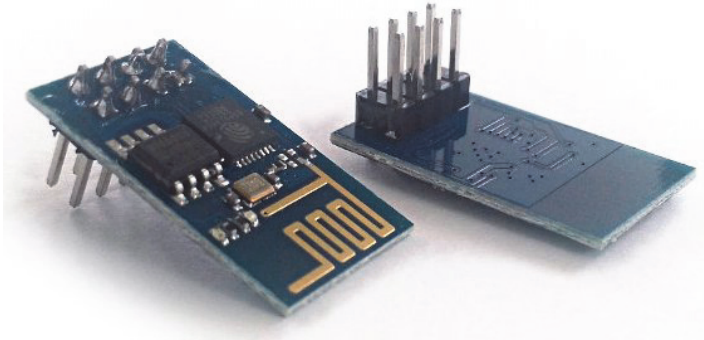


Abb. 1: WiFi-Modul ESP8266 ESP-01

Das Modul wird über den seriellen Bus angesteuert und benötigt keine externe Beschaltung. Auf dem Board selbst ist eine Chipantenne in Form einer mäanderförmigen Leiterbahn aufgebracht.

Anschlussbelegung und Verbindung zum Arduino-Board

In Abbildung 2 und Tabelle 1 sind die Anschlussbelegung und die Pin-Bezeichnungen des ESP8266 ESP-01 dargestellt.

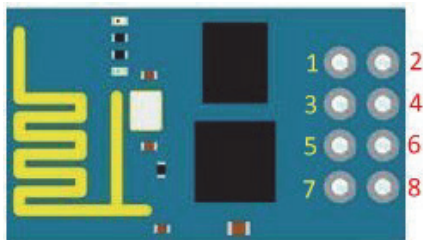


Abb. 2: Anschlussbelegung ESP8266 ESP-01

Pin-Nummer	Bezeichnung	Beschreibung
1	RX	Daten empfangen (3,3 V Pegel)
2	VCC	Versorgungsspannung, max. 3,6 V
3	GPIO0	Digitaler Eingang/Ausgang 0
4	RST	Reset (LOW = Reset aktiv)
5	GPIO2	Digitaler Eingang/Ausgang 2
6	CH_PD	Chip Power Down (LOW = aktiv)
7	GND	Ground
8	TX	Daten senden (3,3 V Pegel)

Tabelle 1: Anschlussbelegung ESP8266 ESP-01

Die Kommunikation mit dem Arduino-Board erfolgt über die seriellen Signale RX und TX. Dazu können die seriellen Pins 0 und 1 des Arduino verwendet werden.

Die Versorgung des Transceiver-Moduls kann über die 3,3-V-Versorgung des Arduino-Boards erfolgen. Zu beachten ist, dass die Versorgungsspannung des verwendeten Boards in der Startphase mindestens 300 mA Strom liefern muss.

Da das ESP8266-Modul nur mit 3,3-V-Signalen angesteuert werden darf, eignet sich ein Arduino UNO nicht optimal für diese Anwendung. Der Autor empfiehlt die Verwendung eines Arduino-Clones mit Signallevel-Umschalter, beispielsweise mit einem Seeedstudio von Seeedstudio (<https://www.seeedstudio.com/Seeeduno-V4-2-p-2517.html>). Aus der Arduino-Reihe eignet sich die 3,3-V-Variante des Arduino Pro Mini. Im Internet werden aber trotz der 3,3-V-Pegel des ESP8266 mit dem Arduino UNO etliche Beispiele publiziert, in denen das WiFi-Modul direkt vom Arduino-Board angesteuert wird. Diese Lösungen ohne Signalpegel-Anpassung sind aber nicht zu empfehlen.

Abbildung 3 zeigt die Verbindungen des ESP8266 zum Arduino-Clone Seeeduno. Bei der Verdrahtung ist zu beachten, dass die Versorgungsspannung des ESP8266 mit dem 3,3-V-Anschluss des Boards verbunden ist.

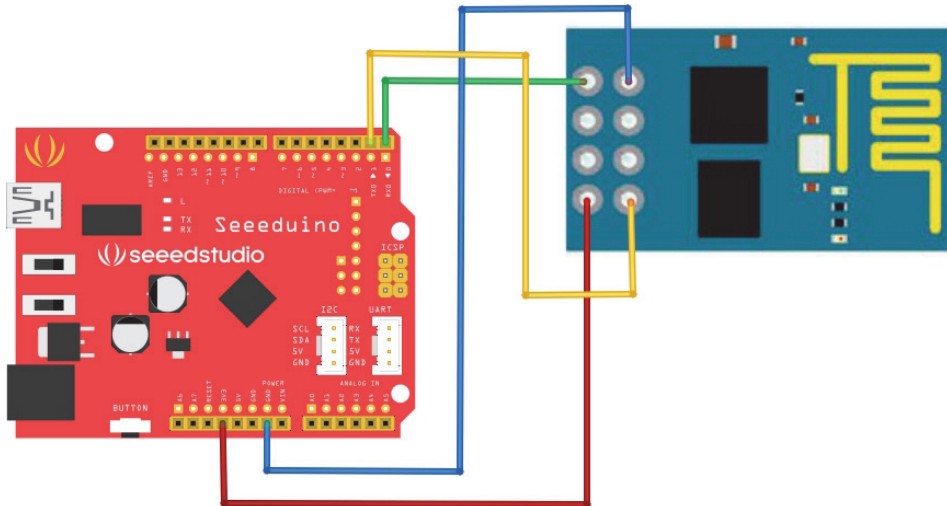


Abb. 3: Seeeduno mit ESP8266 ESP-01

Durch die serielle Kommunikation ist der serielle Port an Pin 0 und 1 belegt und kann nicht für Debugging und Textausgabe auf den seriellen Monitor verwendet werden. Falls eine zusätzliche serielle Schnittstelle zur Datenausgabe auf dem seriellen Monitor oder ein Display benötigt wird, kann eine softwaremäßige serielle Schnittstelle mit der Bibliothek `SoftwareSerial` ergänzt werden.

Abbildung 4 zeigt ein Beispiel von Seedstudio, bei dem ein USB/Seriell-Adapter des Typs UartSBee als zusätzlicher serieller Anschluss mit USB-Stecker eingesetzt wurde. Im Beispiel wird die zusätzliche serielle Schnittstelle an den Pins 10 und 11 angeschlossen. Grundsätzlich eignet sich jeder USB/Seriell-Adapter wie der USB FTDI Buddy oder auch der FTDI-Friend von Adafruit.

USB FTDI Buddy:

http://www.spikenzielabs.com/Catalog/spikenzielabs/micro-usb-ftdi-buddy?cPath=81_74

FTDI Friend:

<https://www.adafruit.com/product/284>

Dieses Beispiel ist im Seedstudio-Wiki als ESP-8266-Anwendung beschrieben:

https://wiki.seedstudio.com/WiFi_Serial_Transceiver_Module/

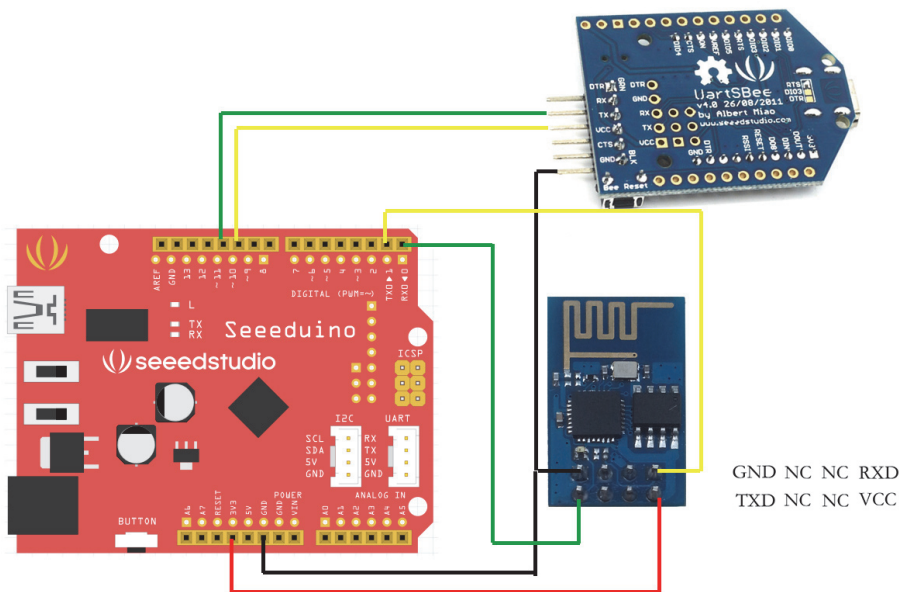


Abb. 4: ESP8266-Anwendung mit serielltem Anschluss zur Datenausgabe (Bild Seedstudio)

Tipp

Alternativ zum Arduino-Clone mit integriertem Signalpegel-Umschalter kann auch ein sogenannter Levelshifter, ein Modul zur Signalanpassung, mit einem Arduino UNO verwendet werden. Ein mögliches Modul liefert Adafruit als 4-kanaliges Breakoutboard, zu finden unter <http://www.adafruit.com/products/757>.

AT-Kommandos zur Ansteuerung des ESP8266 ESP-01

Mit den sogenannten AT-Kommandos, einem speziellen Befehlssatz, der früher zur Konfiguration von Modems verwendet wurde, kann das ESP8266-Modul über den seriellen Port angesteuert werden. Für die Datenübertragung wird dabei eine Baudrate von 115200 verwendet (ab Firmware-Version 0.90, frühere Versionen nutzten 57600)

AT-Kommandos werden über ein Terminalprogramm eingegeben und beginnen immer mit einem AT und einem nachfolgenden Kommando.

Beispiel (Abfrage der Firmware-Version):

```
AT+GMR
```

Abbildung 5 zeigt die Rückmeldung des AT-Kommandos. Der Autor hat dafür ein frei verfügbares Terminalprogramm verwendet.



Abb. 5: Abfrage ESP8266 über ein Terminalprogramm

In der Praxis mit dem Arduino wird eine serielle Ausgabe mittels `Serial.println()` erzeugt, statt ein Terminalprogramm zu verwenden.

Beispiel (Reset):

```
Serial.println("AT+RST");
```

Eine Vorkonfiguration des Transceivers ist nicht notwendig, da alle nötigen Informationen, wie beispielsweise die Zugangsdaten zum Access-Point, im Programmcode des Arduino-Boards eingetragen werden.

In Tabelle 2 sind die wichtigsten AT-Befehle für den Einsatz mit dem ESP8266-Modul aufgelistet.

AT-Befehl	Beschreibung
AT	Test-Befehl, gibt OK zurück
AT+RST	Reset des Moduls
AT+GMR	Abfrage der Firmware-Version
AT+CWMODE=<mode>	Setzen des WiFi-Betriebsmodus AT+CWMODE=1 (Station)
AT+CWMODE?	Abfrage des WiFi-Betriebsmodus
AT+CIPMUX=<mode>	Einstellung Anzahl Verbindungen 0: Einzelverbindung 1: mehrere Verbindungen AT+CIPMUX=1
AT+CWJAP=<ssid>,<pass>	Verbinden mit WiFi-Netzwerk AT+CWJAP="meinWiFi","meinPW"
AT+CIPSTATUS	Abfrage Status der IP-Verbindung
AT+CIPSTART=<Typ>,<Adresse>,<Port>	Erstellen einer IP-Verbindung AT+CIPSTART="TCP","192.168.1.10",8000
AT+CIPSEND=<Datenlänge>	Senden von Daten über einzelne Verbindung AT+CIPSEND=4 >1234 Hinweis: Daten werden erst auf anschließender Zeile übergeben
AT+CIPCLOSE	Schließen einer TCP- oder UDP-Verbindung

Tabelle 2: AT-Befehle zur Anwendung mit dem ESP8266

Eine ausführliche Liste der möglichen Befehle findet man unter:

<https://playground.bxotec.ch/doku.php/wireless/esp8266>

Webclient mit ESP8266

Dank der geringen Größe des ESP8266-Moduls eignet sich dieses ideal als Sensormodul, das empfangene oder erfasste Daten drahtlos an eine interne Webanwendung oder eine Webplattform wie Emoncms (<https://emoncms.org>) sendet. Das Erfassen und Senden dieser Informationen erfolgt meist in zeitlichen Abständen oder wird durch einen Zustand oder eine externe Aktion ausgelöst. Das Board mit

dem WiFi-Modul arbeitet also als Webclient, das Daten an die zentrale Webanwendung liefert. Im nachfolgenden Beispiel wird die Umgebungstemperatur mit einem Sensor des Typs LM35 ermittelt. In Abständen von 10 Sekunden wird der aktuelle Temperaturwert an eine PHP-Funktion im Internet übergeben.

Stückliste (Sensor-Webclient mit ESP8266)

- 1 Seeeduino 328 (siehe Text)
- 1 WiFi-Modul ESP8266 ESP-01
- 1 USB/Seriell-Adapter (USB FTDI Buddy o.ä.)
- 1 Temperatursensor LM35
- 1 Steckbrett
- Anschlussdrähte

In Abbildung 6 ist der Schaltungsaufbau für den Webclient dargestellt. Beim Verdrahten ist zu beachten, dass das WiFi-Modul mit 3,3 Volt versorgt wird. Der Temperatursensor LM35 dagegen muss mit 5 Volt versorgt werden, da die minimale Versorgungsspannung 4 Volt beträgt.

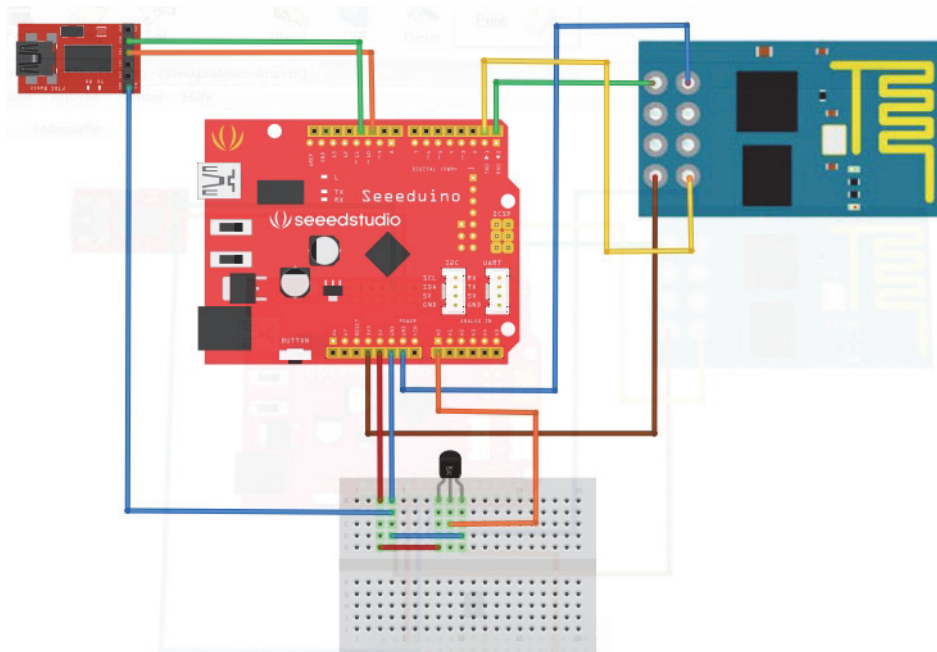


Abb. 6: Webclient mit ESP8266 ESP-01

Der Temperatursensor LM35 liefert eine analoge Ausgangsspannung von 10 mV/Grad Celsius. Bei einem Messbereich von 0 bis 100 Grad Celsius beträgt der ana-

loge Ist-Wert maximal 1,00 Volt. Somit kann der Arduino-Clone, im Beispiel der Seeeduno, auch mit einer Versorgungsspannung von 3,3 Volt den gesamten Messbereich des Sensors erfassen.

Zum Testen des Webclients wird eine PHP-Datei `tempmail.php` im lokalen Editor erstellt und der Code aus Listing 1 eingefügt. Die im Code erwähnte E-Mail-Adresse `meinname@meinewebsite.com` ist durch die eigene E-Mail-Adresse zu ersetzen. Nun kann die PHP-Datei auf einem Webserver platziert werden.

Zum Testen kann diese Skript-Datei nun im Browser aufgerufen werden, wobei ein Temperaturwert mit dem Parameter `t` übergeben werden muss.

`https://meinewebsite/tempmail.php?t=23.44`

Im PHP-Skript wird nun zuerst der Wert des Parameters `t` abgefragt und in der Variablen `$temp` gespeichert.

```
// Tempwert
$temp = $_REQUEST["t"];
```

Nun wird der Text der E-Mail in der Variablen `$mailtext` erstellt. Der eingelesene Temperaturwert wird dazu dem Text hinzugefügt.

```
// Mailtext
$mailtext = "Temp an Sensor: ".$temp." C";
```

Mit der PHP-Anweisung `mail()` kann nun die E-Mail versendet werden, indem die Empfängeradresse, der E-Mail-Titel, der E-Mail-Text und die Absenderadresse übergeben werden.

```
// Email versenden
$mail_sent=mail('meinname@
meinewebsite.com', 'Temp Sensor', $mailtext, "From: 'meinname@
meinewebsite.com");
```

Zum Schluss werden der Versendestatus und ein Infotext ausgegeben. Die Ausgabe ist aber nur im Webbrowser sichtbar.

```
echo "Mail Status: ".$mail_sent;
echo "Arduino sendet Mail".$mailtext;
```

Nachfolgend das ganze PHP-Skript in Listing 1.

```
<?php
// Tempwert
$temp = $_REQUEST["t"];

// Mailtext
$mailtext = "Temp an Sensor: ".$temp." C";

// Email versenden
$mail_sent=mail('meinname@
meinewebsite.com', 'Temp Sensor', $mailtext, "From: 'meinname@
meinewebsite.com");
echo "Mail Status: ".$mail_sent;
echo "Arduino sendet Mail".$mailtext;
?>
```

Listing 1: E-Mail versenden mit PHP

Das Programm auf dem Arduino-Clone beginnt mit dem Einbinden der notwendigen Bibliotheken. In unserem Fall wird die Software-Serial-Bibliothek eingebunden, damit weitere serielle Schnittstellen genutzt werden können.

```
#include <SoftwareSerial.h>
```

Nun werden die Zugangsdaten für den lokalen WiFi-Access-Point und die Daten der Website, auf der das PHP-Skript liegt, eingetragen. Der User-Agent gibt an, welche Art von Webclient den HTTP-Request ausgeführt hat.

Im Logfile des Webserver ist der Eintrag unseres User-Agents anschließend sichtbar.

```
#define SSID "MeineWifiSSID"
#define WIFIPASS "Passwort"
#define DSTIP "IP Server"
#define DSTPORT 80
#define HOST "meinewebsite.com"
#define USER_AGENT "User-Agent: Mozilla/0.1 (esp8266; WifiClient)"
```

Ein SoftwareSerial-Objekt `DbgSerial` wird erstellt und benannt. Dabei müssen die Pin-Nummern für die verwendeten Signale RX und TX angegeben werden.

```
// SoftSerial-Objekt, RX und TX
SoftwareSerial DbgSerial(10, 11);
```

Für die Verwendung des Temperatursensors werden die nötigen Variablen deklariert. Das Messsignal wird am analogen Eingang A0 angeschlossen.

```
// Tempensor
int SensorPin=0;
float valTemp;
float tempC;
```

In der Setup-Routine werden die beiden verwendeten seriellen Schnittstellen gestartet. Die Datenübertragung der Standard-Schnittstelle, die mit dem ESP8266 verbunden ist, muss auf 115200 Baud eingestellt werden.

Die vorher erstellte serielle Schnittstelle `DbgSerial` gibt Debug-Informationen im seriellen Monitor aus.

Die Initialisierung des ESP8266-Moduls erfolgt in der Funktion `espInit()`.

```
void setup()
{
  Serial.begin(115200);
  // Debug
  DbgSerial.begin(9600);
  DbgSerial.println("ESP8266 Start...");
  espInit();
}
```

Das Hauptprogramm wird nun jeweils die Funktion `sendData()` aufrufen, die die Messung der Temperatur und den Datenversand ausführt. Anschließend erfolgt eine Pause von 10 Sekunden.

```
void loop()
{
  sendData();
  delay(10000);
}
```

Die Initialisierungs-Funktion `espInit()` erledigt den Verbindungsaufbau mit dem ESP8266, das anschließend eine Verbindung mit dem angegebenen WiFi-Access-Point herstellt. Im ersten Schritt wird ein Reset-Befehl `AT+RST` gesendet.

```
void espInit()
{
```

```
// Reset
Serial.println("AT+RST");
delay(1000);
```

Nach einer Pause von 1000 Millisekunden wird geprüft, ob das Modul ein »ready« zurückgibt. Der Status wird über den Debug-Kanal ausgegeben.

```
if(Serial.find("ready"))
{
  DbgSerial.println("Module ready...");
}
else
{
  DbgSerial.println("No response...");
}
```

Nun werden die Befehle für den Betriebsmodus und die Verbindungsart gesendet. `cwmode=1` konfiguriert das Modul als Station.

```
Serial.println("AT+CWMODE=1");
delay(500);
```

Mit `AT+CIPMUX=0` wird eine Einzelverbindung definiert.

```
Serial.println("AT+CIPMUX=0");
delay(500);
```

Die Verbindung mit dem WiFi-Access-Point erfolgt über den Befehl `AT+CWJAP`, bei dem in weiteren Befehlszeilen die WiFi-SSID und das Passwort übergeben werden. Nach einer kurzen Pause ist die Initialisierung abgeschlossen.

```
// Anmelden an WiFi
Serial.print("AT+CWJAP=\"");
Serial.print(SSID);
Serial.print("\",\");
Serial.print(WIFIPASS);
Serial.println("\");
delay(3500);
}
```

Wie bereits erwähnt, wird aus dem Hauptprogramm in regelmäßigen Abständen die Funktion `sendData()` zur Datenmessung und dem anschließenden Datenversand aufgerufen.

```
void sendData()
{
    // Daten Sensor
    valTemp = analogRead(SensorPin);
    tempC = (5.0 * valTemp * 100.0)/1024;
```

Über den analogen Eingang A0 wird das analoge Signal des Temperatursensors LM35 gemessen und in der Variablen `valTemp` gespeichert. Der Messwert wird nun in eine reale Temperatur umgerechnet und der Variablen `tempC` zugewiesen.

Der Aufruf der PHP-Datei im Beispiel geschieht folgendermaßen: Der gemessene Temperaturwert wird als Wert mit dem Parameter `t` übergeben.

<https://meinewebsite.com/tempmail.php?t=22.45>

Zum Testen kann diese Adresse direkt im Webbrowser aufgerufen werden.

In der Anwendung wird der Webaufruf getrennt vom Hostnamen (im Beispiel `meinewebsite.com`) in einem Zeichen-Array `tpl[]` gespeichert. Die variablen Werte für Temperatur, Hostname und User-Agent werden als Platzhalter `%s` eingetragen.

```
char tpl[] = "GET /tempmail.php?t=%s HTTP/
1.1\r\nHost: %s\r\n%s\r\n\r\n";
```

Das Array `databuf[]` enthält schließlich den gesamten Webaufruf als String und wird deklariert.

```
char databuf[256] = "";
```

Der Temperaturwert wird für das Zusammenfügen aller Daten wird auch als Array deklariert. Der gemessene und umgerechnete Wert der aktuellen Temperatur wird anschließend mittels `dtostrf()` in das Array `temp[]` kopiert.

```
char temp[6];
dtostrf(tempC, 2, 2, temp);
```

In der Funktion `dtostrf(tempC, 2, 2, temp)` werden 4 Parameter übergeben. Diese bedeuten:

tempC: Float-Wert der Temperatur

2: Gesamte Länge der Zahl

2: Anzahl Nachkommastellen

temp: String, in den der Wert kopiert wird.

Nun liegen alle nötigen Einzeldaten vor und können in den vorbereiteten String `databuf[]` kopiert werden. Die in `tpl[]` vorbereiteten Platzhalter `%s` werden nun mit den Informationen aus `temp`, `HOST` und `USER_AGENT` befüllt.

```
sprintf(databuf, tpl, temp, HOST, USER_AGENT);
```

Um die Länge der zu sendenden Daten zu ermitteln, wird das Daten-Array `databuf[]` durchlaufen bis der Arraywert `databuf[ix]` gleich 0 ist. In diesem Fall wird mittels `break` die Schleifenausführung abgebrochen.

Der Wert `ix` entspricht dann der Länge der vorhandenen Daten.

```
int ix = 1;
for (ix=0; ix<256; ix++) {
    if ( databuf[ix] == 0 ) {
        break;
    }
}
```

Nach der Ermittlung der Länge der Daten beginnt der Verbindungsaufbau zur Ziel-Website (`meinewebsite.com`) mit `AT+CIPSTART`.

```
Serial.print("AT+CIPSTART=\"TCP\", \"\");
```

Nach Aufruf von IP-Adresse und Port-Nummer

```
Serial.print(DSTIP);
Serial.print("\",");
Serial.println(DSTPORT);
delay(500);
```

werden die Daten in `databuf[]` per Anweisung `AT+CIPSEND` versendet.

```
Serial.print("AT+CIPSEND=");
Serial.println(ix);
Serial.println(databuf);
delay(500);
}
```


Somit ist die ganze Funktion `sendData()` durchgearbeitet und der Programmablauf wird im Hauptprogramm mit der Pausenfunktion `delay(10000)` weitergeführt. Das bedeutet, dass der Webclient für 10 Sekunden pausiert, bevor das Hauptprogramm wieder startet.

Nach Starten des Webclients und Öffnen des seriellen Monitors in der Entwicklungsumgebung kann die Datenkommunikation mit dem WiFi-Modul überwacht werden (Abbildung 7). Der vom Temperatursensor gemessene Wert wird nun via HTTP-Request an die externe PHP-Datei gesendet.

Listing 2 zeigt das ganze Programm des drahtlosen Webclients mit dem ESP8266-Modul (`buch5_kap10_esp8266_webclient.ino`).

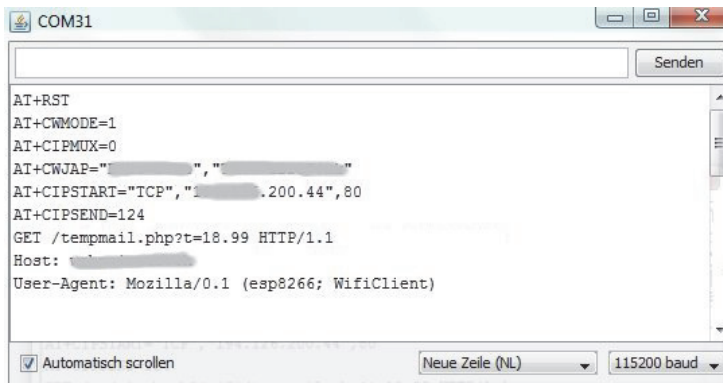


Abb. 7: Daten senden mit ESP8266 ESP-01

```

#include <SoftwareSerial.h>

#define SSID "MeinWiFiSSID"
#define WIFIPASS "Passwort"
#define DSTIP "IP Server"
#define DSTPORT 80
#define HOST "meinewebsite.com"
#define USER_AGENT "User-Agent: Mozilla/0.1 (esp8266; WifiClient)"

// SoftSerial Objekt, RX und TX
SoftwareSerial DbgSerial(10, 11);

// Tempsensor
int SensorPin=0;
float valTemp;
float tempC;
  
```

```
void setup()
{
  Serial.begin(115200);
  // Debug
  DbgSerial.begin(9600);
  DbgSerial.println("ESP8266 Start...");
  espInit();
}

void loop()
{
  sendData();
  delay(10000);
}

void espInit()
{
  // Reset
  Serial.println("AT+RST");
  delay(1000);
  if(Serial.find("ready"))
  {
    DbgSerial.println("Module ready...");
  }
  else
  {
    DbgSerial.println("No response...");
  }
  Serial.println("AT+CWMODE=1");
  delay(500);
  Serial.println("AT+CWMUX=0");
  delay(500);
  // Anmelden an WiFi
  Serial.print("AT+CWJAP=\"");
  Serial.print(SSID);
  Serial.print("\",\"");
  Serial.print(WIFIPASS);
  Serial.println("\");");
  delay(3500);
}
```

```

void sendData()
{
    // Daten Sensor
    valTemp = analogRead(SensorPin);
    tempC = (5.0 * valTemp * 100.0)/1024;
    char tpl[] = "GET /tempmail.php?t=%s HTTP/
1.1\r\nHost: %s\r\n%s\r\n\r\n";
    char databuf[256] = "";
    char temp[6];
    dtostrf(tempC, 2, 2, temp);
    sprintf(databuf, tpl, temp, HOST, USER_AGENT);
    int ix = 1;
    for (ix=0; ix<256; ix++) {
        if ( databuf[ix] == 0 ) {
            break;
        }
    }
    Serial.print("AT+CIPSTART=\"TCP\", \"");
    Serial.print(DSTIP);
    Serial.print("\",");
    Serial.println(DSTPORT);
    delay(500);
    Serial.print("AT+CIPSEND=");
    Serial.println(ix);
    Serial.println(databuf);
    delay(500);
}

```

Listing 2: Webclient mit ESP8266 ESP-01

Hinweis

Trotz der kompakten Bauform des WiFi-Moduls ESP8266 ESP-01 ist der Einsatz dieser Module für batteriebetriebene Anwendungen nicht geeignet. Die Module haben einen relativ hohen Stromverbrauch von bis zu 300 mA während der Startphase. Je nach eingesetztem Arduino-Board kann der Betrieb des ESP8266 die Versorgung stark belasten. Zur Stabilisierung der 3,3-V-Versorgung sollte ein zusätzlicher Stützkondensator von 100 bis 470 μ F parallel zur Versorgungsspannung angebracht werden.