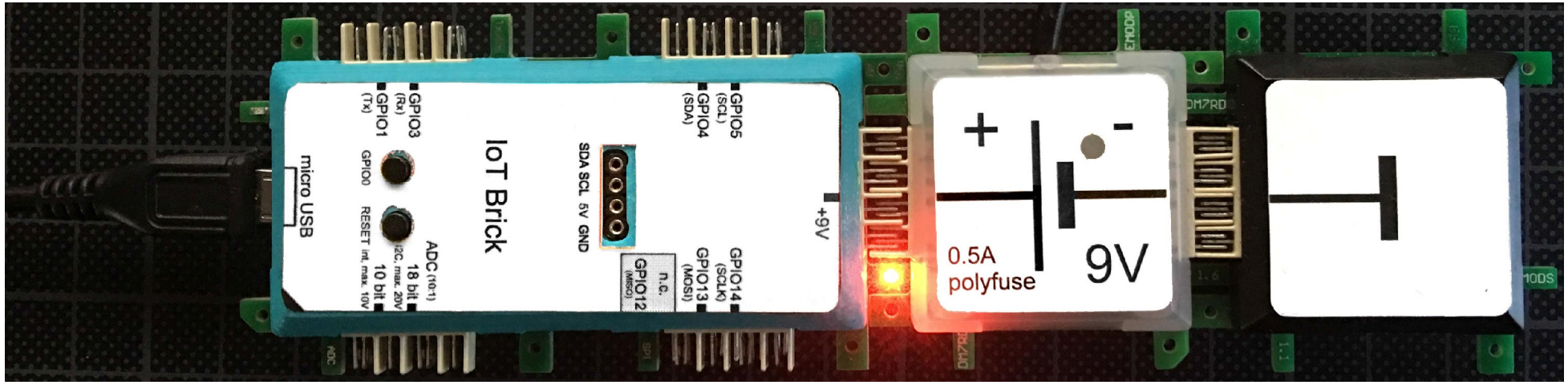
IoT-Brick-Library

Version 2017-07-21

Inhaltsverzeichnis

IoT-Brick Grundlagen



Bei der CPU des IoT-Bricks handelt es sich um einen EPS8266, das ist ein 32 Bit Microcontroller, der kompatibel zur Arduino IDE ist. Die interne Spannungsversorgung beträgt 3,3 Volt. Alle Ein- und Ausgänge des IoT-Bricks dürfen nicht mit mehr als 3,3 Volt betrieben werden. Man kann mit dem IoT-Brick grundsätzlich alle Programme nutzen, die für einen Arduino Nano oder einen anderen Arduino-kompatiblen Microcontroller geschrieben wurden. Dabei gibt es aber einige Einschränkungen wie z.B. die Anzahl der vorhandenen Digital- oder Analog-Leitungen. Der EPS8266 verfügt über 4 MB Flash-Speicher, wovon ein Teil (1 MB) für Programme (Sketche) und ein Teil (3 MB) als Image für Dateien benutzt wird. Der dynamische RAM-Speicher beträgt 96 kB, wovon 80 kB für Variablen zur Verfügung stehen. Die Kommunikation erfolgt über USB mit 115.200 Baud Datenrate. Die Fließkomma-Rechenleistung beträgt etwa 1,4 MFLOPs.

Der IoT-Brick enthält zusätzlich zum EOS8266 noch einen 18 bit A/D-Wandler vom Typ MCP3421, der über i2c angesprochen wird und ein optional einsteckbares blau-monochromes OLED-Display mit 128 x 64 Pixel Auflösung. Er verfügt über einen externen i2c-Bus, an dem weitere i2c-Geräte angeschlossen werden können. Weiterhin sind drei digitale Ein-/Ausgänge vorhanden und ein analoger 10 Bit Eingang. Auf der Oberseite befinden sich noch zwei Taster, einer für einen weiteren Digitaleingang und einer um einen Reset des IoT-Bricks auszulösen. An der Seite befindet sich eine LED, die sich mit dem Taster auf der Oberseite die selbe Digitalleitung teilt.

Programmiermodus

Falls die Kommunikation zwischen der Arduino IDE und dem IoT-Brick einmal nicht klappen sollte, kann man die Programmierung mit folgender Vorgehensweise manuell starten. Das kann notwendig werden, falls ein Programm nicht vollständig hochgeladen wurde weil z.B. der Ladevorgang unterbrochen, weil die Stromzufuhr oder die USB-Verbindung unterbrochen wurde.

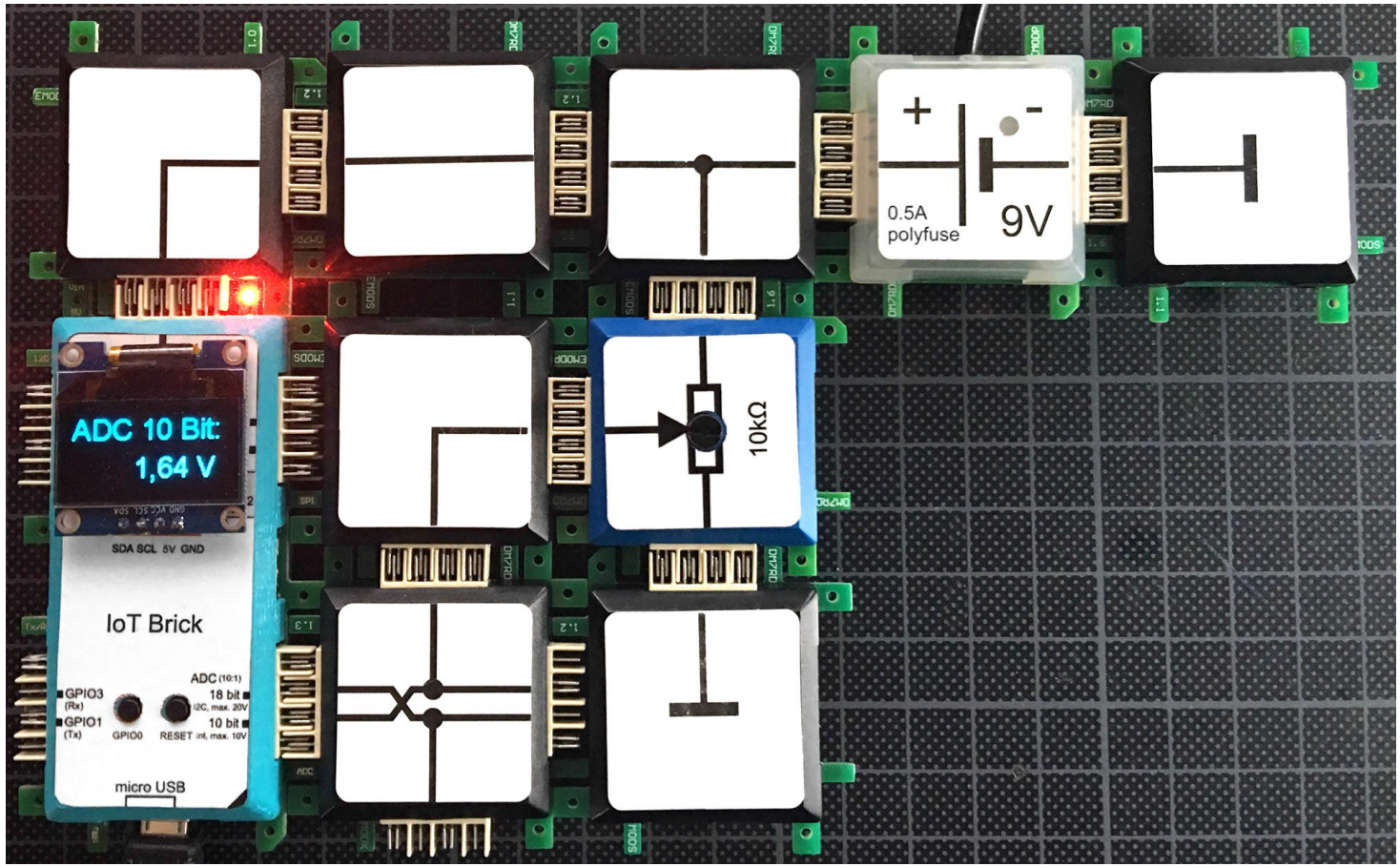
1. Programmier-Taste (GPIO0) gedrückt halten (rote LED unten links leuchtet hell)
2. Gleichzeitig Reset-Taste drücken
3. Reset-Taste wieder loslassen
4. Danach die Programmier-Taste loslassen (rote LED unten links leuchtet schwach)

Einstellungen in der Arduino IDE

```
Board: "Generic ESP8266 Module"
Flash Mode: "QIO"
Flash Size: "4M (3M SPIFFS)"
Debug port: "Disabled"
Debug Level: "Keine"
Reset Method: "nodemcu"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Upload Speed: "115200"
```

IoT-Brick Set Beispiel 6.5.3 Listing

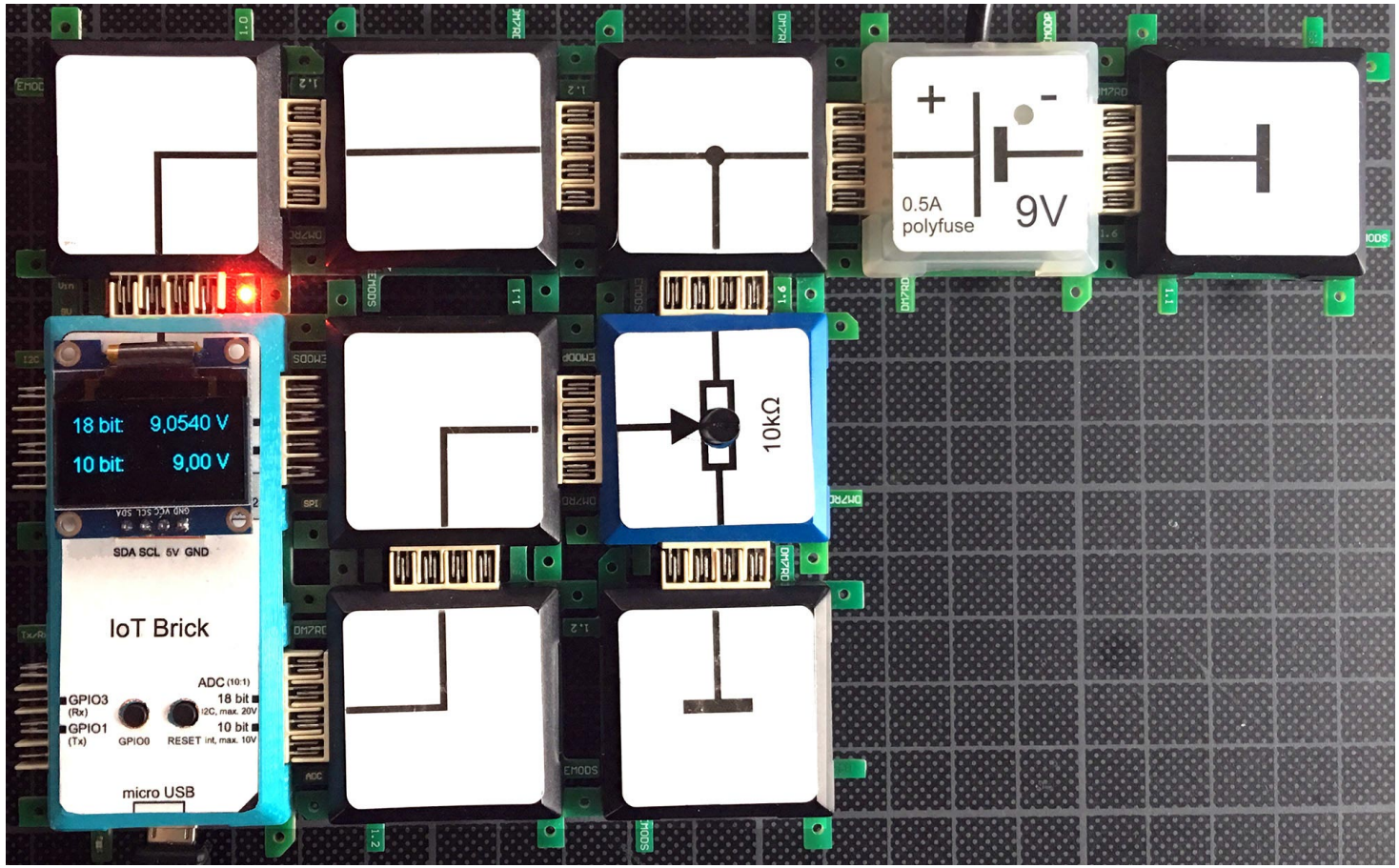
Das folgende Programm hat sich von 46 Zeilen auf 33 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Der Source-Code wurde so modifiziert, dass die Voltzahl mit einem Komma statt einem Punkt angezeigt wird.



```
//  Beispiel 6.5.3 "A/D Wandler 10 bit"
//
//  In diesem Beispiel wird die am internen 10bit ADC gemessene Spannung auf dem OLED angezeigt.
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

void setup(void)
{
    IoT_Init(true);
}

void loop(void)
{
    int analogIN = analogRead(IoT_Analog10bit);   // Liest Spannung als Rohwert: 0 = 0V bis 1023 = 1V-1LSB
    double UMess = ((double)analogIN / 1024) * 10.0;  // Umrechnen in Volt mit analogIN/Auflösung (*1Volt), *10 wegen 10:1 Teiler am ADC
    String UMess_str = strrealform(UMess, 32, 1, 2, false, false); // Spannungswert in String umwandeln für OLED Ausgabe mit Anzeige von 2
Nachkommastellen

    Serial.print(UMess);
    Serial.print(" V\t  digitaler Rohwert: ");
    Serial.println(analogIN);

    IoT_DisplayClear(24);                                          // OLED Display löschen (24 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_LEFT);
    IoT_DisplayDrawText(0, 0, "ADC 10 Bit:");
    IoT_DisplayAlignText(TEXT_ALIGN_RIGHT);
    IoT_DisplayDrawText(120, 32, UMess_str + " V");

    IoT_DisplayUpdate();                                             // OLED-Anzeige aktualisieren

    delay(1000);
}
```

IoT-Brick Set Beispiel 6.5.4 Listing

Das folgende Programm hat sich von 82 Zeilen auf 62 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Es wird gleichzeitig an beiden Wandlern gemessen. Der Source-Code wurde so modifiziert, dass die Voltzahlen mit einem Komma statt einem Punkt angezeigt werden.

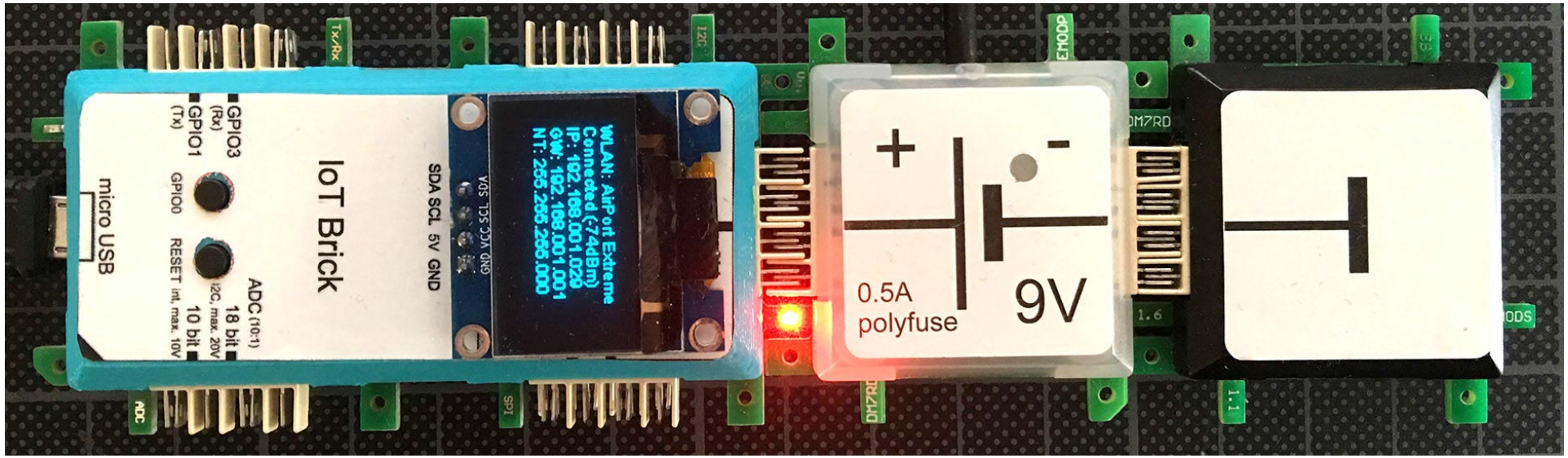


```
//  Beispiel 6.5.4 "A/D Wandler 18 bit"
//
//  In diesem Beispiel wird sowohl mit dem internen 10bit ADC
//  als auch dem I2C 18bit ADC gemessen und die Spannung auf dem OLED angezeigt.
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>


//Korrekturfaktor berechnen: Korrekturfaktor = UMess(Multimeter) / U(Anzeige Brick)
const float Correction_10bit = 1.0679; //hier Korrketurfaktor für 10bit ADC definieren
const float Correction_18bit = 1.0067; //hier Korrketurfaktor für 18bit ADC definieren


void setup(void)
{
    IoT_Init(true);
    Serial.println("18 bit ADC versus 10 bit ADC");
    Wire.begin();
    delay(2000);
}

void loop(void)
{
    int analogIN_10bit = analogRead(IoT_Analog10bit);    // Liest Spannung als Rohwert: 0 = 0V, 1023 = 1V-1LSB
    double UMess_10bit = ((double)analogIN_10bit/1024) * 10.0;  // Umrechnen in Volt analogIN_10bit/Auflösung (*1Volt), *10 wegen 10:1 Teiler am ADC
    // Die nächste Zeile zur Ermittlung des Korrekturfaktors mittels Multimeter-Messung auskommentieren
    UMess_10bit = UMess_10bit*Correction_10bit;  // Korrekturfaktor einrechnen (waehrend Abgleich auskommentieren)
    String UMess_10bit_str = strrealform(UMess_10bit, 32, 1, 2, false, false); // Spannun in String umwandeln für OLED Ausgabe mit 2 Nachkommastellen

    Serial.print("10 bit:\t");
    Serial.print(UMess_10bit);
    Serial.println(" V\t");

    double analogIN_18bit = IoT_ADC18bit.getDouble();       // Liest Spannung als Double-Werte aus I2C ADC, Eingangsbereich: 0 bis 2,048V
    double UMess_18bit = analogIN_18bit * 10.0;    // *10 wegen 10:1 Teiler am ADC
    // Die nächste Zeile zur Ermittlung des Korrekturfaktors mittels Multimeter-Messung bitte auskommentieren!
    UMess_18bit = UMess_18bit*Correction_18bit;  // Korrekturfaktor einrechnen (waehrend Abgleich auskommentieren)
    String UMess_18bit_str = strrealform(UMess_18bit, 32, 1, 4, false, false); // Spannung in String umwandeln für OLED Ausgabe mit 4 Nachkommastellen

    Serial.print("18 bit:\t");
    Serial.print(UMess_18bit_str);
    Serial.println(" V\t");
    Serial.println();

    IoT_DisplayClear(16);                                          // OLED Display löschen (16 Punkt Schrift)

    IoT_DisplayAlignText(TEXT_ALIGN_LEFT);
    IoT_DisplayDrawText(0, 0, "18 bit: ");
```

```
    IoT_DisplayAlignText(TEXT_ALIGN_RIGHT);
    IoT_DisplayDrawText(127, 0, UMess_18bit_str + " V");

    IoT_DisplayAlignText(TEXT_ALIGN_LEFT);
    IoT_DisplayDrawText(0, 32, "10 bit: ");
    IoT_DisplayAlignText(TEXT_ALIGN_RIGHT);
    IoT_DisplayDrawText(127, 32, UMess_10bit_str + " V");

    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren

    delay(1000);
}
```

IoT-Brick Set Beispiel 6.6.1 Listing

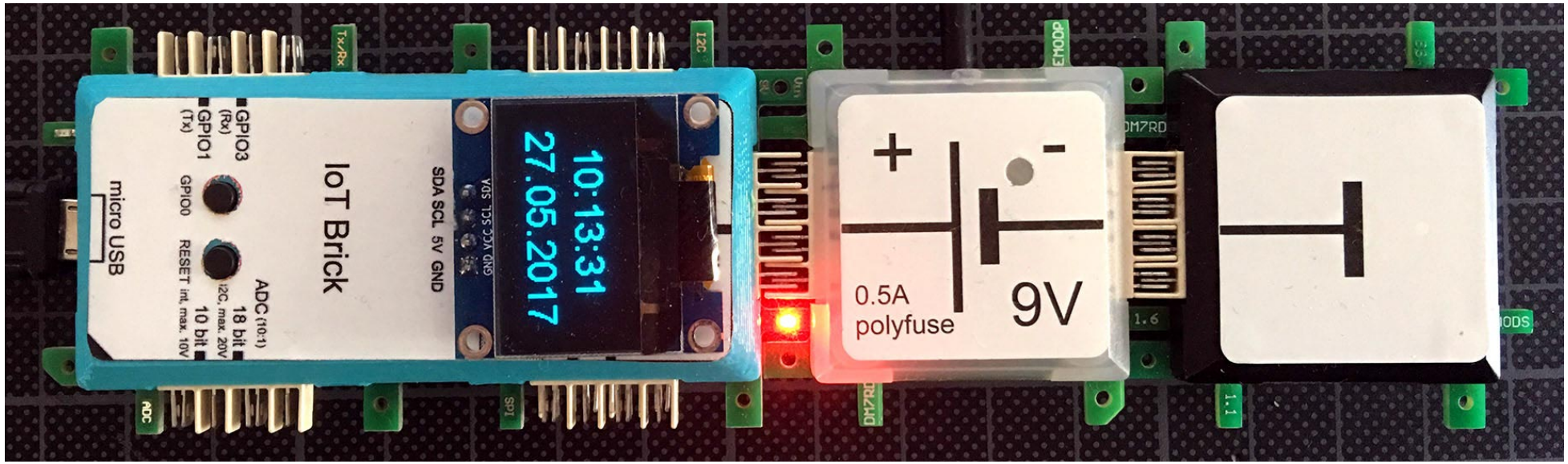
Das folgende Programm hat sich von 138 Zeilen auf 35 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Der Source-Code wurde so modifiziert, dass die Feldstärke in -dBm hinter dem „Connected“ angezeigt wird (aber nur bei erfolgreicher Verbindung), da der Name des WLAN-Netzwerkes oft so lang ist, dass dahinter nichts mehr hinpasst. Die IP-Adressen werden jetzt (wahlweise) mit führenden Nullen angezeigt.



```
//  Beispiel 6.6.1 "WLAN CLient mit OLED"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

void setup()
{
    IoT_Init(true);                                                 // IoT Brick initialiseren
    IoT_WLANconnect("(name)", "(password)");                        // An dieser Stelle die eigenen Zugangsdaten eintragen
}

void loop()
{
    IoT_DisplayClear(10);                                           // OLED Display löschen (10 Punkt Schrift)
    String state = IoT_WLANstatus();                                // Verbindungsstatus
    if (IoT_WLANinited)                                             // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
    {
        IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());       // WLAN Netzwerk Name
        if (state == "Connected")                                   // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
        {
            state = state + " (" + str(IoT_WLANrssi()) + "dBm)";    // Signalstärke
        }
        IoT_DisplayDrawText(5, 10, state);                          // Verbindungsstatus & Signalstärke
        IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true)); // IP Adresse
        IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true)); // IP Gateway
        IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));  // IP Subnetzmaske
    }
    else                                                            // Falls WLAN Verbindung gescheitert ist, Status im OLED Display anzeigen
    {
        IoT_DisplayDrawText(5, 10, state);
    }
    IoT_DisplayUpdate();                                            // OLED-Anzeige aktualisieren
    delay(1000);                                                    // 1 Sekunde warten
}
```

IoT-Brick Set Beispiel 6.6.2 Listing

Das folgende Programm hat sich von 145 Zeilen auf 64 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Der Source-Code wurde so modifiziert, dass die Feldstärke in -dBm hinter dem „Connected“ angezeigt wird (aber nur bei erfolgreicher Verbindung), da der Name des WLAN-Netzwerkes oft so lang ist, dass dahinter nichts mehr hinpasst. Die IP-Adressen werden jetzt (wahlweise) mit führenden Nullen angezeigt. Solange keine korrekte Uhrzeit empfangen wurde, wird ein „Bitte warten“ im OLED-Display angezeigt. Das datum wird mit Punkten „.“ statt Slashes „/“ angezeigt, was für das deutsche Datumsformat richtig ist.

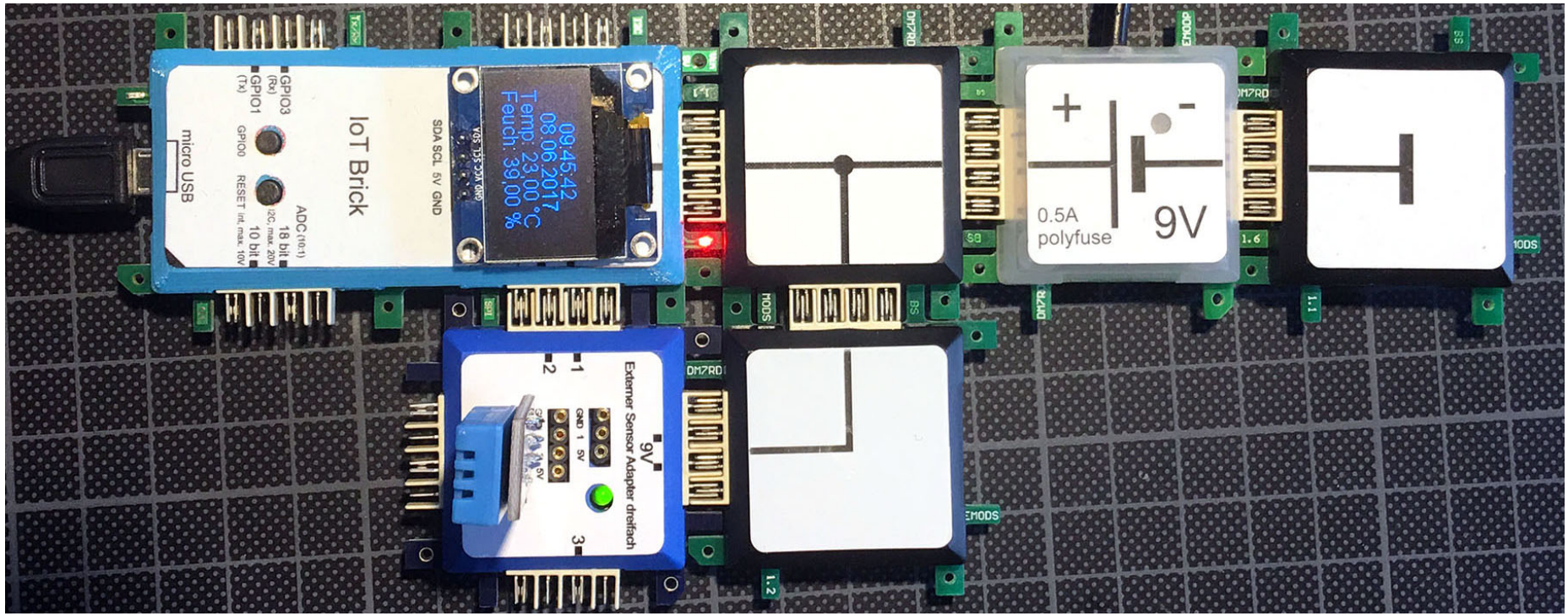


```
//  Beispiel 6.6.2 "Zeit aus dem Internet"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

int counter = 0;

void setup()
{
    IoT_Init(true);
    IoT_WLANconnect("(name)", "(password)");                          // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();
}

void loop()
{
    if (IoT_Keypress())                                               // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayClear(10);                                         // OLED Display löschen (10 Punkt Schrift)
        String state = IoT_WLANstatus();                              // Verbindungsstatus
        if (IoT_WLANinited)                                           // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());     // WLAN Netzwerk Name
            if (state == "Connected")                                 // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
            {
                state = state + " (" + str(IoT_WLANrssi()) + "dBm)";  // Signalstärke
            }
            IoT_DisplayDrawText(5, 10, state);                        // Verbindungsstatus & Signalstärke
            IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true)); // IP Adresse
            IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true)); // IP Gateway
            IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));  // IP Subnetzmaske
        }
        else                                                          // Falls WLAN Verbindung gescheitert ist, Status im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 10, state);
        }
    }
    else
    {
        if (counter % 25 == 0)                                        // Infos seriell nicht zu oft ausgeben
        {
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        }
        IoT_DisplayClear(24);                                         // OLED Display löschen (24 Punkt Schrift)
        if (IoT_NTPvalid())                                           // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
        {
```

```
            IoT_DisplayDrawText(15, 5, IoT_NTPtime());
            IoT_DisplayDrawText(5, 34, IoT_NTPdate());
        }
        else                                                    // "Bitte warten..." zentriert anzeigen
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);            // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Bitte");
            IoT_DisplayDrawText(64, 34, "warten...");
        }
    }
    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren
    delay(100);                                                 // 100 ms. warten
    counter++;
}
```

IoT-Brick Set Beispiel 6.6.3 Listing

Das folgende Programm hat sich von 224 Zeilen auf 121 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Der Source-Code wurde so modifiziert, dass die Feldstärke in -dBm hinter dem „Connected“ angezeigt wird (aber nur bei erfolgreicher Verbindung), da der Name des WLAN-Netzwerkes oft so lang ist, dass dahinter nichts mehr hinpasst. Die IP-Adressen werden jetzt (wahlweise) mit führenden Nullen angezeigt. Solange keine korrekte Uhrzeit empfangen wurde, wird ein „Bitte warten“ im OLED-Display angezeigt. Das datum wird mit Punkten „.“ statt Slashes „/“ angezeigt, was für das deutsche Datumsformat richtig ist. Temperatur und Feuchtigkeit werden mit Hilfe des Sensors DHT11 bestimmt. Falls kein Sensor angeschlossen ist, werden keine Temperatur und keine Feuchtigkeit angezeigt. Das Programm läuft aber in dem Fall trotzdem einwandfrei.Die Sensor-Library (DHT) wurde nicht in die Hauptbibliothek übernommen, da diese zu speziell ist und für die meisten Programme nicht benötigt wird bzw. auch unter der Verwendung unterschiedlicher PIN-Belegungen und Sensortypen Verwendung findet, was sonst nur umständlich möglich wäre.



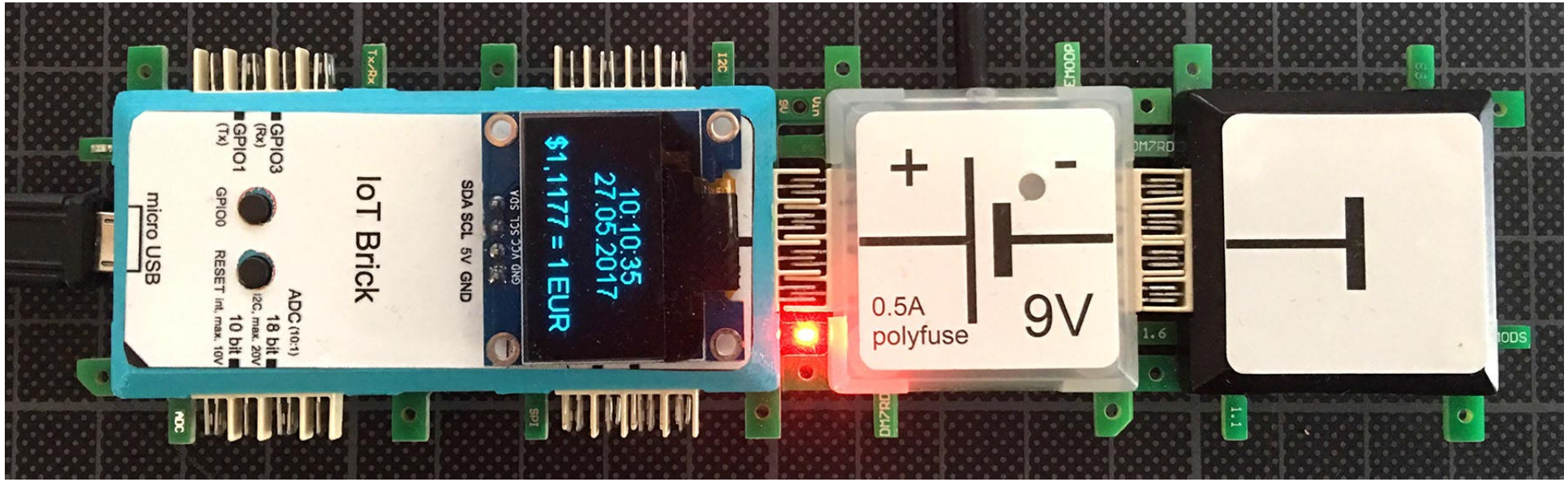
```
//  Beispiel 6.6.3 "Temperatur und Feuchtigkeit messen mit DHT11"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

#include <DHT.h>
#define DHT_TYPE DHT11                                          // Sensortyp definieren für DHT11
const int DHT_PIN = 14;                                         // Datenleitung des Sensors an GPIO14 des IoT Bricks
DHT dht(DHT_PIN, DHT_TYPE);                                     // Variable vom Typ DHT definieren

String Temperature;
String Humidity;

int counter = 0;

void setup()
{
    IoT_Init(true);
    IoT_WLANconnect("(name)", "(password)");                    // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();
    dht.begin();                                                // Sensor initialisieren
}

void loop()
{
    IoT_Idle();

    if (IoT_Keypress())                                         // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayClear(10);                                   // OLED Display löschen (10 Punkt Schrift)
        String state = IoT_WLANstatus();                        // Verbindungsstatus
        if (IoT_WLANinited)                                     // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());       // WLAN Netzwerk Name
            if (state == "Connected")                                   // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
            {
                state = state + " (" + str(IoT_WLANrssi()) + "dBm)";    // Signalstärke
            }
            IoT_DisplayDrawText(5, 10, state);                  // Verbindungsstatus & Signalstärke
            IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true));   // IP Adresse
            IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true));   // IP Gateway
            IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));    // IP Subnetzmaske
        }
        else                                                    // Falls WLAN Verbindung gescheitert ist, Status im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 10, state);
        }
    }
    else
```

```
{
    IoT_Idle();
    if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
    {
        if (IoT_NTPeventAvail)                                          // Zeitevent triggern
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
        Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
        Serial.print("WiFi is ");
        Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
        Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
    }

    IoT_DisplayClear(16);                                               // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                            // Zentrierte Darstellung des Textes
    if (IoT_NTPvalid())                                                 // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
    {
        IoT_DisplayDrawText(63, 0, IoT_NTPtime());
        IoT_DisplayDrawText(63, 15, IoT_NTPdate());
        IoT_DisplayAlignText(TEXT_ALIGN_LEFT);                          // Linksbündige Darstellung des Textes
        if (counter % 100 == 0)                                         // Sensor etwa alle 10 Sekunden abfragen
        {
            Serial.print("Reading Sensors... ");
            IoT_Idle();
            float t = dht.readTemperature();                            //Temperatur auslesen (Celsius)
            if (not(isnan(t)))
            {
                Temperature = strrealform (t, 32, 1, 2, true, false) + " °C"; // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. 0 nach dem Komma
                Serial.print(Temperature + ". ");
            }
            else
            {
                Serial.print("No Temperature Sensor. ");
            }
            IoT_Idle();
            float h = dht.readHumidity();                                 //Feuchtigkeit auslesen (Prozent)
            if (not(isnan(h)))
            {
                Humidity = strrealform (h, 32, 1, 2, true, false) + " %"; // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. Nullen nach dem Komma
                Serial.print(Humidity + ". ");
            }
            else
            {
                Serial.print("No Humidity Sensor. ");
            }
            Serial.println("");
        }
        IoT_Idle();
```

```
        }
        else                                                    // "Bitte warten..." zentriert anzeigen
        {
            IoT_DisplayClear(24);                               // OLED Display löschen (24 Punkt Schrift)
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);            // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Bitte");
            IoT_DisplayDrawText(64, 34, "warten...");
        }
        if (Temperature != "")
        {
            IoT_DisplayDrawText(5, 30, "Temp: " + Temperature);       // Temperatur anzeigen
        }
        if (Humidity != "")
        {
            IoT_DisplayDrawText(5, 45, "Feuch: " + Humidity);         // Feuchtigkeit anzeigen
        }
    }
    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren
    delay(100);                                                 // 100 ms. warten
    counter++;
}
```

IoT-Brick Set Beispiel 6.6.4 Listing

Das folgende Programm hat sich von 215 Zeilen auf 80 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Der Source-Code wurde so modifiziert, dass die Feldstärke in -dBm hinter dem „Connected“ angezeigt wird (aber nur bei erfolgreicher Verbindung), da der Name des WLAN-Netzwerkes oft so lang ist, dass dahinter nichts mehr hinpasst. Die IP-Adressen werden jetzt (wahlweise) mit führenden Nullen angezeigt. Solange keine korrekte Uhrzeit empfangen wurde, wird ein „Bitte warten“ im OLED-Display angezeigt. Das datum wird mit Punkten „.“ statt Slashes „/“ angezeigt, was für das deutsche Datumsformat richtig ist. Die Währungsanzeige im Display ist jetzt mit „$kurs = 1 EUR“ angezeigt. Die Währungs-Library wurde nicht in die Hauptbibliothek übernommen, da diese zu speziell ist und für die meisten Programme nicht benötigt wird. Daher bleiben die Währungs-Aufrufe wie gehabt.



```
//  Beispiel 6.6.4 "Dollarkurs aus dem Internet"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <CurrencylayerClient.h>

CurrencylayerClient currencylayer;
int counter = 0;

void setup()
{
    IoT_Init(true);
    IoT_WLANconnect("(name)", "(password)");                          // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();
}

void loop()
{
    if (IoT_Keypress())                                               // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayClear(10);                                         // OLED Display löschen (10 Punkt Schrift)
        String state = IoT_WLANstatus();                              // Verbindungsstatus
        if (IoT_WLANinited)                                           // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());     // WLAN Netzwerk Name
            if (state == "Connected")                                 // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
            {
                state = state + " (" + str(IoT_WLANrssi()) + "dBm)";  // Signalstärke
            }
            IoT_DisplayDrawText(5, 10, state);                        // Verbindungsstatus & Signalstärke
            IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true));  // IP Adresse
            IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true));  // IP Gateway
            IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));   // IP Subnetzmaske
        }
        else                                                          // Falls WLAN Verbindung gescheitert, Status im OLED Display
        {
            IoT_DisplayDrawText(5, 10, state);
        }
    }
    else
    {
        if (counter % 25 == 0)                                        // Infos seriell nicht zu oft ausgeben
        {
            if (IoT_NTPeventAvail)                                    // Zeitevent triggern
            {
                IoT_NTPprintEvent(IoT_NTPcurrentEvent);
                IoT_NTPeventAvail = false;
            }
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
```

```
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        }
        if (counter % 50 == 0)                                      // Umrechnungskurs etwa alle 5 Sekunden aktualisieren
        {
            currencylayer.getLastChannelItem();
        }
        IoT_DisplayClear(16);                                       // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
        if (IoT_NTPvalid())                                         // Prüfen, ob gültige Uhrzeit aus dem Internet empfangen wurde
        {
            IoT_DisplayDrawText(63, 0, IoT_NTPtime());
            IoT_DisplayDrawText(63, 15, IoT_NTPdate());
            IoT_DisplayAlignText(TEXT_ALIGN_LEFT);
            IoT_DisplayFontSize(16);
            IoT_DisplayDrawText(0, 45, "$" + replace(currencylayer.getFieldValue(0), ".", ",") + " = 1 EUR   ");
        }
        else                                                        // "Bitte warten..." zentriert anzeigen
        {
            IoT_DisplayClear(24);                                   // OLED Display löschen (24 Punkt Schrift)
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Bitte");
            IoT_DisplayDrawText(64, 34, "warten...");
        }
    }
    IoT_DisplayUpdate();                                            // OLED-Anzeige aktualisieren
    delay(100);                                                     // 100 ms. warten
    counter++;
}
```

IoT-Brick Set Beispiel 6.6.5 Listing

Das folgende Programm hat sich von 304 Zeilen auf 162 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Die Sensor-Library (DHT) wurde nicht in die Hauptbibliothek übernommen, da diese zu speziell ist und für die meisten Programme nicht benötigt wird bzw. auch unter der Verwendung unterschiedlicher PIN-Belegungen und Sensortypen Verwendung findet, was sonst nur umständlich möglich wäre. Daher bleiben die Sensor-Aufrufe wie gehabt. Die Klasse `IoT_WebServer` repräsentiert den Webserver des IoT-Bricks und ist in der Library definiert. Der `loop()` wurde an mehreren Stellen mit `IoT_Idle()` entspannt. Zusätzlich wird am Ende des `loop()` eine Verzögerung `delay(100)` von 100 ms. eingebaut. Diese Verzögerung gibt dem im Hintergrund laufenden Prozessen genug Zeit und verhindert das unkontrollierte Neu-Starten des IoT-Bricks. Falls kein Sensor angeschlossen ist, werden keine Temperatur und keine Feuchtigkeit angezeigt. Das Programm läuft aber in dem Fall trotzdem einwandfrei.

Hello World!

Das ist eine sehr einfache Website von deinem IoT Brick, die Datum, Uhrzeit, Temperatur und Feuchtigkeit anzeigt.

Datum: 27.05.2017

Uhrzeit: 10:03:11

Temperatur: 23,45 °C

Feuchte: 62,34 %

Durch Neuladen der Website können die Werte jederzeit aktualisiert werden.

```
//  Beispiel 6.6.5 "Meine erste Webseite"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

#include <DHT.h>

#define DHT_TYPE DHT11                                          // Sensortyp definieren für DHT11
const int DHT_PIN = 14;                                         // Datenleitung des Sensors an GPIO14 des IoT Bricks
String Temperature;
String Humidity;

DHT dht(DHT_PIN, DHT_TYPE);                                     // Variable vom Typ DHT definieren

int counter = 0;

void setup()
{
    IoT_Init(true);
    IoT_WLANconnect("(name)", "(password)");                    // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();
    dht.begin();                                                // Sensor initialisieren
                                                                // Sobald der Browser direkt auf das Stammverzeichnis zugreift,
    IoT_WebServer.on("/", handleRoot);                          // führe die Funktion handleRoot (siehe unten) aus.
    IoT_WebServer.begin();                                      // ab jetzt "hört" der Server auf HTTP-Anfragen
    Serial.println("HTTP server started");
}

void loop()
{
    IoT_Idle();

    IoT_WebServer.handleClient();                               // Bediene die http Anfragen

    if (IoT_Keypress())                                           // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayClear(10);                                     // OLED Display löschen (10 Punkt Schrift)
        String state = IoT_WLANstatus();                          // Verbindungsstatus
        if (IoT_WLANinited)                                       // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());       // WLAN Netzwerk Name
            if (state == "Connected")                                   // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
            {
                state = state + " (" + str(IoT_WLANrssi()) + "dBm)";    // Signalstärke
            }
            IoT_DisplayDrawText(5, 10, state);                          // Verbindungsstatus & Signalstärke
            IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true)); // IP Adresse
            IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true)); // IP Gateway
            IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));  // IP Subnetzmaske
```

```
    }
    else                                                    // Falls WLAN Verbindung gescheitert ist, Status im OLED Display anzeigen
    {
        IoT_DisplayDrawText(5, 10, state);
    }
}
else
{
    IoT_Idle();
    if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
    {
        if (IoT_NTPeventAvail)                              // Zeitevent triggern
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
        Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
        Serial.print("WiFi is ");
        Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
        Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
    }

    IoT_DisplayClear(16);                                   // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                // Zentrierte Darstellung des Textes
    if (IoT_NTPvalid())                                     // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
    {
        IoT_DisplayDrawText(63, 0, IoT_NTPtime());
        IoT_DisplayDrawText(63, 15, IoT_NTPdate());
        IoT_DisplayAlignText(TEXT_ALIGN_LEFT);              // Linksbündige Darstellung des Textes
        if (counter % 100 == 0)                             // Sensor etwa alle 10 Sekunden abfragen
        {
            Serial.print("Reading Sensors... ");
            IoT_Idle();
            float t = dht.readTemperature();                //Temperatur auslesen (Celsius)
            if (not(isnan(t)))
            {
                Temperature = strrealform (t, 32, 1, 2, true, false) + " °C"; // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. 0 nach dem Komma
                Serial.print(Temperature + ". ");
            }
            else
            {
                Serial.print("No Temperature Sensor. ");
            }
            IoT_Idle();
            float h = dht.readHumidity();                           //Feuchtigkeit auslesen (Prozent)
            if (not(isnan(h)))
            {
                Humidity = strrealform (h, 32, 1, 2, true, false) + " %"; // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. Nullen nach dem Komma
                Serial.print(Humidity + ". ");
            }
```

```
                else
                {
                    Serial.print("No Humidity Sensor. ");
                }
                Serial.println("");
            }
            IoT_Idle();
        }
        else                                                    // "Bitte warten..." zentriert anzeigen
        {
            IoT_DisplayClear(24);                               // OLED Display löschen (24 Punkt Schrift)
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);            // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Bitte");
            IoT_DisplayDrawText(64, 34, "warten...");
        }
        if (Temperature != "")
        {
            IoT_DisplayDrawText(5, 30, "Temp: " + Temperature);         // Temperatur anzeigen
        }
        if (Humidity != "")
        {
            IoT_DisplayDrawText(5, 45, "Feuch: " + Humidity);           // Feuchtigkeit anzeigen
        }
    }
    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren
    delay(100);                                                 // 100 ms. warten
    counter++;
}
```

```
//Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot()
{
    String content;
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    String temp_web = Temperature;
    String humi_web = Humidity;
    content = "<!DOCTYPE html>";
    content += "<html>";
    content += "<head>";
    // Falls du die Werte auf deiner Website aktualisieren möchtest kannst du folgende Zeile verwenden.
    // Vorher musst du die IP-Adresse durch die von deinem IoT Brick ersetzen:
    // content += "<meta http-equiv=\"refresh\" content=\"5; URL=http://192.168.1.153\">";
    // zur automatischen Aktualisierung der Seite nach 5 Sekunden
    content += "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">";  // damit das Grad-Zeichen "°" korrekt dargestellt wird
    content += "<title>Meine erste IoT Brick-Website</title>";
    content += "</head>";
    content += "<body>";
    content += "<h1> Hello World! </h1>";
    content += "<p>Das ist eine sehr einfache Website von deinem IoT Brick, die Datum, Uhrzeit, Temperatur und Feuchtigkeit anzeigt.</p>";
    content += "<h2>Datum: "+date_web+"</h2>";
    content += "<h2>Uhrzeit: "+time_web+"</h2>";
    content += "<h2>Temperatur: "+temp_web+"</h2>";
    content += "<h2>Feuchte: "+humi_web+"</h2>";
    content += "<p>Durch Neuladen der Website können die Werte jederzeit aktualisiert werden.</p>";
    content += "</body>";
    content += "</html>";
    IoT_WebServer.send(200, "text/html", content);   // HTTP-Statuscode 200 = OK (Anfrage wurde erfolgreich bearbeitet)
}
```

IoT-Brick Set Beispiel 6.6.6 Listing

Das folgende Programm hat sich von 476 Zeilen auf 358 Zeilen Source-Code verringert. Alle hardwarespezifischen Aufrufe sind aus dem Sktch verschwunden, daher braucht beim Wechsel auf eine andere CPU lediglich die Library angepasst werden. Die Filesystem-Library (FS) wurde nicht in die Hauptbibliothek übernommen, da diese sehr speziell ist und für die meisten Programme nicht benötigt wird. Die Klasse `IoT_WebServer` repräsentiert den Webserver des IoT-Bricks und ist in der Library definiert. Der `loop()` wurde an mehreren Stellen mit `IoT_Idle()` entspannt. Zusätzlich wird am Ende des `loop()` eine Verzögerung `delay(100)` von 100 ms. eingebaut. Diese Verzögerung gibt dem im Hintergrund laufenden Prozessen genug Zeit und verhindert das unkontrollierte Neu-Starten des IoT-Bricks. Dieses Programm ist vor allem als Machbarkeits-Beispiel zu sehen. Hier wurde mit Java gearbeitet, um ein modernes UI (Benutzer-Schnittstelle) zu zeigen. Der Web-Server wird durch Java allerdings deutlich langsamer, so dass die Anzahl der Elemente auf dieser Seite sehr beschränkt ist.

Die zusätzlich benötigten Daten (z.B. Java, bootstrap) werden in einen speziell dafür reservierten Teil des 4 MB großen Flash Speichers (Image-Bereich) geladen. Dieser Image-Bereich kann wahlweise 1 oder 3 MB groß sein. Die Größe wird in der Arduino-IDE festgelegt. Mit einem zusätzlich zu installierenden Java-Tool, welches in dem „Werkzeuge“-Menü der Arduino-IDE erscheint, wird dann aus dem neben dem Programm befindlichen „data“-Ordner das benötigte Image erstellt und auf den IoT-Brick geladen. Das Image hat dabei stets eine feste Größe von 1 oder 3 MB, unabhängig davon, wieviele Daten sich tatsächlich darin befinden. Das Laden eines 3 MB großen Images benötigt etwa 5 Minuten. Das liegt daran, dass der IoT-Brick über USB die Daten nur mit etwa 100 kBit Übertragungsrate empfangen kann. Das Hochladen des Programms dauert etwa 35 Sekunden. Die Internetseite im Browser sieht so aus:

Das Neuladen der Web-Seite dauert etwa 3 Sekunden. Die Seite sollte allerdings nur von maximal zwei Browsern oder Fenstern in einem Browser gleichzeitig aufgerufen werden. Wenn man die Seite von einem dritten Browser aus aufruft, dauert es bereits rund 30 Sekunden und die Uhrzeit im OLED-Display bleibt währenddessen mehrmals für einige Sekunden stehen. Dieses Verhalten ist allerdings nicht immer reproduzierbar und hängt davon ab, ob die Aufrufe wirklich gleichzeitig oder mit einiger Pause dazwischen erfolgen. Die in den vier Handle-Funktionen für die GPIO-Schalter enthaltenen Befehle „`IoT_WebServer.send(200, "text/html"...`“ haben anscheinend keine Funktion, denn wenn man diese auskommentiert, ergeben sich keine Änderungen an der Webseite und deren Funktionalität.

```
//  Beispiel 6.6.6 "LEDs via Website schalten"
//
//  Die Bilddateien sowie die CSS und JavaScript Dateien aus dem Verzeichnis "data"
//  (siehe Unterverzeichnis im Beispiel_6.6.6), müssen mit dem "ESP8266 Sketch Data Upload"
//  Tool übertragen werden. Dieses erscheint nach der Installation im "Werkzeuge"-Menü.
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

#include "FS.h"

int counter = 0;

int gpio13 = 13;
int gpio14 = 14;
String gpio13Name = "GPIO13";
String gpio14Name = "GPIO14";
int dOn = HIGH;
int dOff = LOW;

void setup()
{
    IoT_Init(true);
    IoT_WLANconnect("(name)", "(password)");                        // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();

    IoT_WebServer.on("/", handleRoot); // Wenn der Browser direkt auf das Stammverzeichnis zugreift, führe die Funktion handleRoot (siehe unten) aus.
    /*  Folgende Einträge bestimmen, welche Funktionen ausgeführt werden wenn man bestimmte Links im Browser aufruft,
     *  falls z.B. der IoT Brick die IP Adresse 192.168.1.153 vom DHCP Server erhalten hat, dann würde das Ausrufen der
     *  Adresse http://192.168.1.153/img/brklogo.png dazu führen, dass die Funktion handleImgLogo ausgeführt wird, die wiederum
     *  die Bilddatei brklogo.png aus dem internen Dateisystem (SPIFFS) liest und an den Webserver liefert.
     */

    // JS (Javascript)
    IoT_WebServer.on("/js/jquery", handleJsJquery);
    IoT_WebServer.on("/js/bootstrap", handleJsBootstrap);
    IoT_WebServer.on("/js/bootstrap-switch", handleJsBootstrapSwitch);
```

```
    // CSS (Cascaded Style Sheets
    IoT_WebServer.on("/css/bootstrap", handleCssBootstrap);
    IoT_WebServer.on("/css/bootstrap-switch", handleCssBootstrapSwitch);
    // Images (Bilder)
    IoT_WebServer.on("/img/bg.png", handleImgBg);
    IoT_WebServer.on("/img/brklogo.png", handleImgLogo);

    // WebServer Handles für die GPIO Schaltfunktionen
    IoT_WebServer.on("/switch13-on", handleGpio13On);
    IoT_WebServer.on("/switch13-off", handleGpio13Off);
    IoT_WebServer.on("/switch14-on", handleGpio14On);
    IoT_WebServer.on("/switch14-off", handleGpio14Off);

    IoT_WebServer.begin();
    Serial.println("HTTP server started");
    setupPins();
}

void loop()
{
    IoT_Idle();

    IoT_WebServer.handleClient(); //Bediene die HTTP-Anfragen

    IoT_Idle();

    if (IoT_Keypress())                                          // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        if (counter % 10 == 0) //Infos seriell nicht zu oft ausgeben (ca. jede Sekunden, nur wenn der Button gedrückt wird)
        {
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        }
    }
    else
    {
        IoT_Idle();
        if (counter % 25 == 0)                                    // NTP Event alle 2 Sekunden auslesen
        {
            if (IoT_NTPeventAvail)                                  // Zeitevent-Infos im Terminal ausgeben
            {
                IoT_NTPprintEvent(IoT_NTPcurrentEvent);
                IoT_NTPeventAvail = false;
            }
        }
        IoT_DisplayClear(16);                                     // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                  // Zentrierte Darstellung des Textes
```

```
        if (IoT_NTPvalid())                                             // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
        {
            IoT_DisplayDrawText(63, 0, IoT_NTPtime());
            IoT_DisplayDrawText(63, 15, IoT_NTPdate());
            IoT_Idle();
        }
        else                                                            // "Bitte warten..." zentriert anzeigen
        {
            IoT_DisplayClear(24);                                         // OLED Display löschen (24 Punkt Schrift)
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                      // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Bitte");
            IoT_DisplayDrawText(64, 34, "warten...");
        }
    }
    IoT_DisplayUpdate();                                              // OLED-Anzeige aktualisieren
    delay(100);                                                       // 100 ms. warten
    counter++;
}

void handleRoot()
{
    String content;
    content = "<!DOCTYPE html>";
    content += "<html>";
    content += "<head>";
    content += "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">";  // wichtig, damit das Grad-Zeichen "°" korrekt dargestellt
wird
    content += "<meta http-equiv=\"cache-control\" content=\"max-age=0\">";
    content += "<meta http-equiv=\"cache-control\" content=\"no-cache\">";
    content += "<meta http-equiv=\"expires\" content=\"0\">";
    content += "<meta http-equiv=\"expires\" content=\"Tue, 01 Jan 1980 1:00:00 GMT\">";
    content += "<meta http-equiv=\"pragma\" content=\"no-cache\">";
    content += "<meta http-equiv=\"Content-Language\" content=\"de_DE\">";
    content += "<meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\">";
    content += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1, user-scalable=no\">";
    content += "<meta name=\"description\" content=\"BrickRknowledge IoT Brick-Website\">";
    content += "<meta name=\"author\" content=\"ALLNET\">";
    content += "<title>IoT Brick via WLAN " + IoT_WLANssid() + " schalten</title>";
    content += "<link rel=\"stylesheet\" href=\"/css/bootstrap\">";
    content += "<link rel=\"stylesheet\" href=\"/css/bootstrap-switch\">";
    content += "<style>body{background-image:url(/img/bg.png);margin:0;padding:20px;background-size:100% auto;background-repeat:no-repeat;font-fa-
mily:\"Helvetica Neue\",Helvetica,Arial,sans-serif;font-size:14px;line-height:1.5;color:#333;background-color:#fff}.col-sm-2{margin-top:20px}.img-
thumbnail{border:0}</style>";
    content += "</head>";
    content += "<body>";
    content += "<div class=\"container-fluid\">";
    content += "<div class=\"row\">";
    content += "<div class=\"col-sm-2\"><img class=\"img-thumbnail\" src=\"/img/brklogo.png\"></div>";
    content += "</div>";
    content += "<div class=\"row\">";
```

```
    content += "<div class=\"col-sm-2\"><input type=\"checkbox\" id=\"switch14\" data-label-text=\""+gpio14Name+"\" data-label-width=\"120\"></div>";
    content += "<div class=\"col-sm-2\"><input type=\"checkbox\" id=\"switch13\" data-label-text=\""+gpio13Name+"\" data-label-width=\"120\"></div>";
    content += "</div>";
    content += "</div>";
    content += "<script src=\"/js/jquery\"></script>";
    content += "<script src=\"/js/bootstrap\"></script>";
    content += "<script src=\"/js/bootstrap-switch\"></script>";
    content += "<script type=\"text/javascript\">";
    content += "$('input[type=\"checkbox\"]').bootstrapSwitch({onSwitchChange:function(){$.ajax({url:'/'+$(this).prop('id')+'-
'+($(this).prop('checked')?'on':'off')});});});";
    content += "var
setAdc=function(){$.ajax({url:'/adc'}).done(function(data){data=data||{};if(data.hasOwnProperty('data')){$('#adc').text(data.data);}}).fail(function(jq
xhr,textStatus,error){}).always(function(){setTimeout(setAdc,1000);});};setAdc();";
    content += "</script>";
    content += "</body>";
    content += "</html>";
    IoT_WebServer.send(200, "text/html", content);   // HTTP-Statuscode 200 = OK (Anfrage wurde erfolgreich bearbeitet)
    IoT_Idle();
}

/*********************************************************************************************
 * Für alle die sich schon gut mit HTML auskennen und selbst die Website modifizieren
 * möchten, finden im Folgenden den HTML-Code aus der Funktion handleRoot() "im Klartext".
 * Am besten du kopiert ihn in einen HTML-Editor deiner Wahl.
 * *******************************************************************************************
 <!DOCTYPE html>
 <html>
 <head>
 <meta charset="utf-8">
 <meta http-equiv="cache-control" content="max-age=0">
 <meta http-equiv="cache-control" content="no-cache">
 <meta http-equiv="expires" content="0">
 <meta http-equiv="expires" content="Tue, 01 Jan 1980 1:00:00 GMT">
 <meta http-equiv="pragma" content="no-cache">
 <meta http-equiv="Content-Language" content="de_DE">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no"><meta name="description" content="BrickRknowledge IoT Brick
Website">
 <meta name="author" content="ALLNET">
 <title>IoT Brick via "+network+" schalten</title>
 <link rel="stylesheet" href="/css/bootstrap">
 <link rel="stylesheet" href="/css/bootstrap-switch">
 <style>
 body{background-image:url(/img/bg.png);margin:0;padding:20px;background-size:100% auto;background-repeat:no-repeat;font-family:"Helvetica
Neue",Helvetica,Arial,sans-serif;font-size:14px;line-height:1.5;color:#333;background-color:#fff}.col-sm-2{margin-top:20px}.img-thumbnail{border:0}
 </style>
 </head>
 <body>
 <div class="container-fluid">
 <div class="row">
```

```
 <div class="col-sm-2">
 <img class="img-thumbnail" src="/img/brklogo.png">
 </div>
 </div>
 <div class="row">
 <div class="col-sm-2">
 <input type="checkbox" id="switch13" data-label-text="gpio13Name" data-label-width="120">
 </div>
 <div class="col-sm-2">
 <input type="checkbox" id="switch14" data-label-text="gpio14Name+" data-label-width="120">
 </div>
 </div>
 </div>
 <script src="/js/jquery"></script>
 <script src="/js/bootstrap"></script>
 <script src="/js/bootstrap-switch"></script>
 <script type="text/javascript">$('input[type="checkbox"]').bootstrapSwitch({onSwitchChange:function(){$.ajax({url:'/'+$(this).prop('id')+'-
'+($(this).prop('checked')?'on':'off')});}});var
setAdc=function(){$.ajax({url:'/adc'}).done(function(data){data=data||{};if(data.hasOwnProperty('data')){$('#adc').text(data.data);}}).fail(function(jq
xhr,textStatus,error){}).always(function(){setTimeout(setAdc,1000);});};setAdc();</script>
 </body>
 </html>
 ******************************
 *    Ende des HTML-Codes     *
 ******************************
 */


// GPIO 13 und 14 werden als Ausgang konfiguriert um die LEDs schalten zu können
void setupPins()
{
    pinMode(gpio13, OUTPUT);
    pinMode(gpio14, OUTPUT);
    digitalWrite(gpio13, dOff);
    digitalWrite(gpio14, dOff);
}

//Handle-Funktion für GPIO13 ON
void handleGpio13On()
{
    digitalWrite(gpio13, dOn);
    //IoT_WebServer.send(200, "text/html", gpio13Name + " on");
    Serial.println("GPIO13 -> ON");
}

//Handle-Funktion für GPIO13 OFF
void handleGpio13Off()
{
    digitalWrite(gpio13, dOff);
    //IoT_WebServer.send(200, "text/html", gpio13Name + " off");
```

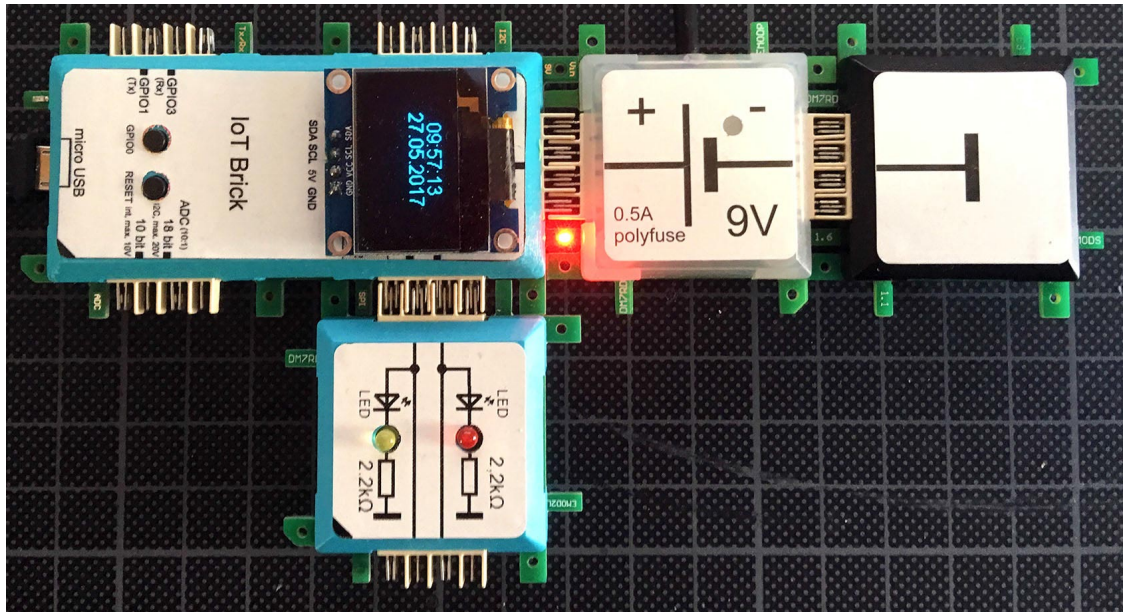
```
    Serial.println("GPIO13 -> OFF");
}

//Handle-Funktion für GPIO14 ON
void handleGpio14On()
{
    digitalWrite(gpio14, dOn);
    //IoT_WebServer.send(200, "text/html", gpio14Name + " on");
    Serial.println("GPIO14 -> ON");
}

//Handle-Funktion für GPIO14 OFF
void handleGpio14Off()
{
    digitalWrite(gpio14, dOff);
    //IoT_WebServer.send(200, "text/html", gpio14Name + " off");
    Serial.println("GPIO14 -> OFF");
}

/*
 * Folgende Funktionen sind notwendig um diverse Dateien die JQuery und JavaScript Funktionen
 * beinhalten aus dem internen Filesystem (SPIFFS definiert in FS.h) des IoT Bricks auszulesen
 * und dem WebServer zur Verfügung zu stellen.
 */

void handleJsJquery() {
    String path = "/jquery.js.gz";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f)
    {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "application/javascript");
        f.close();
    }
    else
    {
        Serial.println("JsJquery open failed");
    }
    SPIFFS.end();
}

void handleJsBootstrap()
{
    String path = "/bootstrap.js.gz";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f) {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "application/javascript");
```

```
        f.close();
    } else {
        Serial.println("JsBootstrap open failed");
    }
    SPIFFS.end();
}

void handleJsBootstrapSwitch()
{
    String path = "/bootstrap-switch.js.gz";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f) {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "application/javascript");
        f.close();
    } else {
        Serial.println("JsBootstrapSwitch open failed");
    }
    SPIFFS.end();
}

void handleCssBootstrap()
{
    String path = "/bootstrap.css.gz";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f) {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "text/css");
        f.close();
    } else {
        Serial.println("CssBootstrap open failed");
    }
    SPIFFS.end();
}

void handleCssBootstrapSwitch()
{
    String path = "/bootstrap-switch.css.gz";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f) {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "text/css");
        f.close();
    } else {
        Serial.println("CssBootstrapSwitch open failed");
    }
    SPIFFS.end();
```

```
}

void handleImgBg()
{
    String path = "/bg.png";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f) {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "image/png");
        f.close();
    } else {
        Serial.println("ImgBg open failed");
    }
    SPIFFS.end();
}

void handleImgLogo()
{
    String path = "/brklogo.png";
    SPIFFS.begin();
    File f = SPIFFS.open(path, "r");
    if(f) {
        f.setTimeout(0);
        size_t sent = IoT_WebServer.streamFile(f, "image/png");
        f.close();
    } else {
        Serial.println("ImgBrklogo open failed");
    }
    SPIFFS.end();
}
```

IoT-Brick Set Beispiel 7.1 Listing

Das folgende Programm mit 185 Zeilen ist nicht Bestandteil des IoT-Handbuchs und zeigt die Erstellung einer Webseite mit verschiedenen Elementen wie Ausgabefeldern, Eingabefeldern, Checkboxen und Radioboxen. Die Klasse `IoT_WebServer` repräsentiert den Webserver des IoT-Bricks und ist in der Library definiert. Der `loop()` wurde an mehreren Stellen mit `IoT_Idle()` entspannt. Zusätzlich wird am Ende des `loop()` eine Verzögerung `delay(100)` von 100 ms. eingebaut. Diese Verzögerung gibt dem im Hintergrund laufenden Prozessen genug Zeit und verhindert das unkontrollierte Neu-Starten des IoT-Bricks. Dieses Programm ist als Machbarkeits-Beispiel für eine komplexe Webseite zu sehen. Auf Java wurde dabei komplett verzichtet. Das Neuladen der Web-Seite dauert weniger als 300 Millisekunden. Dadurch werden mit dieser Art der Programmierung auch Webseiten mit dutzenden von Elementen möglich. Selbst wenn man die Seite in zehn verschiedenen Fenstern hintereinander öffnet, bleiben die Ladezeiten unter einer Sekunde.



```
//  Beispiel 7.1 "Eingabe eines Vor- und Nachnamens auf der HTML-Seite"
//               "und Ausgabe desselben Namens im Terminal und auf der"
//               "Internetseite selbst. Brick Display kann geschaltet"
//               "werden, GPIOS können geschaltet werden."
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>


int counter = 0;
int currentGPIO = 0;                                            // 0 = Beide aus, 1 = 13 ein, 2 = 14 ein, 3 = Beide ein
int unicodeMode = 1;                                            // 0 = Asc II Namen, 1 = Unicode Namen
int displayMode = 1;                                            // 0 = Display aus,  1 = Display ein

String vorname_web    = "(Vorname)";
String nachname_web   = "(Nachname)";


void setup()
{
    IoT_Init(true);
    IoT_WLANconnect("(name)", "(password)");                    // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();

    IoT_WebServer.on("/", handleRoot); // Sobald der Browser direkt auf das Stammverzeichnis zugreift, führe die Funktion handleRoot (siehe unten) aus.
    IoT_WebServer.on("/submit", handleSubmit);                  // Handler für die Eingabe der Namen in den Eingabefeldern
    IoT_WebServer.begin();                                      // ab jetzt "horcht" der Server auf HTTP-Anfragen
    Serial.println("HTTP server started");
}

void loop()
{
    IoT_Idle();
    IoT_WebServer.handleClient();                                 // Bediene die http Anfragen
    if (IoT_Keypress())                                           // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        if (counter % 10 == 0)                                    // Infos seriell nicht zu oft ausgeben (ca. 1 Sekunde)
        {
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        }
    }
    else
    {
        IoT_Idle();
        if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
```

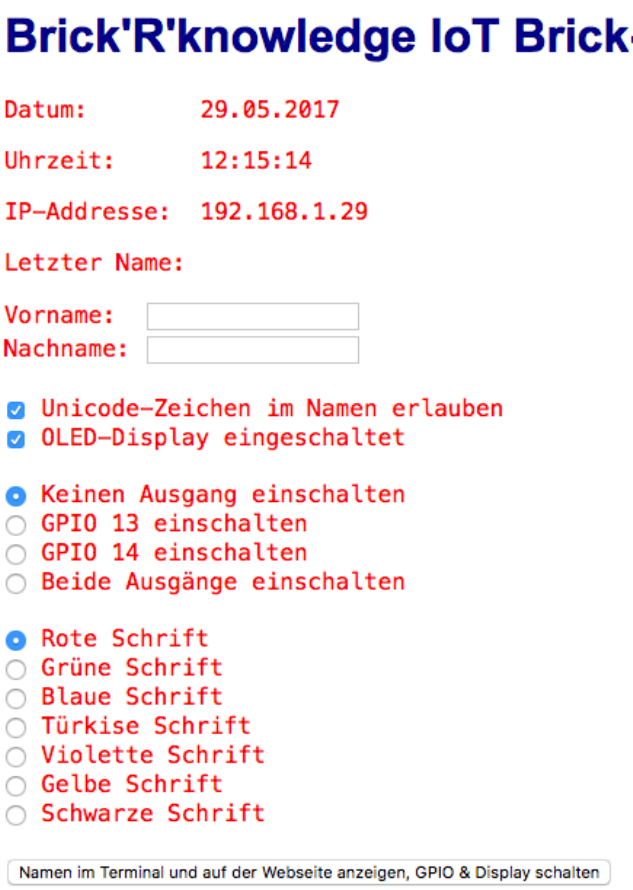
```
        {
            if (IoT_NTPeventAvail)                                  // Zeitevent triggern
            {
                IoT_NTPprintEvent(IoT_NTPcurrentEvent);
                IoT_NTPeventAvail = false;
            }
        }

        IoT_DisplayClear(16);                                       // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
        if (IoT_NTPvalid())                                         // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
        {
            if (displayMode == 1)
            {
                IoT_DisplayDrawText(63, 0, IoT_NTPtime());          // Uhrzeit im Display anzeigen
                IoT_DisplayDrawText(63, 15, IoT_NTPdate());         // Datum im Display anzeigen
                IoT_Idle();
            }
        }
        else                                                        // "Bitte warten..." zentriert anzeigen
        {
            if (displayMode == 1)
            {
                IoT_DisplayClear(24);                               // OLED Display löschen (24 Punkt Schrift)
                IoT_DisplayAlignText(TEXT_ALIGN_CENTER);            // Zentrierte Darstellung des Textes
                IoT_DisplayDrawText(64, 5, "Bitte");
                IoT_DisplayDrawText(64, 34, "warten...");
            }
        }
    }
    IoT_DisplayUpdate();                                            // OLED-Anzeige aktualisieren
    delay(100);                                                     // 100 ms. warten
    counter++;
}

//Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot ()
{
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    String ipaddr_web  = IoT_WLANaddress(false);
    String title_web = "Brick'R'knowledge IoT Brick-Website";
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String content =                // Die HTML-Seite in lesbarer Formatierung, darf auch Variablen enthalten
    "<html>\
    <head>\
    <title>" + title_web + "</title>\
```

```
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: #000088; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p>Datum:          " + date_web + "</p>\
    <p>Uhrzeit:        " + time_web + "</p>\
    <p>IP-Addresse:  " + ipaddr_web + "</p>\
    <p>Letzter Name: " + vorname_web + " " + nachname_web + "</p>\
    <form action = 'http://" + ipaddr_web + "/submit' method='POST'>\
    Vorname:  <input type='text' name='vname' value=''><br>\
    Nachname: <input type='text' name='nname' value=''><br>\
    <br>\
    <input type='checkbox' name='unicode' value='chkd' checked> Unicode-Zeichen im Namen erlauben<br>\
    <input type='checkbox' name='display' value='chkd' checked> OLED-Display eingeschaltet<br>\
    <br>\
    <input type='radio' id='led0' name='LEDgroup' value='none' checked> Keinen Ausgang einschalten<br>\
    <input type='radio' id='led1' name='LEDgroup' value='GPIO13'> GPIO 13 einschalten<br>\
    <input type='radio' id='led2' name='LEDgroup' value='GPIO14'> GPIO 14 einschalten<br>\
    <input type='radio' id='led3' name='LEDgroup' value='both'> Beide Ausg&auml;nge einschalten<br>\
    <br>\
    <input type = 'submit' value='Namen im Terminal und auf der Webseite anzeigen, GPIO & Display schalten'><br>\
    </form>\
    </body>\
    </html>";
    IoT_WebServer.send(200, "text/html", content);                    // HTTP-Statuscode 200 = OK (Anfrage wurde erfolgreich bearbeitet)
}
```

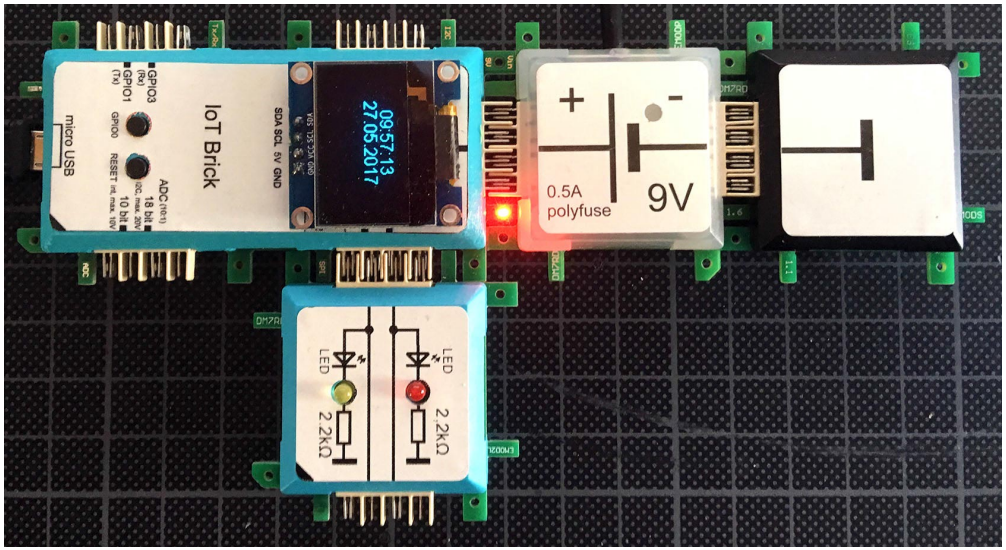
```
void handleSubmit ()
{
    Serial.println("Button wurde betätigt");
    IoT_WebPrintAllFields();                                    // Alle gefundenen Felder auf dem Terminal ausgeben
    if (IoT_WebTestField("vname"))                              // Testen, ob ein Feld mit dem Namen "vname" existiert
    {
        vorname_web = IoT_WebGetField("vname");                 // Das Web-Feld "vname" auslesen
    }
    if (IoT_WebTestField("nname"))                              // Testen, ob ein Feld mit dem Namen "vname" existiert
    {
        nachname_web = IoT_WebGetField("nname");                // Das Web-Feld "nname" auslesen
    }
    unicodeMode = boolval(IoT_WebTestField("unicode"));         // Testen, ob die Checkbox mit dem Namen "unicode" angekreuzt wurde
    displayMode = boolval(IoT_WebTestField("display"));         // Testen, ob die Checkbox mit dem Namen "display" angekreuzt wurde
    if (IoT_WebTestField("LEDgroup"))                           // Testen, ob Felder mit dem Namen "LEDgroup" existieren
    {
        String selection = IoT_WebGetField("LEDgroup");
        if (selection == "none")                                // Beide Leitungen aus
        {
            digitalWrite(13, LOW);
            digitalWrite(14, LOW);
            currentGPIO = 0;
        }
        else if (selection == "GPIO13")                         // GPIO 13
        {
            digitalWrite(13, HIGH);
            digitalWrite(14, LOW);
            currentGPIO = 1;
        }
        else if (selection == "GPIO14")                         // GPIO 14
        {
            digitalWrite(13, LOW);
            digitalWrite(14, HIGH);
            currentGPIO = 2;
        }
        else if (selection == "both")                           // Beide Leitungen ein
        {
            digitalWrite(13, HIGH);
            digitalWrite(14, HIGH);
            currentGPIO = 3;
        }
    }
    if (unicodeMode == 0)
    {
        vorname_web  = cleanasc(vorname_web);
        nachname_web = cleanasc(nachname_web);
    }
    handleRoot();                                               // Wieder auf die Hauptseite zurück schalten
}
```

IoT-Brick Set Beispiel 7.2 Listing

Das folgende Programm mit 266 Zeilen ist nicht Bestandteil des IoT-Handbuchs und ist eine Erweiterung zum Programm 7.1 mit einer weiteren Gruppe Radioboxen zur Farbwahl. Der eigentliche Versuchsaufbau ist hardwaretechnisch zum 7.1 identisch. In diesem Programm wird beim Starten auf eine Terminalverbindung gewartet, damit man den kompletten Verbindungsverlauf im Terminal beobachten kann. Der dadurch verzögerte WLAN-Verbindungsaufbau stellt eine besondere Herausforderung an den IoT-Brick dar. Siehe `IoT_WLANconnect`.




```
WLAN access point AirPort Extreme
Connecting ....................................
Connected to AirPort Extreme, IP: 192.168.1.29
HTTP server started
Got NTP time: 11:59:57 29/05/2017
```



```
//  Beispiel 7.2 "Erweiterung des Sketches 7.1 um einige Terminal-Befehle"
//               "und einer neuen Radio-Box-Gruppe zur Schrift-Farbauswahl."
//               "Außerdem erscheint eine Fehlermeldung, wenn keine WLAN-"
//               "Verbindung hergestellt werden konnte."
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>


int counter = 0;
int currentGPIO = 0;                                        // 0 = Beide aus, 1 = 13 ein, 2 = 14 ein, 3 = Beide ein
int unicodeMode = 1;                                        // 0 = Asc II Namen, 1 = Unicode Namen
int displayMode = 1;                                        // 0 = Display aus,  1 = Display ein
int colorMode = 1;                                          // 1 = Rote Schrift, 2 = Grüne Schrift, 3 = Blaue Schrift usw.

String vorname_web    = "(Vorname)";
String nachname_web   = "(Nachname)";


void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANconnect("(name)", "(password)");                // An dieser Stelle die eigenen Zugangsdaten eintragen
    IoT_NTPinit();

    IoT_WebServer.on("/", handleRoot); // Sobald der Browser direkt auf das Stammverzeichnis zugreift, führe die Funktion handleRoot (siehe unten) aus.
    IoT_WebServer.on("/submit", handleSubmit);              // Handler für die Eingabe der Namen in den Eingabefeldern
    IoT_WebServer.begin();                                  // ab jetzt "horcht" der Server auf HTTP-Anfragen
    Serial.println("HTTP server started");
}

void loop()
{
    IoT_Idle();

    IoT_WebServer.handleClient();                           // Bediene die http Anfragen

    if (IoT_Keypress())                                     // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        if (counter % 10 == 0)                              // Infos seriell nicht zu oft ausgeben (ca. 1 Sekunde)
        {
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        }
    }
```

```
    else
    {
        IoT_Idle();
        if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
        {
            if (IoT_NTPeventAvail)                                          // Zeitevent triggern
            {
                IoT_NTPprintEvent(IoT_NTPcurrentEvent);
                IoT_NTPeventAvail = false;
            }
        }

        IoT_DisplayClear(16);                                               // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                            // Zentrierte Darstellung des Textes
        if (IoT_NTPvalid())                                                 // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
        {
            if (displayMode == 1)
            {
                IoT_DisplayDrawText(63, 0, IoT_NTPtime());                    // Uhrzeit im Display anzeigen
                IoT_DisplayDrawText(63, 15, IoT_NTPdate());                   // Datum im Display anzeigen
                IoT_Idle();
            }
        }
        else                                                                // "Bitte warten..." zentriert anzeigen
        {
            if (displayMode == 1)
            {
                if (IoT_WLANinited)
                {
                    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
                    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
                    IoT_DisplayDrawText(64, 5, "Bitte");
                    IoT_DisplayDrawText(64, 34, "warten...");
                }
                else
                {
                    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
                    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
                    IoT_DisplayDrawText(64, 5, "Kein");
                    IoT_DisplayDrawText(64, 34, "WLAN");
                }
            }
        }
    }
    IoT_DisplayUpdate();                                                  // OLED-Anzeige aktualisieren
    delay(100);                                                           // 100 ms. warten
    t_TerminalRead();                                                     // Reset-Anforderung prüfen
    counter++;
}
```

```
//Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot ()
{
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    String ipaddr_web  = IoT_WLANaddress(false);
    String title_web = "Brick'R'knowledge IoT Brick-Website";
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String ColorHexVal = "";
    switch (colorMode)
    {
        case 1:
            ColorHexVal = "#FF0000";
            break;
        case 2:
            ColorHexVal = "#00FF00";
            break;
        case 3:
            ColorHexVal = "#0000FF";
            break;
        case 4:
            ColorHexVal = "#00FFFF";
            break;
        case 5:
            ColorHexVal = "#FF00FF";
            break;
        case 6:
            ColorHexVal = "#FFFF00";
            break;
        case 7:
            ColorHexVal = "#000000";
            break;
        default:
            ColorHexVal = "#000000";
            break;
    }
    String content =                                                    // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
    "<html>\
    <head>\
    <title>" + title_web + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
```

```
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p>Datum:          " + date_web + "</p>\
    <p>Uhrzeit:        " + time_web + "</p>\
    <p>IP-Addresse:  " + ipaddr_web + "</p>\
    <p>Letzter Name: " + vorname_web + " " + nachname_web + "</p>\
    <form action = 'http://" + ipaddr_web + "/submit' method='POST'>\
    Vorname:  <input type='text' name='vname' value=''><br>\
    Nachname: <input type='text' name='nname' value=''><br>\
    <br>\
    <input type='checkbox' name='unicode' value='chkd' checked> Unicode-Zeichen im Namen erlauben<br>\
    <input type='checkbox' name='display' value='chkd' checked> OLED-Display eingeschaltet<br>\
    <br>\
    <input type='radio' id='led0' name='LEDgroup' value='none' checked> Keinen Ausgang einschalten<br>\
    <input type='radio' id='led1' name='LEDgroup' value='GPIO13'> GPIO 13 einschalten<br>\
    <input type='radio' id='led2' name='LEDgroup' value='GPIO14'> GPIO 14 einschalten<br>\
    <input type='radio' id='led3' name='LEDgroup' value='both'> Beide Ausg&auml;nge einschalten<br>\
    <br>\
    <input type='radio' id='color1' name='COLORgroup' value='red' checked> Rote Schrift<br>\
    <input type='radio' id='color2' name='COLORgroup' value='green'> Gr&uuml;ne Schrift<br>\
    <input type='radio' id='color3' name='COLORgroup' value='blue'> Blaue Schrift<br>\
    <input type='radio' id='color4' name='COLORgroup' value='cyan'> T&uuml;rkise Schrift<br>\
    <input type='radio' id='color5' name='COLORgroup' value='magenta'> Violette Schrift<br>\
    <input type='radio' id='color6' name='COLORgroup' value='yellow'> Gelbe Schrift<br>\
    <input type='radio' id='color7' name='COLORgroup' value='black'> Schwarze Schrift<br>\
    <br>\
    <input type = 'submit' value='Namen im Terminal und auf der Webseite anzeigen, GPIO & Display schalten'><br>\
    </form>\
    </body>\
    </html>";

    IoT_WebServer.send(200, "text/html", content);                  // HTTP-Statuscode 200 = OK (Anfrage wurde erfolgreich bearbeitet)
}

void handleSubmit ()
{
    Serial.println("Button wurde betätigt");
    IoT_WebPrintAllFields();                                        // Alle gefundenen Felder auf dem Terminal ausgeben
    if (IoT_WebTestField("vname"))                                  // Testen, ob ein Feld mit dem Namen "vname" existiert
    {
        vorname_web = IoT_WebGetField("vname");                       // Das Web-Feld "vname" auslesen
    }
    if (IoT_WebTestField("nname"))                                  // Testen, ob ein Feld mit dem Namen "vname" existiert
    {
        nachname_web = IoT_WebGetField("nname");                      // Das Web-Feld "nname" auslesen
    }
    unicodeMode = boolval(IoT_WebTestField("unicode"));             // Testen, ob die Checkbox mit dem Namen "unicode" angekreuzt wurde
    displayMode = boolval(IoT_WebTestField("display"));             // Testen, ob die Checkbox mit dem Namen "display" angekreuzt wurde
    if (IoT_WebTestField("LEDgroup"))                               // Testen, ob Felder mit dem Namen "LEDgroup" existieren
```

```
{
    String selection = IoT_WebGetField("LEDgroup");
    if (selection == "none")                                    // Beide Leitungen aus
    {
        digitalWrite(13, LOW);
        digitalWrite(14, LOW);
        currentGPIO = 0;
    }
    else if (selection == "GPIO13")                             // GPIO 13
    {
        digitalWrite(13, HIGH);
        digitalWrite(14, LOW);
        currentGPIO = 1;
    }
    else if (selection == "GPIO14")                             // GPIO 14
    {
        digitalWrite(13, LOW);
        digitalWrite(14, HIGH);
        currentGPIO = 2;
    }
    else if (selection == "both")                               // Beide Leitungen ein
    {
        digitalWrite(13, HIGH);
        digitalWrite(14, HIGH);
        currentGPIO = 3;
    }
}
if (IoT_WebTestField("COLORgroup"))                            // Testen, ob Felder mit dem Namen "COLORgroup" existieren
{
    String selection = IoT_WebGetField("COLORgroup");
    if (selection == "red")                                     // Rot
    {
        colorMode = 1;
    }
    else if (selection == "green")                             // Grün
    {
        colorMode = 2;
    }
    else if (selection == "blue")                               // Blau
    {
        colorMode = 3;
    }
    else if (selection == "cyan")                               // Türkis
    {
        colorMode = 4;
    }
    else if (selection == "magenta")                            // Violett
    {
        colorMode = 5;
    }
```

```
        else if (selection == "yellow")                                 // Gelb
        {
            colorMode = 6;
        }
        else if (selection == "black")                                  // Schwarz
        {
            colorMode = 7;
        }
    }
    if (unicodeMode == 0)
    {
        vorname_web  = cleanasc(vorname_web);
        nachname_web = cleanasc(nachname_web);
    }
    handleRoot();                                                   // Wieder auf die Hauptseite zurück schalten
}
```

IoT-Brick Set Beispiel 7.3 Listing

Das folgende Programm mit 285 Zeilen ist nicht Bestandteil des IoT-Handbuchs und ist eine Erweiterung zum Programm 7.2 mit zwei neuen Buttons zum Neustarten und Ausschalten des IoT-Bricks. Der eigentliche Versuchsaufbau ist hardwaretchnisch zum 7.1 identisch. In diesem Programm wird wie bei 7.2 beim Starten auf eine Terminalverbindung gewartet. Wenn man nicht auf die Terminalverbindung warten möchte, kann man statt `IoT_TerminalWaitInit(true)` einfach den Befehl `t_TerminalInit()` benutzen.. Der WLAN-Aufbau erfolgt über den Befehl `IoT_WLANautoConnect(true)`, welcher den WiFi-Manager benutzt und eine einmal eingestellte WLAN-Verbindung automatisch wieder aufbaut, falls sich am Name und Password nichts geändert hat. Die Elemente der Webseite werden über Library-Aufrufe erzeugt, was eine bessere Lesbarkeit und mehr Möglichkeiten bei den Parametern bietet.

```
//  Beispiel 7.3 "Erweiterung des Sketches 7.2 um automatischen WLAN-Aufbau"
//               "Darüber hinaus werden Seiten-Elemente mit Funktionen generiert"
//               "und die zuletzt angeklickten Einstellungen bleiben angeklickt."
//               "Weiterhin sind zwei neue Action-Buttons implementiert worden"
//               "um den IoT-Brick neu zu starten und auszuschalten."
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>


int counter = 0;
int currentGPIO = 0;                                    // 0 = Beide aus, 1 = GPIO13 ein, 2 = GPIO14 ein, 3 = Beide ein
int unicodeMode = 1;                                    // 0 = Asc II Namen, 1 = Unicode Namen
int displayMode = 1;                                    // 0 = Display aus,  1 = Display ein
int colorMode = 0;                                      // 0 = Rote Schrift, 1 = Grüne Schrift, 2 = Blaue Schrift usw.

String vorname_web    = "(Vorname)";
String nachname_web   = "(Nachname)";

void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANautoConnect(true);                          // Verbindung, ggf. gespeicherte, über WLAN herstellen
    IoT_NTPinit();

    IoT_WebServer.on("/", handleRoot);                  // Browser-Handler (siehe unten) für das Stammverzeichnis
    IoT_WebServer.on("/form01", handleForm01);          // Handler für die Eingabe der Namen in den Eingabefeldern
    IoT_WebServer.on("/form02", handleForm02);          // Handler für "IoT-Brick neu starten"
    IoT_WebServer.on("/form03", handleForm03);          // Handler für "IoT-Brick ausschalten"
    IoT_WebServer.begin();                              // ab jetzt "horcht" der Server auf HTTP-Anfragen
    Serial.println("HTTP server started");
}
void loop()
```

```
{
    IoT_Idle();

    IoT_WebServer.handleClient();                                       // Bediene die http Anfragen

    if (IoT_Keypress())                                                 // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        if (counter % 10 == 0)                                            // Infos seriell nicht zu oft ausgeben (ca. 1 Sekunde)
        {
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        }
    }
    else
    {
        IoT_Idle();
        if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
        {
            if (IoT_NTPeventAvail)                                        // Zeitevent triggern
            {
                IoT_NTPprintEvent(IoT_NTPcurrentEvent);
                IoT_NTPeventAvail = false;
            }
        }

        IoT_DisplayClear(16);                                           // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                        // Zentrierte Darstellung des Textes
        if (IoT_NTPvalid())                                             // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
        {
            if (displayMode == 1)
            {
                IoT_DisplayDrawText(63, 0, IoT_NTPtime());                 // Uhrzeit im Display anzeigen
                IoT_DisplayDrawText(63, 15, IoT_NTPdate());                // Datum im Display anzeigen
                IoT_Idle();
            }
        }
        else                                                            // "Bitte warten..." zentriert anzeigen
        {
            if (displayMode == 1)
            {
                if (IoT_WLANinited)
                {
                    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
                    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
                    IoT_DisplayDrawText(64, 5, "Bitte");
                    IoT_DisplayDrawText(64, 34, "warten...");
                }
```

```
                else
                {
                    IoT_DisplayClear(24);                                   // OLED Display löschen (24 Punkt Schrift)
                    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                // Zentrierte Darstellung des Textes
                    IoT_DisplayDrawText(64, 5, "Kein");
                    IoT_DisplayDrawText(64, 34, "WLAN");
                }
            }
        }
    }
    IoT_DisplayUpdate();                                                    // OLED-Anzeige aktualisieren
    delay(100);                                                             // 100 ms. warten
    t_TerminalRead();                                                       // Reset-Anforderung prüfen
    counter++;
}

//Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot ()
{
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    String ipaddr_web  = IoT_WLANaddress(false);
    String title_web = "Brick'R'knowledge IoT Brick-Website";
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String ColorHexVal = "";
    switch (colorMode)
    {
        case 0:
            ColorHexVal = "#FF0000";
            break;
        case 1:
            ColorHexVal = "#00FF00";
            break;
        case 2:
            ColorHexVal = "#0000FF";
            break;
        case 3:
            ColorHexVal = "#00FFFF";
            break;
        case 4:
            ColorHexVal = "#FF00FF";
            break;
        case 5:
            ColorHexVal = "#FFFF00";
            break;
        case 6:
            ColorHexVal = "#000000";
```

```
        break;
    default:
        ColorHexVal = "#000000";
        break;
}
String content =                                                    // Die HTML-Seite in lesbarer Formatierung,
// darf auch Variablen enthalten
"<html>\
<head>\
<title>" + title_web + "</title>\
<style>\
body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
</style>\
<style type='text/css'>\
h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
</style>\
</head>\
<body>\
<h1>" + title_web + "</h1>\
<p>Datum:          " + date_web + "</p>\
<p>Uhrzeit:        " + time_web + "</p>\
<p>IP-Addresse:  " + ipaddr_web + "</p>\
<p>Letzter Name: " + vorname_web + " " + nachname_web + "</p>\
" + IoT_WebFormOpen("form01") + "\
" + IoT_WebInput("Vorname:  ", "vname", "", 20) + "<br>\
" + IoT_WebInput("Nachname: ", "nname", "", 20) + "<br>\
<br>\
" + IoT_WebCheckBox(" Unicode-Zeichen im Namen erlauben", "unicode", "chkd", unicodeMode) + "<br>\
" + IoT_WebCheckBox(" OLED-Display eingeschaltet", "display", "chkd", displayMode) + "<br>\
<br>\
" + IoT_WebRadioButton(" Keinen Ausgang einschalten", 0, "LEDgroup", "none",   currentGPIO) + "<br>\
" + IoT_WebRadioButton(" GPIO 13 einschalten",        1, "LEDgroup", "GPIO13", currentGPIO) + "<br>\
" + IoT_WebRadioButton(" GPIO 14 einschalten",        2, "LEDgroup", "GPIO14", currentGPIO) + "<br>\
" + IoT_WebRadioButton(" Beide Ausgänge einschalten", 3, "LEDgroup", "both",   currentGPIO) + "<br>\
<br>\
" + IoT_WebRadioButton(" Rote Schrift",     0, "COLORgroup", "red",     colorMode) + "<br>\
" + IoT_WebRadioButton(" Grüne Schrift",    1, "COLORgroup", "green",   colorMode) + "<br>\
" + IoT_WebRadioButton(" Blaue Schrift",    2, "COLORgroup", "blue",    colorMode) + "<br>\
" + IoT_WebRadioButton(" Türkise Schrift",  3, "COLORgroup", "cyan",    colorMode) + "<br>\
" + IoT_WebRadioButton(" Violette Schrift", 4, "COLORgroup", "magenta", colorMode) + "<br>\
" + IoT_WebRadioButton(" Gelbe Schrift",    5, "COLORgroup", "yellow",  colorMode) + "<br>\
" + IoT_WebRadioButton(" Schwarze Schrift", 6, "COLORgroup", "black",   colorMode) + "<br>\
<br>\
" + IoT_WebFormSubmitButton("Namen im Terminal und auf der Webseite anzeigen, Farbe, GPIO & Display schalten") + "<br>\
" + IoT_WebFormClose() + "\
" + IoT_WebFormActionButton ("form02", "IoT-Brick neu starten") + "\
" + IoT_WebFormActionButton ("form03", "IoT-Brick ausschalten") + "\
</body>\
</html>";
```

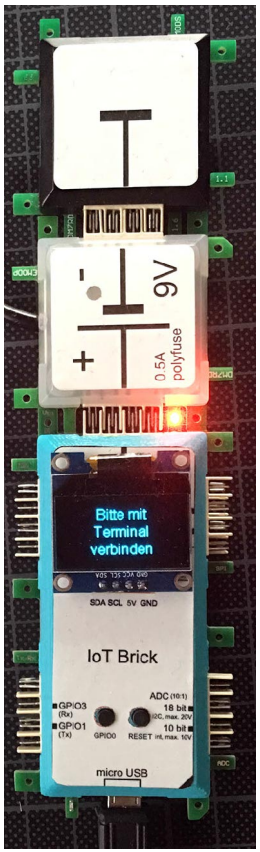
```
    IoT_WebServer.send(200, "text/html", content);                  // HTTP-Statuscode 200 = OK (Anfrage wurde erfolgreich bearbeitet)
}

void handleForm01 ()
{
    Serial.println("Button wurde betätigt: Form01");
    IoT_WebPrintAllFields();                                        // Alle gefundenen Felder auf dem Terminal ausgeben
    if (IoT_WebTestField("vname"))                                  // Testen, ob ein Feld mit dem Namen "vname" existiert
    {
        vorname_web = IoT_WebGetField("vname");                       // Das Web-Feld "vname" auslesen
    }
    if (IoT_WebTestField("nname"))                                  // Testen, ob ein Feld mit dem Namen "vname" existiert
    {
        nachname_web = IoT_WebGetField("nname");                      // Das Web-Feld "nname" auslesen
    }
    unicodeMode = boolval(IoT_WebTestField("unicode"));             // Testen, ob die Checkbox mit dem Namen "unicode" angekreuzt wurde
    displayMode = boolval(IoT_WebTestField("display"));             // Testen, ob die Checkbox mit dem Namen "display" angekreuzt wurde
    if (IoT_WebTestField("LEDgroup"))                               // Testen, ob Felder mit dem Namen "LEDgroup" existieren
    {
        String selection = IoT_WebGetField("LEDgroup");
        if (selection == "none")                                      // Beide Leitungen aus
        {
            digitalWrite(13, LOW);
            digitalWrite(14, LOW);
            currentGPIO = 0;
        }
        else if (selection == "GPIO13")                               // GPIO 13
        {
            digitalWrite(13, HIGH);
            digitalWrite(14, LOW);
            currentGPIO = 1;
        }
        else if (selection == "GPIO14")                               // GPIO 14
        {
            digitalWrite(13, LOW);
            digitalWrite(14, HIGH);
            currentGPIO = 2;
        }
        else if (selection == "both")                                 // Beide Leitungen ein
        {
            digitalWrite(13, HIGH);
            digitalWrite(14, HIGH);
            currentGPIO = 3;
        }
    }
    if (IoT_WebTestField("COLORgroup"))                             // Testen, ob Felder mit dem Namen "COLORgroup" existieren
    {
        String selection = IoT_WebGetField("COLORgroup");
        if (selection == "red")                                       // Rot
        {
```

```
            colorMode = 0;
        }
        else if (selection == "green")                                  // Grün
        {
            colorMode = 1;
        }
        else if (selection == "blue")                                   // Blau
        {
            colorMode = 2;
        }
        else if (selection == "cyan")                                   // Türkis
        {
            colorMode = 3;
        }
        else if (selection == "magenta")                                // Violett
        {
            colorMode = 4;
        }
        else if (selection == "yellow")                                 // Gelb
        {
            colorMode = 5;
        }
        else if (selection == "black")                                  // Schwarz
        {
            colorMode = 6;
        }
    }
    if (unicodeMode == 0)
    {
        vorname_web  = cleanasc(vorname_web);
        nachname_web = cleanasc(nachname_web);
    }
    handleRoot();                                                       // Wieder auf die Hauptseite zurück schalten
}

void handleForm02 ()
{
    Serial.println("Button wurde betätigt: Form02");
    handleRoot();                                                       // Wieder auf die Hauptseite zurück schalten
    t_Restart();
}

void handleForm03 ()
{
    Serial.println("Button wurde betätigt: Form03");
    handleRoot();                                                       // Wieder auf die Hauptseite zurück schalten
    IoT_ShutDown();
}
```

IoT-Brick Set Beispiel 8 Listing

Das folgende Programm mit 65 Zeilen ist nicht Bestandteil des IoT-Handbuchs und berechnet die Rechenleistung des IoT-Bricks. Diese beträgt ca. 1,4 Mio. Fließkomma-Additionen pro Sekunde. In diesem Programm wird beim Starten auf eine Terminalverbindung gewartet, damit man den kompletten Testverlauf beobachten kann und nichts verloren geht. Eine besondere Herausforderung stellt dabei der Watch-Dog des IoT-Bricks dar, der bei Berechnungen, die mehr als eine Sekunde benötigen, was hier der Fall ist, einen Reset auslöst. Aus diesem Grund wird hier der Watch-Dog ausgeschaltet. Wenn man für die Terminalverbindung den seriellen Monitor der Arduino IDE benutzen möchte, muss man zum Starten der Terminal-Verbindung 5 mal „t“ eingeben, also „ttttt“ und dann auf den „Senden“-Button klicken. Mit dem Programm „UniTerminal“ muss man lediglich eine Verbindung herstellen.

```
Welcome to IoT Brick GFLOP Test                                    (c) 2015-2017
------------------------------------------------------------------------------

CPU ESP 8266 (ID 10), 32 Bit, 4.096 kB Flash-RAM, 96 kB SRAM

Date : 29.05.2017 (Montag)
Time : 12:51:14

Der GFLOP Test berechnet die Anzahl an Fließkomma-Additionen, die der IoT-Brick
pro Sekunde berechnen kann. Mit jeder Zeile werden mehr Additionen berechnet,
wodurch eine höhere Genauigkeit erreicht wird. Das Maximum sind 10 Mio. Fließ-
komma-Additionen. Der IoT-Brick berechnet ca. 1,4 Mio. Additionen pro Sekunde.
Arduino Nano, Uno oder Mega 2560 berechnen ca. 0,1 Mio. Additionen pro Sekunde.

10 Schritte - benötigte Zeit = 0 ms. = 769.230 FLOPs.
100 Schritte - benötigte Zeit = 0 ms. = 1.388.888 FLOPs.
1.000 Schritte - benötigte Zeit = 0 ms. = 1.424.501 FLOPs.
10.000 Schritte - benötigte Zeit = 7 ms. = 1.427.551 FLOPs.
100.000 Schritte - benötigte Zeit = 70 ms. = 1.428.448 FLOPs.
1.000.000 Schritte - benötigte Zeit = 700 ms. = 1.428.410 FLOPs.
10.000.000 Schritte - benötigte Zeit = 7.317 ms. = 1.366.619 FLOPs.

Fertig.
```



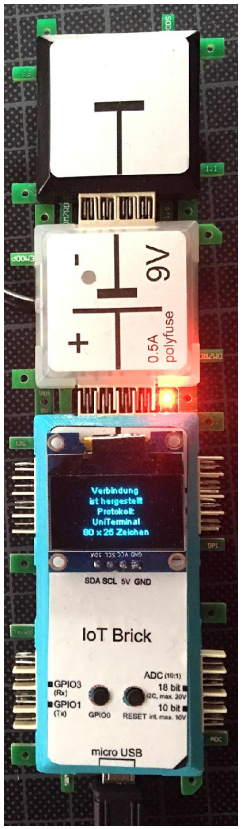
```
//  Beispiel 8 "Berechnung der Fließkomma-Rechenleistung"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

float a = 0.0, b = 3.0, Zeit = 0.0;
long Start = 0, Ende = 0, Groesse = 1;

void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);      // Auf Terminalverbindung warten, damit nichts verloren geht
    IoT_WatchDog(false);             // Watchdog ausschalten, damit der Benchmark kein Reset auslöst
    Serial.println("");
    Serial.print(chr(12));
    Serial.println("Welcome to IoT Brick GFLOP Test" + spc(35) + "(c) 2015-2017");
    Serial.println(repeatstr("-", 79) + chr(13));
    Serial.print("CPU " + t_CPUname());
    Serial.print(" (ID " + str(t_CPUtype()) + "), ");
    Serial.print(str(t_CPUbits()) + " Bit, ");
    Serial.print(str(t_CPUflashRAM()) + " kB Flash-RAM, ");
    Serial.println(str(t_CPUstaticRAM()) + " kB SRAM" + char(13));
    t_DateTime DateTime = t_CompileDateTime();        // Uhrzeit, wann der Sketch kompiliert wurde
    byte dayOfWeek = dayofweek(DateTime.year, DateTime.month, DateTime.day);
    Serial.println("Date : " + datestr(DateTime)+" (" + dayname(dayOfWeek)+")");
    Serial.println("Time : " + timestr(DateTime));
    Serial.println("");//....1....!....2....!....3....!....4....!....5....!....6....!....7....!....8
    Serial.println("Der GFLOP Test berechnet die Anzahl an Fließkomma-Additionen, die der IoT-Brick");
    Serial.println("pro Sekunde berechnen kann. Mit jeder Zeile werden mehr Additionen berechnet,");
    Serial.println("wodurch eine höhere Genauigkeit erreicht wird. Das Maximum sind 10 Mio. Fließ-");
    Serial.println("komma-Additionen. Der IoT-Brick berechnet ca. 1,4 Mio. Additionen pro Sekunde.");
    Serial.println("Arduino Nano, Uno oder Mega 2560 berechnen ca. 0,1 Mio. Additionen pro Sekunde.");
    Serial.println("");
    while (Zeit < 5000000)
    {
        Groesse = Groesse * 10;
        Start = micros();
        a = 0.0;
        for (long i = 1; i <= Groesse; i++)
        {
            a = a + b;
        }
        Ende = micros();
        Zeit = Ende - Start;
        if (Zeit > 0)
        {
            Serial.print(strform(Groesse, 32, 1, true));
            Serial.print(" Schritte - benötigte Zeit = ");
            Serial.print(strform(Zeit / 1000, 32, 1, true));
```

```
            Serial.print(" ms. = ");
            Serial.print(strform(Groesse * 1000.0 / Zeit * 1000.0, 32, 1, true));
            Serial.println(" FLOPs.");
            delay(10);
        }
    }
    String S = String(a); // Verhindert, dass die for-Schleife wegoptimiert wird
    Serial.println("");
    Serial.println("Fertig.");
}

void loop()
{
    IoT_ShutDown(); // Brick incl. OLED-Display ausschalten
}

// Arduino Nano ( 16 MHz,  8 Bit) =   104.024 FLOPs ( 1 Mio. Schritte)
// Arduino Mega ( 16 MHz,  8 Bit) =   100.717 FLOPs ( 1 Mio. Schritte)
// Arduino Due  ( 84 MHz, 32 Bit) =   822.665 FLOPs (10 Mio. Schritte)
// Teensy 3.1   ( 96 MHz, 32 Bit) = 1.408.183 FLOPs (10 Mio. Schritte)
// Teensy 3.1   (120 MHz, 32 Bit) = 1.709.976 FLOPs (10 Mio. Schritte)
// Teensy 3.1   (144 MHz, 32 Bit) = 1.690.433 FLOPs (10 Mio. Schritte)
// ESP 8266     ( 80 MHz, 32 Bit) = 1.428.530 FLOPs ( 1 Mio. Schritte)
```

IoT-Brick Set Beispiel 9 Listing

Das folgende Programm mit 31 Zeilen ist nicht Bestandteil des IoT-Handbuchs und zeigt eine einzeilige Eingabe eines Textes über das Terminal. Die crc64-Prüfsumme (hash-Wert) ist so lange im Klartext lesbar, wie der eingegebene Text nicht länger als 8 Zeichen ist. Wenn man für die Terminalverbindung den seriellen Monitor der Arduino IDE benutzen möchte, muss man zum Starten der Terminal-Verbindung 5 mal „t“ eingeben, also „ttttt“ und dann auf den „Senden“-Button klicken. Mit dem Programm „UniTerminal“ muss man lediglich eine Verbindung herstellen.

```
Willkommen zum IoT Brick Terminal-Test                              (c) 2017
--------------------------------------------------------------------------------

Geben Sie ‘ende‘ oder ‘quit‘ ein, um dieses Beispiel zu beenden.

Bitte einen Text eingeben: BRKclock

Ihre Eingabe war “BRKclock“ und besteht aus 8 Zeichen.

Die crc64-Prüfsumme (hash-Wert) des Textes ist = 42524B636C6F636B
Die xorshift128plus-Prüfsumme (hash-Wert) des Textes ist = 6D9A3009904C03C4

Bitte einen Text eingeben: Das ist ein ganz langer Text.

Ihre Eingabe war “Das ist ein ganz langer Text.“ und besteht aus 29 Zeichen.

Die crc64-Prüfsumme (hash-Wert) des Textes ist = 17F35852AB7675E2
Die xorshift128plus-Prüfsumme (hash-Wert) des Textes ist = 3D5D29A3953E112A

Bitte einen Text eingeben:
```



```
//  Beispiel 9 "Texteingabe im Terminal-Programm"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);      // Auf Terminalverbindung warten, damit nichts verloren geht
    Serial.println("");
    Serial.print(chr(12));
    Serial.println("Willkommen zum IoT Brick Terminal-Test" + spc(33) + "(c) 2017");
    Serial.println(repeatstr("-", 79) + chr(13));
    Serial.println("Geben Sie 'ende' oder 'quit' ein, um dieses Beispiel zu beenden.");
    Serial.println("");
}

void loop()
{
    String s = t_TerminalInput ("Bitte einen Text eingeben: ", 50, t_Input_String);
    Serial.println("");
    Serial.println("");
    Serial.print("Ihre Eingabe war " + chr(34) + s + chr(34));
    Serial.println(" und besteht aus " + str(len(s)) + " Zeichen.");
    Serial.println("");
    Serial.print("Die crc64-Prüfsumme (hash-Wert) des Textes ist = ");
    Serial.println(hex(crc64(s, 0)));
    Serial.print("Die xorshift128plus-Prüfsumme (hash-Wert) des Textes ist = ");
    Serial.println(hex(xorshift128plus(s, 0)));
    Serial.println("");
    if ((lcase(s) == "ende") || (lcase(s) == "quit"))
    {
        IoT_ShutDown(); // Display ausschalten und Bearbeitung beenden
    }
}
```

IoT-Brick Set Beispiel 10 Listing

Das folgende Programm mit 42 Zeilen ist nicht Bestandteil des IoT-Handbuchs und zeigt eine Laufschrift auf der BRK-Clock.

```
//  Beispiel 10 "Laufschrift mit 64 programmierbaren RGB-Bricks (8 x 8)"
//
//  Unter Verwendung der IoT Brick Library und der BRK Clock Library

#include <all_IoT.h>
#include <all_BRKclock.h>

void setup()
{
    IoT_Init(true);                                          // IoT-Brick initialisieren
    t_TerminalInit();                                        // Terminal initialisieren
    BRKclock_Init();                                         // BRK-Clock initialisieren
    BRKclock_TestMatrix(50);                                 // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(10000, true);                           // 10 Sekunden warten, kann mit Taster übersprungen werden
    BRKclock_Matrix.begin();                                 // Ausgabe auf der Matrix beginnen
    BRKclock_Matrix.setTextWrap(false);                      // Kein Zeilenumbruch
    BRKclock_Matrix.setTextSize(1);                          // Normale Textgröße, keine Vergrößerung
    BRKclock_Matrix.setBrightness(100);                      // Helligkeit 100%
}

int x = BRKclock_Matrix.width();                             // Breite der Matrix (8)
int pass = 1;

void loop()
{
    String letters = "Brick'R'knowledge";
    int L = (len(letters) - 1) * 8;                          // Breite der Laufschrift insgesamt in Pixel
    BRKclock_Matrix.setTextColor(BRKclock_Matrix8Color[pass]);  // Indizierte Farbe aus 8 Farben auswählen
    BRKclock_Matrix.fillScreen(0);                           // Alle Pixel ausschalten
    BRKclock_Matrix.setCursor(x, 0);                         // Von unten rechts reinlaufen lassen, x verringert sich im loop()
    BRKclock_Matrix.print(letters);                          // Laufschrift "Brick'R'knowledge" loslassen...
    if (--x < -L)
    {
        x = BRKclock_Matrix.width();
        if (++pass > 7)                                      // 7 Durchgänge mit 7 Farben, danach wieder von vorne
        {
            pass = 1;
        }
    }
    BRKclock_Matrix.show();
    delay(100);                                              // 100 ms. warten
}
```

IoT-Brick Set Beispiel 11 Listing

Das folgende Programm mit 41 Zeilen ist nicht Bestandteil des IoT-Handbuchs und zeigt farbige Rechtecke in 16 Farben auf der BRK-Clock.

```
//  Beispiel 11 "Farbige Rechtecke mit 64 programmierbaren RGB-Bricks (8 x 8)"
//
//  Unter Verwendung der IoT Brick Library und der BRK Clock Library

#include <all_IoT.h>
#include <all_BRKclock.h>

void setup()
{
    IoT_Init(true);                                         // IoT-Brick initialisieren
    BRKclock_Init();                                        // BRK-Clock initialisieren
    BRKclock_TestMatrix(50);                                // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(10000, true);                          // 10 Sekunden warten, kann mit Taster übersprungen werden
    BRKclock_Matrix.begin();                                // Ausgabe auf der Matrix beginnen
    BRKclock_Matrix.setTextWrap(false);                     // Kein Zeilenumbruch
    BRKclock_Matrix.setTextSize(1);                         // Normale Textgröße, keine Vergrößerung
    BRKclock_Matrix.setBrightness(100);                     // Helligkeit 100%
    BRKclock_Matrix.fillScreen(0);                          // Alle Pixel ausschalten
}

int i = 0;

void loop()
{
    i++;
    BRKclock_Matrix.drawRect(0, 0, 8, 8, BRKclock_Matrix16Color[i & 15]);
    BRKclock_Matrix.show();
    delay(500);                                             // Eine halbe Sekunde warten
    i++;
    BRKclock_Matrix.drawRect(1, 1, 6, 6, BRKclock_Matrix16Color[i & 15]);
    BRKclock_Matrix.show();
    delay(500);                                             // Eine halbe Sekunde warten
    i++;
    BRKclock_Matrix.drawRect(2, 2, 4, 4, BRKclock_Matrix16Color[i & 15]);
    BRKclock_Matrix.show();
    delay(500);                                             // Eine halbe Sekunde warten
    i++;
    BRKclock_Matrix.drawRect(3, 3, 2, 2, BRKclock_Matrix16Color[i & 15]);
    BRKclock_Matrix.show();
    delay(500);                                             // Eine halbe Sekunde warten
}
```

IoT-Brick Set Beispiel 12 Listing

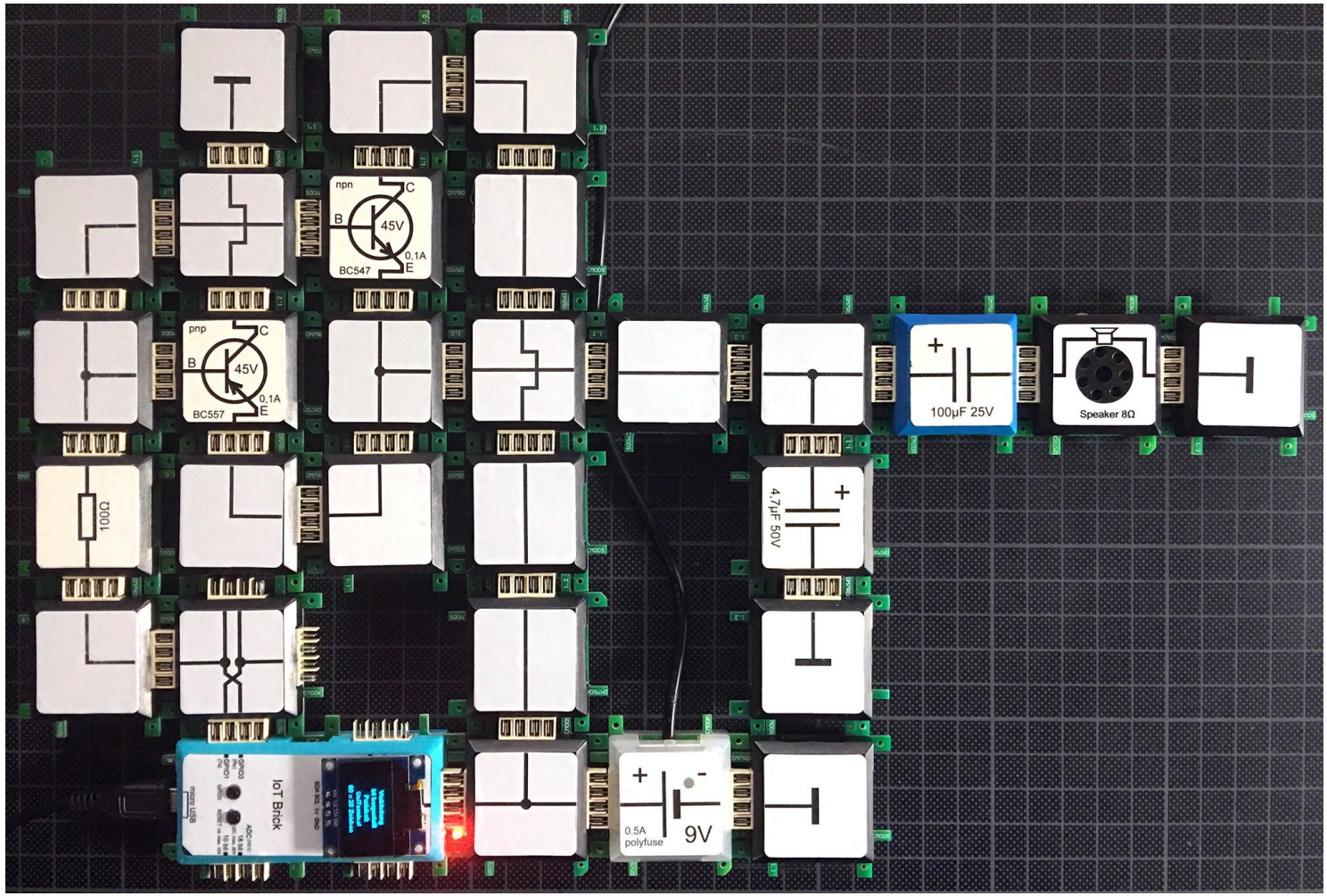
Das folgende Programm mit 72 Zeilen ist nicht Bestandteil des IoT-Handbuchs und zeigt ein farbiges Kaleidoskop in wahlweise 8 oder 16 Farben auf der BRK-Clock. Man kann das Programm modifizieren, dass nur 8 (besser unterscheidbare) Farben benutzt werden. Dafür ist einfach die Variable `color16` auf `false` zu setzen. Dadurch wird das Kaleidoskop aber weniger facettenreich.

```
//  Beispiel 12 "Kaleidoscope mit 64 programmierbaren RGB-Bricks (8 x 8)"
//
//  Unter Verwendung der IoT Brick Library und der BRK Clock Library

#include <all_IoT.h>
#include <all_BRKclock.h>

bool color16 = true;

void Kaleidoscope (int size)
{
    long t = millis();
    size--;
    for (int w = 0; w <= size * 10; w++)
    {
        for (int i = 0; i <= (size / 2); i++)
        {
            for (int j = 0; j <= (size / 2); j++)
            {
                int k = i + j;
                int color = j * 3 / (i + 3) + (i + 1) * w * 3;
                Serial.print(str(color) + ",");
                if (color16)
                {
                    BRKclock_Plot16color(i, k, color);
                    BRKclock_Plot16color(k, i, color);
                    BRKclock_Plot16color(size - i, size - k, color);
                    BRKclock_Plot16color(size - k, size - i, color);
                    BRKclock_Plot16color(k, size - i, color);
                    BRKclock_Plot16color(size - i, k, color);
                    BRKclock_Plot16color(i, size - k, color);
                    BRKclock_Plot16color(size - k, i, color);
                }
                else
                {
                    BRKclock_Plot8color(i, k, color);
                    BRKclock_Plot8color(k, i, color);
                    BRKclock_Plot8color(size - i, size - k, color);
                    BRKclock_Plot8color(size - k, size - i, color);
                    BRKclock_Plot8color(k, size - i, color);
                    BRKclock_Plot8color(size - i, k, color);
```

```
                    BRKclock_Plot8color(i, size - k, color);
                    BRKclock_Plot8color(size - k, i, color);
                }
                BRKclock_Matrix.show();
                delay(50);
            }
        }
    }
    Serial.println("");
    Serial.println("");
    Serial.println("Ende eines Durchlaufs (" + str(millis() - t) + " ms.)");
    Serial.println("");
}

void setup()
{
    IoT_Init(true);                                         // IoT-Brick initialisieren
    BRKclock_Init();                                        // BRK-Clock initialisieren
    BRKclock_TestMatrix(50);                                // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(10000, true);                          // 10 Sekunden warten, kann mit Taster übersprungen werden
    BRKclock_Matrix.begin();                                // Ausgabe auf der Matrix beginnen
    BRKclock_Matrix.setTextWrap(false);                     // Kein Zeilenumbruch
    BRKclock_Matrix.setTextSize(1);                         // Normale Textgröße, keine Vergrößerung
    BRKclock_Matrix.setBrightness(100);                     // Helligkeit 100%
    BRKclock_Matrix.fillScreen(0);                          // Alle Pixel ausschalten
}

void loop()
{
    Kaleidoscope(8);
}
```

IoT-Brick Set Beispiel 13 Listing

Das folgende Programm mit 23 Zeilen ist nicht Bestandteil des IoT-Handbuchs und erzeugt jede Minute einen 1 kHz Beep für eine halbe Sekunde und jede Sekunde einen 1 kHz Click für 5 Millisekunden. Für die Tonausgabe sollte immer eine Verstärkerschaltung wie diese hier verwendet werden. Ein direkter Anschluß des Lautsprechers kann diesen beschädigen und sollte daher vermieden werden.



```
//  Beispiel 13 "Sound-Ausgabe"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <all_Sound.h>

void setup()
{
    IoT_Init(true);                                     // IoT-Brick initialisieren
    snd_Init(3);                                        // GPIO 3 (Rx) für Sound-Ausgabe wählen
}

void loop()
{
    snd_beep();
    delay(1000);                                        // 1 Sekunde warten
    for (int i = 0; i < 59; i++)
    {
        snd_click();
        delay(1000);                                    // 1 Sekunde warten
    }
}
```

IoT-Brick Set Beispiel 14 Listing

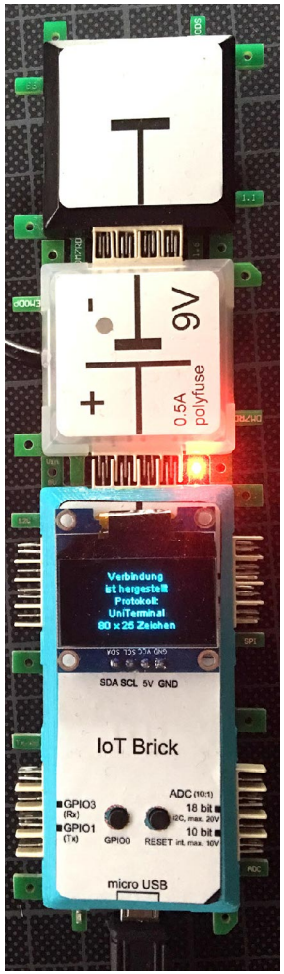
Das folgende Programm mit 52 Zeilen ist nicht Bestandteil des IoT-Handbuchs und berechnet 65.536 Zufallszahlen zwischen 0 und 255. Dabei wird statistisch ausgewertet, wie oft jede der 256 möglichen Zufallszahlen (0-255) tatsächlich auftritt. Rein theoretisch müsste der Wert jeweils 256 mal sein. In Wirklichkeit schwankt der Wert aber um den Wert 256 (Hexadezimal $0100) herum, weil der theoretische Wert erst bei sehr viel mehr Zufallszahlen exakt erreicht werden würde.

Durch die Verwendung des Microsekunden-Timers, bei der Erzeugung der Zufallszahlen, entsteht bei jedem Aufruf des Programms nicht nur eine andere Folge von Zufallszahlen, sondern auch eine andere statistische Verteilung derselben. Die mit den Befehlen `rndbyte()` oder `rnd()` aus der String-Library erzeugten Zufallszahlen sind dabei kryptographisch wesentlich sicherer, als die standardmäßige Reihe von Zufallszahlen, mit der eingebauten `random()` Funktion, welche sich zum Einen irgendwann wiederholende Zufallszahlen erzeugt und zum Anderen bei jedem Start des Programms dieselbe Reihe von Zufallszahlen erzeugt.

Zufallszahlen werden an verschiedenen Stellen benötigt. Dabei ist die Qualität von Zufallszahlen, besonderns bei kryptographischen Anwendungen, extrem wichtig. Andererseite sollen Zufallszahlen natürlich auch tatsächlich zufällig sein, was die statistische Häufung von bestimmten einzelnen Zahlen betrifft. Dabei muss bei jedem Programmstart eine komplett andere Folge von Zufallszahlen erzeugt werden. Das alles leisten die Befehle aus der String-Library. Möchte man eine noch sicherere Zufallszahl erhalten, so muss man die Funktion `rndbytesecure()` benutzen. Diese Berechnet eine Zufallszahl (0-255) mittels 32 Zufallszahlen, die mit `rndbyte()` erzeugt und über eine 126 Bit Polynomfunktion verbessert wurden.

Wenn man für die Terminalverbindung den seriellen Monitor der Arduino IDE benutzen möchte, muss man zum Starten der Terminal-Verbindung 5 mal „t“ eingeben, also „ttttt“ und dann auf den „Senden“-Button klicken. Mit dem Programm „UniTerminal“ muss man lediglich eine Verbindung herstellen.

```
Welcome to IoT Brick Random-Number-Generator                    (c) 2015-2017
------------------------------------------------------------------------------

Der Random-Number-Generator berechnet 65536 richtige Zufallszahlen zwischen 0
und 255. Dabei wird die Häufigkeit, mit der jede der 256 möglichen Zahlen auf-
tritt, statistisch ausgewertet. Die Berechnung der Zufallszahlen bezieht dabei
den Microsekunden-Timer mit ein, wodurch bei jedem Programmstart immer neue,
nicht reprozierbare Zufallszahlen entstehen. Statistische Verteilung:

010F 0129 00EC 0108 011C 0100 00F0 0113 00F1 0113 0102 010A 00FC 012A 00F3 0105
00EE 0107 0102 0103 00FF 010E 0115 0108 0112 00EF 011A 010E 010C 0113 00EF 00FA
0113 0112 0116 00EC 0105 0109 0107 00F3 00F0 010E 011F 00E6 0113 0111 010F 010A
010E 0101 00F3 010C 00E7 00F4 00F2 00EF 00FA 00FF 011A 00F9 0116 010F 010E 00EA
00FA 00F8 00F9 00FD 0110 00FA 0120 010F 00F8 0102 0117 00D1 00ED 00E8 010D 00F9
00F1 0110 0118 0107 0102 00F2 00D9 00EB 010F 0109 0101 00E5 00FB 00E4 00EF 0106
010A 0107 00FE 010C 0111 00E8 00E7 010B 0110 0100 0115 0106 00F5 00F0 00F4 0101
010E 0105 00FC 00F3 0113 0105 00FE 00EC 00F1 00F6 0104 00E7 010E 0104 00EB 0103
0112 0110 00E9 0101 010E 00FB 010D 0109 00F0 0109 0101 00F9 010B 00E9 00FF 010F
0115 0109 00F0 00F5 0102 010A 0105 00F5 00ED 00ED 010D 0112 00F3 0104 0119 0106
0114 0101 010A 0105 0112 010E 010D 00F9 00F8 00ED 010F 00FD 010D 00F5 00FD 00F5
011E 010B 0101 0103 00F8 00E8 0102 0101 0103 00FC 00F1 00E4 0109 00E9 00EB 0112
0102 00FB 00E4 00E8 00FF 0109 010E 00FD 00FA 00F2 010D 00EA 00F4 00FD 00F9 0107
0100 00FF 010D 0109 00EA 0115 00EE 010F 00E6 00F5 00FA 00E8 00D9 0100 0102 00F2
00FB 0104 010A 00FD 00ED 0103 00FA 011B 0108 0119 00FD 00EF 0104 010D 010B 00F6
011E 0102 010E 0104 00DB 00E6 00EA 0108 00F5 00E5 00F5 0114 00FF 00F6 0105 00F6
```

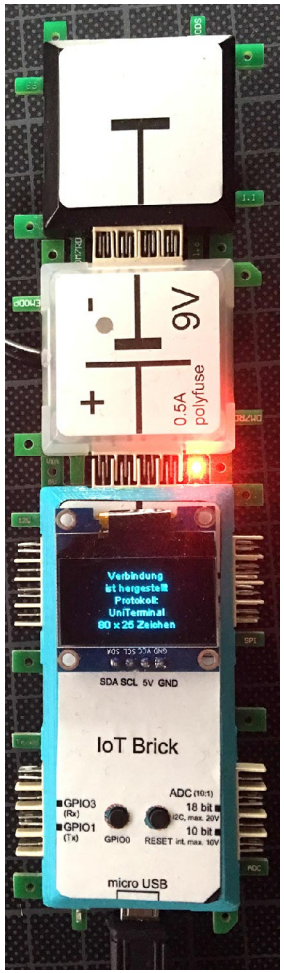


```
//  Beispiel 14 "Berechnung von richtigen, nicht reprozierbaren Zufallszahlen"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

float a = 0.0, b = 3.0, Zeit = 0.0;
long Start = 0, Ende = 0, Groesse = 1;
long stat[256];

void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);      // Auf Terminalverbindung warten, damit nichts verloren geht
    IoT_WatchDog(false);
    Serial.println("");
    Serial.print(chr(12));
    Serial.println("Welcome to IoT Brick Random-Number-Generator" + spc(22) + "(c) 2015-2017");
    Serial.println(repeatstr("-", 79));
    Serial.println("");//....1....!....2....!....3....!....4....!....5....!....6....!....7....!....8
    Serial.println("Der Random-Number-Generator berechnet 65536 richtige Zufallszahlen zwischen 0");
    Serial.println("und 255. Dabei wird die Häufigkeit, mit der jede der 256 möglichen Zahlen auf-");
    Serial.println("tritt, statistisch ausgewertet. Die Berechnung der Zufallszahlen bezieht dabei");
    Serial.println("den Microsekunden-Timer mit ein, wodurch bei jedem Programmstart immer neue,");
    Serial.println("nicht reprozierbare Zufallszahlen entstehen. Statistische Verteilung:");
    Serial.println("");

    for (int i = 0; i <= 255; i++)
    {
        stat[i] = 0;
    }
    for (long i = 0; i <= 0xFFFF; i++)
    {
        byte r = rndbyte();
        stat[r] += 1;
    }
    for (int i = 0; i <= 255; i++)
    {
        Serial.print(hexword(stat[i]));
        if (i < 255)
        {
            Serial.print(" ");
        }
    }
}

void loop()
{
    IoT_ShutDown();
}
```

IoT-Brick Set Beispiel 15 Listing

Das folgende Programm mit 82 Zeilen ist nicht Bestandteil des IoT-Handbuchs. Es schreibt und liest Werte des EEPROMs. Die zuletzt geschriebenen, alten Werte (100) werden am Anfang angezeigt, danach werden neue Werte (142) geschrieben. Wenn man jetzt die Reset-Taste auf dem IoT-Brick drückt, dann stehen die zuvor geschriebenen Werte (dann 142) ganz oben als die alten, neuen Werte.

```
Welcome to IoT Brick EEPROM Test                                    (c) 2015-2017
--------------------------------------------------------------------------------

Alte Werte aus dem EEPROM lesen...

Byte = 100
Word = 12.340
Long = 1.697.736.035

EEPROM löschen...
EEPROM lesen und überprüfen...
EEPROM schreiben...

Byte = 142
Word = 42.763
Long = 3.064.830.149

EEPROM lesen...

Byte = 142
Word = 42.763
Long = 3.064.830.149

Bitte die Reset-Taste auf dem IoT-Brick drücken und die Werte vergleichen.
```



```
//  Beispiel 15 "EEPROM Benutzung für den EPS8266"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>


void setup()
{
    uint8_t  b = 0;
    uint16_t i = 0;
    uint32_t l = 0;

    IoT_Init(true);
    IoT_TerminalWaitInit(true);      // Auf Terminalverbindung warten, damit nichts verloren geht
    IoT_WatchDog(false);
    Serial.println("");
    Serial.print(chr(12));
    Serial.println("Welcome to IoT Brick EEPROM Test" + spc(34) + "(c) 2015-2017");
    Serial.println(repeatstr("-", 79));
    Serial.println("");//....1....!....2....!....3....!....4....!....5....!....6....!....7....!....8

    Serial.println("Alte Werte aus dem EEPROM lesen...");
    Serial.println("");
    EEPROM.get(0, b);
    EEPROM.get(4, i);
    EEPROM.get(8, l);
    Serial.println("Byte = " + str(b));
    Serial.println("Word = " + str(i));
    Serial.println("Long = " + str(l));

    Serial.println("");
    Serial.println("EEPROM löschen...");
    IoT_EEPROMclear();

    Serial.println("EEPROM lesen und überprüfen...");
    EEPROM.get(0, b);
    EEPROM.get(4, i);
    EEPROM.get(8, l);
    if ((b != 0) || (i != 0) || (l != 0))
    {
        Serial.println("");
        Serial.println("Falsche Werte im EEPROM:");
        Serial.println("Byte = " + str(b));
        Serial.println("Word = " + str(i));
        Serial.println("Long = " + str(l));
        Serial.println("");
    }

    Serial.println("EEPROM schreiben...");
```

```
    Serial.println("");
    b = rnd(255);
    i = rnd(65535);
    l = rnd(4294967295);
    Serial.println("Byte = " + str(b));
    Serial.println("Word = " + str(i));
    Serial.println("Long = " + str(l));
    Serial.println("");
    EEPROM.put(0, b);
    EEPROM.put(4, i);
    EEPROM.put(8, l);

    b = 0;
    i = 0;
    l = 0;

    Serial.println("EEPROM lesen...");
    Serial.println("");
    EEPROM.get(0, b);
    EEPROM.get(4, i);
    EEPROM.get(8, l);
    Serial.println("Byte = " + str(b));
    Serial.println("Word = " + str(i));
    Serial.println("Long = " + str(l));
    Serial.println("");
    Serial.println("Bitte die Reset-Taste auf dem IoT-Brick drücken und die Werte vergleichen.");
}

void loop()
{
    IoT_ShutDown(); // Daten in das EEPROM zurückschreibe
}
```

IoT-Brick Set Beispiel 16.1 Listing

Das folgende Programm mit 122 Zeilen ist nicht Bestandteil des IoT-Handbuchs. Es stellt eine MQTT-Verbindung mit insgesamt vier verschiedenen Servern her. Der Allnet-MQTT-Server wird dabei zusätzlich auch zwei verschiedene Arten angesteuert, einmal mit der URL (iot.allnet.de) und einmal mit der IP-Adresse (212.18.29.166) des Servers. Vom Allnet-MQTT-Server werden verschiedene Werte wie Temperatur in einigen deutschen Großstädten, DAX-Kurs, Gold-Kurs etc. geladen und auf dem terminal ausgegeben.

```
*WM:
*WM: AutoConnect
*WM: Connecting as wifi client...
*WM: Using last saved values, should be faster
*WM: Connection result:
*WM: 3
*WM: IP Address:
*WM: 192.168.1.23
Connected to AirPort Extreme, IP: 192.168.1.23
Looking for MQTT server ‘broker.mqtt-dashboard.com‘... Connected.
Looking for MQTT server ‘broker.hivemq.com‘... Connected.
Looking for MQTT server ‘mqtt.mydevices.com‘... Connected.
Looking for MQTT server ‘212.18.29.166‘... Connected.
Looking for MQTT server ‘iot.allnet.de‘... Connected.
.
```

```
//  Beispiel 16.1: MQTT-Test
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <all_MQTT.h>


int counter = 0;


void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANautoConnect(true);                                  // Verbindung, ggf. gespeicherte, über WLAN herstellen
    IoT_NTPinit();                                              // NTP-Timeserver initialisieren

    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
    IoT_DisplayDrawText(64, 5, "Suche mqtt");
    IoT_DisplayDrawText(64, 34, "Server...");
    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren

    MQTT_Init("broker.mqtt-dashboard.com", 1883, newuniqueid(), "", "");
    MQTT_Connect("test");

    MQTT_Init("broker.hivemq.com", 1883, "BrickRknowledge", "", "");
    MQTT_Connect("alpha/a");

    MQTT_Init("mqtt.mydevices.com", 1883, "82410050-608e-11e7-a041-7d8fa03d7b64",
              "9d7e33d0-a71e-11e6-839f-8bfd46afe676", "c4f99459f8f214cf1bc0c600d22e52c661157022");
    MQTT_Connect("general/data/#");

    MQTT_Init(IPval("212.18.29.166"), 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
    MQTT_Connect("general/data/#");

    MQTT_Init("iot.allnet.de", 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
    MQTT_Connect("general/data/#");
}

void loop()
{
    IoT_Idle();

    if (IoT_Keypress())                                         // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        IoT_DisplayUpdate();                                      // OLED-Anzeige aktualisieren
        Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
```

```
    Serial.print("WiFi is ");
    Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
    Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
    IoT_WaitNoKeypress();
}
else
{
    IoT_Idle();
    if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
    {
        if (IoT_NTPeventAvail)                                          // Zeitevent triggern
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }

    IoT_DisplayClear(16);                                             // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                          // Zentrierte Darstellung des Textes
    if (IoT_WLANinited)
    {
        IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
        if (MQTT_Client.connected())
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                      // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Verbunden");
            IoT_DisplayDrawText(64, 34, "mit mqtt");
        }
        else
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                      // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Suche mqtt");
            IoT_DisplayDrawText(64, 34, "Server...");
        }
        IoT_DisplayUpdate();                                            // OLED-Anzeige aktualisieren
    }
    else
    {
        IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                        // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Kein");
        IoT_DisplayDrawText(64, 34, "WLAN");
    }
}
IoT_DisplayUpdate();                                                // OLED-Anzeige aktualisieren

bool success = MQTT_HandleClient();                                 // Verbindung aufrecht erhalten, ggf. neu aufbauen

if (MQTT_EventAvail)
{
```

```
        MQTT_GetNextEvent();
        Serial.println("");
        Serial.print("Time: "      + cleanasc(String(MQTT_EventCurrentTime)));
        Serial.println(", Topic: " + cleanasc(String(MQTT_EventCurrentTopic)));
        Serial.println("Message: " + String(MQTT_EventCurrentMessage));
    }
    else
    {
        if (MQTT_Client.connected())
        {
            Serial.print(".");
        }
        else
        {
            Serial.print("?");
        }
        delay(1000);                                        // 1 Sekunde warten
        t_TerminalRead();                                   // Reset-Anforderung prüfen
    }

    counter++;
}
```

IoT-Brick Set Beispiel 16.2 Listing

Das folgende Programm mit 169 Zeilen ist nicht Bestandteil des IoT-Handbuchs. Es stellt eine MQTT-Verbindung mit demAllnet-MQTT-Server her. Vom Allnet-MQTT-Server werden verschiedene Werte wie Temperatur in einigen deutschen Großstädten, DAX-Kurs, Gold-Kurs, Dollar-Kurs etc. geladen und auf dem Terminal formatiert aufbereitet ausgegeben.

```
Verbindungsstatus mit Server 'iot.allnet.de': Verbindung hergestellt

-------------------------------- Temperatur --------------------------------

Hamburg     = 22,3°C                    Berlin      = 18,0°C
Frankfurt   = 24,5°C                    Stuttgart   = 22,4°C
Hannover    = 24,5°C                    München     = 23,6°C
Köln        = 26,2°C                    Düsseldorf  = 25,5°C
Nürnberg    = 20,8°C                    Dresden     = 22,9°C

------------------------- Aktien- und Währungskurse -------------------------

DAX         = 12.356,98                 Differenz   = -24,27 (-0,20%)
MDAX        = 24.510,55                 Differenz   = -46,53 (-0,19%)
ESTX50      =  3.456,55                 Differenz   = -5,51 (-0,16%)
Gold        =  1.220,50                 Differenz   = -2,80 (-0,23%)
Öl          =     44,38                 Differenz   = -1,14 (-2,50%)
EUR/USD     =      1,14                 Differenz   = -0,00 (-0,01%)
BitCoin/USD =  2.572,10                 Differenz   = -4,90 (-0,19%)

--------------------------------- Sonstige ---------------------------------

```

```
//  Beispiel 16.2: MQTT-Allnet Daten vom Server holen und anzeigen
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <all_MQTT.h>


int counter = 0;


void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANautoConnect(true);                                  // Verbindung, ggf. gespeicherte, über WLAN herstellen
    IoT_NTPinit();                                              // NTP-Timeserver initialisieren

    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
    IoT_DisplayDrawText(64, 5, "Suche mqtt");
    IoT_DisplayDrawText(64, 34, "Server...");
    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren
    Serial.println("");

    // MQTT_Init(IPval("212.18.29.166"), 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
    // MQTT_Connect("general/data/#");

    MQTT_Init("iot.allnet.de", 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
    MQTT_Connect("general/data/#");

    Serial.print(t_TerminalClearScreen());
    MQTT_ConnectTrace = false;
}

void loop()
{
    IoT_Idle();

    if (IoT_Keypress())                                         // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        IoT_DisplayUpdate();                                      // OLED-Anzeige aktualisieren
        Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
        Serial.print("WiFi is ");
        Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
        Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
        IoT_WaitNoKeypress();
    }
    else
```

```
{
    IoT_Idle();
    if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
    {
        if (IoT_NTPeventAvail)                                          // Zeitevent triggern
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }

    IoT_DisplayClear(16);                                             // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                          // Zentrierte Darstellung des Textes
    if (IoT_WLANinited)
    {
        IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
        if (MQTT_Client.connected())
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Verbunden");
            IoT_DisplayDrawText(64, 34, "mit mqtt");
        }
        else
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Suche mqtt");
            IoT_DisplayDrawText(64, 34, "Server...");
        }
        IoT_DisplayUpdate();                                            // OLED-Anzeige aktualisieren
    }
    else
    {
        IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                        // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Kein");
        IoT_DisplayDrawText(64, 34, "WLAN");
    }
}
IoT_DisplayUpdate();                                                // OLED-Anzeige aktualisieren

bool success = MQTT_HandleClient();                                 // Verbindung aufrecht erhalten, ggf. neu aufbauen

if (MQTT_EventAvail)
{
    MQTT_GetNextEvent();
    if (not(MQTT_AllnetParseEventCurrent()))
    {
        Serial.print("Time: "      + cleanasc(String(MQTT_EventCurrentTime)));
        Serial.println(", Topic: " + cleanasc(String(MQTT_EventCurrentTopic)));
        Serial.println("Message: " + String(MQTT_EventCurrentMessage));
```

```
    }
}
else
{
    String eol = t_TerminalClearEndOfLine();
    t_TerminalRead();                                                   // Reset-Anforderung prüfen
    Serial.print(t_TerminalCursorHome());
    Serial.print("Verbindungsstatus mit Server '" + MQTT_ServerURL + "': ");
    IoT_DisplayClear(24);                                               // OLED Display löschen (24 Punkt Schrift)
    if (MQTT_Client.connected())
    {
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                          // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Verbunden");
        IoT_DisplayDrawText(64, 34, "mit mqtt");
        Serial.println("Verbindung hergestellt" + eol);
    }
    else
    {
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                          // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Suche mqtt");
        IoT_DisplayDrawText(64, 34, "Server...");
        Serial.println("Server wird gesucht" + eol);
    }
    IoT_DisplayUpdate();                                                // OLED-Anzeige aktualisieren
    Serial.println("");
    Serial.print  (fillcenter("- Temperaturen ", "-", 80));
    Serial.println(eol);
    Serial.print  ("Hamburg     = " + strrealform(MQTT_AllnetTemperaturHamburg,      32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Berlin      = " + strrealform(MQTT_AllnetTemperaturBerlin,       32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Frankfurt   = " + strrealform(MQTT_AllnetTemperaturFrankfurt,    32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Stuttgart   = " + strrealform(MQTT_AllnetTemperaturStuttgart,    32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Hannover    = " + strrealform(MQTT_AllnetTemperaturHannover,     32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("München     = " + strrealform(MQTT_AllnetTemperaturMuenchen,     32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Köln        = " + strrealform(MQTT_AllnetTemperaturKoeln,        32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Düsseldorf  = " + strrealform(MQTT_AllnetTemperaturDuesseldorf,  32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Nürnberg    = " + strrealform(MQTT_AllnetTemperaturNuernberg,    32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Dresden     = " + strrealform(MQTT_AllnetTemperaturDresden,      32, 2, 1, false, false) + "°C" + eol);
    Serial.println(eol);
    Serial.print  (fillcenter(" Aktien- und Währungskurse ", "-", 81)); // 81 weil das "ä" längentechnisch 2 Zeichen belegt
    Serial.println(eol);
    Serial.print  ("DAX         = " + strrealform(MQTT_AllnetFinanceDaxVal,          32, 6, 2, true, false)  + spc(16));
    Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceDaxDifVal,       32, 1, 2, false, false, true) + " (");
    Serial.println(                   strrealform(MQTT_AllnetFinanceDaxDifPerc,     32, 1, 2, false, false, true) + "%)" + eol);
    Serial.print  ("MDAX        = " + strrealform(MQTT_AllnetFinanceMDaxVal,         32, 6, 2, true, false)  + spc(16));
    Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceMDaxDifVal,      32, 1, 2, false, false, true)  + " (");
    Serial.println(                   strrealform(MQTT_AllnetFinanceMDaxDifPerc,    32, 1, 2, false, false, true) + "%)" + eol);
    Serial.print  ("ESTX50      = " + strrealform(MQTT_AllnetFinanceEStx50Val,       32, 6, 2, true, false)  + spc(16));
    Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceEStx50DifVal,    32, 1, 2, false, false, true)  + " (");
    Serial.println(                   strrealform(MQTT_AllnetFinanceEStx50DifPerc,  32, 1, 2, false, false, true) + "%)" + eol);
    Serial.print  ("Gold        = " + strrealform(MQTT_AllnetFinanceGoldVal,         32, 6, 2, true, false)  + spc(16));
```

```
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceGoldDifVal,      32, 1, 2, false, false, true)  + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceGoldDifPerc,     32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("Öl          = " + strrealform(MQTT_AllnetFinanceOelVal,          32, 6, 2, true, false)  + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceOelDifVal,       32, 1, 2, false, false, true)  + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceOelDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("EUR/USD     = " + strrealform(MQTT_AllnetFinanceEURVal,          32, 6, 2, true, false)  + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceEURDifVal,       32, 1, 2, false, false, true)  + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceEURDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("BitCoin/USD = " + strrealform(MQTT_AllnetFinanceBTCVal,          32, 6, 2, true, false)  + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceBTCDifVal,       32, 1, 2, false, false, true)  + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceBTCDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
        Serial.println(eol);
        Serial.print  (fillcenter(" Sonstige ", "-", 80));
        Serial.println(eol);
        delay(1000);                                                    // 1 Sekunde warten
    }

    counter++;
}
```

IoT-Brick Set Beispiel 16.3 Listing

Das folgende Programm mit 250 Zeilen ist nicht Bestandteil des IoT-Handbuchs. Es stellt eine MQTT-Verbindung mit demAllnet-MQTT-Server her. Vom Allnet-MQTT-Server werden verschiedene Werte wie Temperatur in einigen deutschen Großstädten, DAX-Kurs, Gold-Kurs, Dollar-Kurs etc. geladen und als Web-Seite formatiert aufbereitet ausgegeben.

Brick'R'knowledge Allnet MQTT-Explorer

```
Verbindungsstatus mit Server 'iot.allnet.de': Verbindung hergestellt

———————————————————— Temperaturen ————————————————————

Hamburg     = 22,3°C                Berlin      = 18,0°C
Frankfurt   = 24,5°C                Stuttgart   = 22,4°C
Hannover    = 24,5°C                München     = 23,6°C
Köln        = 26,2°C                Düsseldorf  = 25,5°C
Nürnberg    = 20,8°C                Dresden     = 22,9°C

———————————————————— Aktien- und Währungskurse ————————————————————

DAX         = 12.356,98             Differenz   = -24,27 (-0,20%)
MDAX        = 24.510,55             Differenz   = -46,53 (-0,19%)
ESTX50      =  3.456,55             Differenz   = -5,51 (-0,16%)
Gold        =  1.220,50             Differenz   = -2,80 (-0,23%)
Öl          =     44,38             Differenz   = -1,14 (-2,50%)
EUR/USD     =      1,14             Differenz   = -0,00 (-0,01%)
BitCoin/USD =  2.572,10             Differenz   = -4,90 (-0,19%)
```

```
//  Beispiel 16.3: MQTT-Allnet Daten vom Server holen und auf der Webseite anzeigen
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <all_MQTT.h>


int counter = 0;


void setup()
{
    IoT_Init(true);
    t_TerminalInit();
    IoT_WLANautoConnect(true);                                  // Verbindung, ggf. gespeicherte, über WLAN herstellen
    IoT_NTPinit();                                              // NTP-Timeserver initialisieren

    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
    IoT_DisplayDrawText(64, 5, "Suche mqtt");
    IoT_DisplayDrawText(64, 34, "Server...");
    IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren
    Serial.println("");

    // MQTT_Init(IPval("212.18.29.166"), 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
    // MQTT_Connect("general/data/#");

    MQTT_Init("iot.allnet.de", 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
    MQTT_Connect("general/data/#");

    MQTT_ConnectTrace = false;

    IoT_WebServer.on("/", handleRoot);                          // führe die Funktion handleRoot (siehe unten) aus.
    IoT_WebServer.begin();                                      // ab jetzt "hört" der Server auf HTTP-Anfragen
    Serial.print(t_TerminalClearScreen());
}

void loop()
{
    IoT_Idle();

    IoT_WebServer.handleClient();                               // Bediene die http Anfragen

    if (IoT_Keypress())                                         // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayWLANstatus();
        IoT_DisplayUpdate();                                      // OLED-Anzeige aktualisieren
        Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
        Serial.print("WiFi is ");
```

```
    Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
    Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
    IoT_WaitNoKeypress();
}
else
{
    IoT_Idle();
    if (counter % 25 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 2 Sekunden)
    {
        if (IoT_NTPeventAvail)                                          // Zeitevent triggern
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }

    IoT_DisplayClear(16);                                             // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                          // Zentrierte Darstellung des Textes
    if (IoT_WLANinited)
    {
        IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
        if (MQTT_Client.connected())
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                      // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Verbunden");
            IoT_DisplayDrawText(64, 34, "mit mqtt");
        }
        else
        {
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                      // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Suche mqtt");
            IoT_DisplayDrawText(64, 34, "Server...");
        }
        IoT_DisplayUpdate();                                            // OLED-Anzeige aktualisieren
    }
    else
    {
        IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                        // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Kein");
        IoT_DisplayDrawText(64, 34, "WLAN");
    }
}
IoT_DisplayUpdate();                                                // OLED-Anzeige aktualisieren

bool success = MQTT_HandleClient();                                 // Verbindung aufrecht erhalten, ggf. neu aufbauen

if (MQTT_EventAvail)
{
    MQTT_GetNextEvent();
```

```
    if (not(MQTT_AllnetParseEventCurrent()))
    {
        Serial.print("Time: "      + cleanasc(String(MQTT_EventCurrentTime)));
        Serial.println(", Topic: " + cleanasc(String(MQTT_EventCurrentTopic)));
        Serial.println("Message: " + String(MQTT_EventCurrentMessage));
        delay(10000);                                                  // 10 Sekunden warten
    }
}
else
{
    String eol = t_TerminalClearEndOfLine();
    IoT_WebServer.handleClient();                                   // Bediene die http Anfragen
    t_TerminalRead();                                               // Reset-Anforderung prüfen
    Serial.print(t_TerminalCursorHome());
    Serial.print("Verbindungsstatus mit Server '" + MQTT_ServerURL + "': ");
    IoT_DisplayClear(24);                                                 // OLED Display löschen (24 Punkt Schrift)
    if (MQTT_Client.connected())
    {
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                            // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Verbunden");
        IoT_DisplayDrawText(64, 34, "mit mqtt");
        Serial.println("Verbindung hergestellt" + eol);
    }
    else
    {
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                            // Zentrierte Darstellung des Textes
        IoT_DisplayDrawText(64, 5, "Suche mqtt");
        IoT_DisplayDrawText(64, 34, "Server...");
        Serial.println("Server wird gesucht" + eol);
    }
    IoT_DisplayUpdate();                                                // OLED-Anzeige aktualisieren
    Serial.println(eol);
    Serial.print  (fillcenter("- Temperatur ", "-", 80));
    Serial.println(eol);
    Serial.print  ("Hamburg     = " + strrealform(MQTT_AllnetTemperaturHamburg,      32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Berlin      = " + strrealform(MQTT_AllnetTemperaturBerlin,       32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Frankfurt   = " + strrealform(MQTT_AllnetTemperaturFrankfurt,    32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Stuttgart   = " + strrealform(MQTT_AllnetTemperaturStuttgart,    32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Hannover    = " + strrealform(MQTT_AllnetTemperaturHannover,     32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("München     = " + strrealform(MQTT_AllnetTemperaturMuenchen,     32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Köln        = " + strrealform(MQTT_AllnetTemperaturKoeln,        32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Düsseldorf  = " + strrealform(MQTT_AllnetTemperaturDuesseldorf,  32, 2, 1, false, false) + "°C" + eol);
    Serial.print  ("Nürnberg    = " + strrealform(MQTT_AllnetTemperaturNuernberg,    32, 2, 1, false, false) + "°C" + spc(19));
    Serial.println("Dresden     = " + strrealform(MQTT_AllnetTemperaturDresden,      32, 2, 1, false, false) + "°C" + eol);
    Serial.println(eol);
    Serial.print  (fillcenter(" Aktien- und Währungskurse ", "-", 81)); // 81 weil das "ä" längentechnisch 2 Zeichen belegt
    Serial.println(eol);
    Serial.print  ("DAX         = " + strrealform(MQTT_AllnetFinanceDaxVal,          32, 6, 2, true, false) + spc(16));
    Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceDaxDifVal,       32, 1, 2, false, false, true) + " (");
    Serial.println(                   strrealform(MQTT_AllnetFinanceDaxDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
```

```
        Serial.print  ("MDAX        = " + strrealform(MQTT_AllnetFinanceMDaxVal,         32, 6, 2, true, false) + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceMDaxDifVal,      32, 1, 2, false, false, true) + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceMDaxDifPerc,     32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("ESTX50      = " + strrealform(MQTT_AllnetFinanceEStx50Val,       32, 6, 2, true, false) + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceEStx50DifVal,    32, 1, 2, false, false, true) + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceEStx50DifPerc,   32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("Gold        = " + strrealform(MQTT_AllnetFinanceGoldVal,         32, 6, 2, true, false) + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceGoldDifVal,      32, 1, 2, false, false, true) + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceGoldDifPerc,     32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("Öl          = " + strrealform(MQTT_AllnetFinanceOelVal,          32, 6, 2, true, false) + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceOelDifVal,       32, 1, 2, false, false, true) + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceOelDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("EUR/USD     = " + strrealform(MQTT_AllnetFinanceEURVal,          32, 6, 2, true, false)  + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceEURDifVal,       32, 1, 2, false, false, true)  + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceEURDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
        Serial.print  ("BitCoin/USD = " + strrealform(MQTT_AllnetFinanceBTCVal,          32, 6, 2, true, false)  + spc(16));
        Serial.print  ("Differenz   = " + strrealform(MQTT_AllnetFinanceBTCDifVal,       32, 1, 2, false, false, true)  + " (");
        Serial.println(                   strrealform(MQTT_AllnetFinanceBTCDifPerc,      32, 1, 2, false, false, true) + "%)" + eol);
        Serial.println(eol);
        Serial.print  (fillcenter(" Sonstige ", "-", 80));
        Serial.println(eol);
        IoT_WebServer.handleClient();                                   // Bediene die http Anfragen
        delay(1000);                                                    // 1 Sekunde warten
    }

    counter++;
}

void handleRoot()
{
    String title_web = "Brick'R'knowledge Allnet MQTT-Explorer";
    String stat = "";
    if (MQTT_Client.connected())
    {
        stat = "Verbindung hergestellt";
    }
    else
    {
        stat = "Server wird gesucht";
    }

    String content =                                                  // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
    "<html>\
    <head>\
    <title>" + cleanhtml(title_web) + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: #000000; }\
    </style>\
    <style type='text/css'>\
```

```
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p> </p>\
    <p>" + cleanhtml("Verbindungsstatus mit Server '" + MQTT_ServerURL + "': " + stat)    + "\
    <p> </p>\
    <p>" + cleanhtml(fillcenter(" Temperaturen ", "-", 80)) + "</p>\
    <p> </p>\
    <p>" + cleanhtml("Hamburg     = " + strrealform(MQTT_AllnetTemperaturHamburg,      32, 2, 1, false, false) + "°C" + spc(19))    + \
    cleanhtml("Berlin      = " + strrealform(MQTT_AllnetTemperaturBerlin,       32, 2, 1, false, false) + "°C  ")            + "</p>\
    <p>" + cleanhtml("Frankfurt   = " + strrealform(MQTT_AllnetTemperaturFrankfurt,     32, 2, 1, false, false) + "°C" + spc(19))    + \
    cleanhtml("Stuttgart   = " + strrealform(MQTT_AllnetTemperaturStuttgart,    32, 2, 1, false, false) + "°C  ")            + "</p>\
    <p>" + cleanhtml("Hannover    = " + strrealform(MQTT_AllnetTemperaturHannover,     32, 2, 1, false, false) + "°C" + spc(19))    + \
    cleanhtml("München     = " + strrealform(MQTT_AllnetTemperaturMuenchen,     32, 2, 1, false, false) + "°C  ")            + "</p>\
    <p>" + cleanhtml("Köln         = " + strrealform(MQTT_AllnetTemperaturKoeln,          32, 2, 1, false, false) + "°C" + spc(19))    + \
    cleanhtml("Düsseldorf  = " + strrealform(MQTT_AllnetTemperaturDuesseldorf,  32, 2, 1, false, false) + "°C  ")            + "</p>\
    <p>" + cleanhtml("Nürnberg    = " + strrealform(MQTT_AllnetTemperaturNuernberg,    32, 2, 1, false, false) + "°C" + spc(19))    + \
    cleanhtml("Dresden     = " + strrealform(MQTT_AllnetTemperaturDresden,      32, 2, 1, false, false) + "°C  ")            + "</p>\
    <p> </p>\
    <p>" + cleanhtml(fillcenter(" Aktien- und Währungskurse ", "-", 81)) + "</p>\
    <p> </p>\
    <p>" + cleanhtml("DAX          = " + strrealform(MQTT_AllnetFinanceDaxVal,          32, 6, 2, true, false) + spc(16))     + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceDaxDifVal,       32, 1, 2, false, false, true) + " (") + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceDaxDifPerc,      32, 1, 2, false, false, true) + "%)") + "</p>\
    <p>" + cleanhtml("MDAX         = " + strrealform(MQTT_AllnetFinanceMDaxVal,         32, 6, 2, true, false)  + spc(16))    + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceMDaxDifVal,      32, 1, 2, false, false, true) + " (") + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceMDaxDifPerc,     32, 1, 2, false, false, true) + "%)") + "</p>\
    <p>" + cleanhtml("ESTX50       = " + strrealform(MQTT_AllnetFinanceEStx50Val,       32, 6, 2, true, false) + spc(16))     + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceEStx50DifVal,    32, 1, 2, false, false, true) + " (")  + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceEStx50DifPerc,   32, 1, 2, false, false, true) + "%)")  + "</p>\
    <p>" + cleanhtml("Gold         = " + strrealform(MQTT_AllnetFinanceGoldVal,         32, 6, 2, true, false) + spc(16))     + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceGoldDifVal,      32, 1, 2, false, false, true) + " (")  + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceGoldDifPerc,     32, 1, 2, false, false, true) + "%)")  + "</p>\
    <p>" + cleanhtml("Öl           = " + strrealform(MQTT_AllnetFinanceOelVal,          32, 6, 2, true, false) + spc(16))     + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceOelDifVal,       32, 1, 2, false, false, true) + " (")  + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceOelDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>\
    <p>" + cleanhtml("EUR/USD      = " + strrealform(MQTT_AllnetFinanceEURVal,          32, 6, 2, true, false) + spc(16))     + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceEURDifVal,       32, 1, 2, false, false, true) + " (")  + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceEURDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>\
    <p>" + cleanhtml("BitCoin/USD = " + strrealform(MQTT_AllnetFinanceBTCVal,         32, 6, 2, true, false) + spc(16))     + \
    cleanhtml("Differenz   = " + strrealform(MQTT_AllnetFinanceBTCDifVal,       32, 1, 2, false, false, true) + " (")  + \
    cleanhtml(                   strrealform(MQTT_AllnetFinanceBTCDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>\
    <p> </p>\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                        // HTTP-Seite aktualisieren
}
```

IoT-Brick Set Beispiel 17 Listing

Das folgende Programm mit 46 Zeilen ist nicht Bestandteil des IoT-Handbuchs. Es stellt eine HTTP-Verbindung mit einem Web-Server her und gibt den HTTP-Quellcode auf dem Terminal aus. Dabei ist zu beachten, dass die verwendete Web-Seite nicht sehr groß sein darf, da sonst der Speicher des ESP8266 nicht ausreicht, um alle Daten von der Web-Seite zu laden.

```
*WM:
*WM: AutoConnect
*WM: Connecting as wifi client...
*WM: Already connected. Bailing out.
*WM: IP Address:
*WM: 192.168.1.23
Connected to AirPort Extreme, IP: 192.168.1.23
HTTP GET Result: 400, Length = 311
---------------------- http://www.brickrknowledge.de ----------------------
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.2.22 (Debian) Server at brickrknowledge.com Port 80</address>
</body></html>
------------------------------------------------------------------------------
```

```
//  Beispiel 17: HTTP Daten von Internetseite holen und anzeigen
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <ESP8266HTTPClient.h>


void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANautoConnect(true);                                      // Verbindung, ggf. gespeicherte, über WLAN herstellen
}

void loop()
{
    String page = "http://www.brickrknowledge.de";
    HTTPClient http;
    http.begin(page); //HTTP
    int httpCode = http.GET();  // start connection and send HTTP header
    Serial.print("HTTP GET Result: " + str(httpCode) + ", Length = ");
    if (httpCode > 0)  // httpCode will be negative on error
    {
        String payload = http.getString();
        int L = len(payload);
        Serial.println(str(L));
        Serial.println(fillcenter(" " + page + " ", "-", 80) + chr(8));
        for (int i = 0; i < L; i++)
        {
            int c = payload[i];
            if (c == 10)
            {
                c = 13;
            }
            Serial.print(chr(c));
        }
        Serial.println(repeatstr("-", 80) + chr(8));
    }
    else
    {
        Serial.println("0, Error Message: " + http.errorToString(httpCode));
    }
    http.end();
    IoT_ShutDown();
}
}
```

IoT-Brick Set Beispiel 18 Listing

Das folgende Programm mit 108 Zeilen ist nicht Bestandteil des IoT-Handbuchs. Es stellt eine HTTP-Verbindung mit einem ALL-3500 Home-Automations-Server her und gibt Namen, Typ und Werte aller Sensoren auf dem Terminal aus. Der String `page` in der `loop()` Funktion muss vor der ersten Benutzung auf die IP-Adresse des geünschten ALL-3500 sowie der für die Fernsteuerung festgelegte Benutzernamen und das zugehörige Password geändert werden.

```
*WM:
*WM: AutoConnect
*WM: Connecting as wifi client...
*WM: Already connected. Bailing out.
*WM: IP Address:
*WM: 192.168.1.23
Connected to AirPort Extreme, IP: 192.168.1.23
HTTP GET Result: 200, Length = 7.439

----- http://allnetuser:allnetpassword@192.168.1.64/xml/json.php?mode=all ------
01 : Interner Sensor           = Temperatursensor          = 37.56°C
02 : Sauna                     = Temperatursensor          = 24.25°C
03 : Netzwerkschrank           = Temperatursensor          = 27.25°C
04 : Dachgeschoß               = Temperatursensor          = 27.62°C
05 : Werkstatt                 = Temperatursensor          = 20.50°C
06 : Werkstatt                 = Feuchtigkeitssensor       = 68.52%
07 : Labor                     = Temperatursensor          = 25.34°C
08 : Labor                     = Feuchtigkeitssensor       = 52.54%
09 : Waschkeller               = Temperatursensor          = 22.96°C
10 : Waschkeller               = Feuchtigkeitssensor       = 50.48%
-------------------------------------------------------------------------
```

```
//  Beispiel 18: HTTP Daten von einem ALL3500 holen und anzeigen
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>
#include <ESP8266HTTPClient.h>

// Beispiel für den ersten Sensor, der übergeben wird:
//
// L...!....1....!....2....!....3....!....4....!....5....!....6....!....7....!....8....!....9....!....0
// [{"id":"1","name":"Interner Sensor","description":"Temperatursensor","fe_view":"1","sort":"2:2",
// "fe_cvs_show_typ":"1","actor_analogValue":null,"digitalToText":"0::","tileColors":"1e7eac:900000:900000",
// "tileFormats":"55:","lang_port_identifier":null,"fading":null,"device_type":"4","value":"36.00","error":0,
// "config":{"icon":"","display":{"min":"0","max":"60"},"limit":{"min":"5","max":"50"}},"info":{"activ":"1",
// "enabled":"1","unit":"\u00b0C","type":"1","view":"1","chipid":"2","chipnumber":"2","chipaddress":"3",
// "helperchipnumber":"0","helperchipaddress":"0","bitaddress":"0"},"minmax":{"today":{"min":"35.50","max":"37.12"},
// "absolute":{"min":"19.62","max":"255.93"}},"connection":{"port":"4","bus":"67","group":"0"}},


void setup()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANautoConnect(true);                                      // Verbindung, ggf. gespeicherte, über WLAN herstellen
}

void loop()
{
    String page = "http://allnetuser:allnetpassword@192.168.1.64/xml/json.php?mode=all";    // Webseite des ALL3500 hier eintragen incl. Zugangsdaten
    HTTPClient http;
    http.begin(page); //HTTP
    int httpCode = http.GET();  // start connection and send HTTP header
    Serial.print("HTTP GET Result: " + str(httpCode) + ", Length = ");
    if (httpCode > 0)  // httpCode will be negative on error
    {   // HTTP header has been send and Server response header has been handled
        if (httpCode > 0)
        {
            String result = http.getString();
            int L = len(result);
            int p = 1;
            int n = 1;
            Serial.println(str(L));
            Serial.println("");
            Serial.println(fillcenter(" " + page + " ", "-", 80) + chr(8));
            int count = 0;
            while ((p > 0) && (n > 0))
            {
                p = position(result, quotate("name"), p);
                if (p > 0)
                {
```

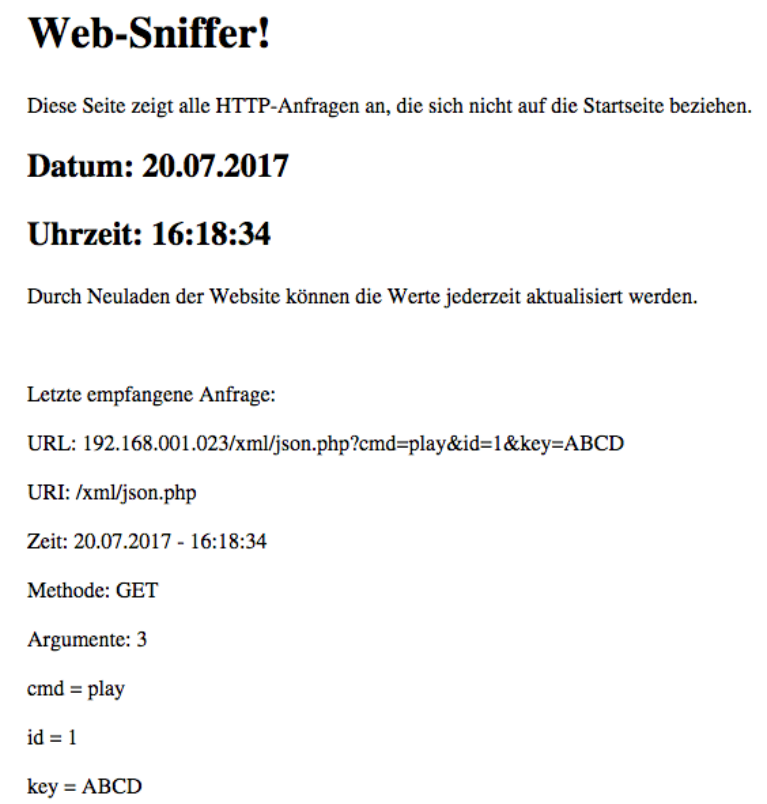
```
            n = position(result, chr(34), p + 8);
            if (n > 0)
            {
                int L = n - p - 8;
                String SensorName = convertescape(mid(result, p + 8, L));
                p = position(result, quotate("description"), n);
                if (p > 0)
                {
                    n = position(result, chr(34), p + 15);
                    if (n > 0)
                    {
                        int L = n - p - 15;
                        String SensorDescr = convertescape(mid(result, p + 15, L));
                        p = position(result, quotate("value"), n);
                        if (p > 0)
                        {
                            n = position(result, chr(34), p + 9);
                            if (n > 0)
                            {
                                int L = n - p - 9;
                                String SensorValue = convertescape(mid(result, p + 9, L));
                                p = position(result, quotate("unit"), n);
                                if (p > 0)
                                {
                                    n = position(result, chr(34), p + 8);
                                    if (n > 0)
                                    {
                                        int L = n - p - 8;
                                        String SensorUnit = convertescape(mid(result, p + 8, L));
                                        Serial.print(strform(count + 1, 48, 2, false) + " : ");
                                        Serial.print(left(SensorName + spc(25), 25));
                                        Serial.print(" = ");
                                        Serial.print(left(SensorDescr + spc(25), 25));
                                        Serial.print(" = ");
                                        Serial.print(left(SensorValue, 20));
                                        Serial.print(SensorUnit);
                                        Serial.println("");
                                        count += 1;
                                        p = position(result, "," + quotate("group") + ":", p);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    Serial.println(repeatstr("-", 80) + chr(8));
}
```

```
    }
    else
    {
        Serial.println("0, Error Message: " + http.errorToString(httpCode));
    }
    http.end();
    IoT_ShutDown();
}
```

IoT-Brick Set Beispiel 19 Listing

Das folgende Programm mit 168 Zeilen ist nicht Bestandteil des IoT-Handbuchs und implementiert einen einfachen Web-Sniffer. Man kann mit einem beliebigen HTTP-fähigen Gerät oder einem beliebigen Browser eine Internet-Verbindung zum IoT-Brick herstellen und der IoT-Brick protokolliert alle ankommenden Anfragen, egal ob HTTP, JSON oder ein anderes Format. Dadurch lässt sich anschaulich zeigen, wann und welche Aufrufe an den IoT-Brick gesendet werden. Im Beispiel unten wird eine Anfrage empfangen, bei der im Browser die URL „`192.168.1.23/xml/json.php?cmd=play&id=1&key=ABCD`“ eingegeben wurde. Diese Eingabe wird als URL angezeigt, darunter die URI, d.h. der Pfad (`/xml/json.php`), der übermittelt wurde sowie die drei Parameter (Argumente), die nach dem Fragezeichen „`?`“ stehen und mit einem „`&`” voneinander getrennt sind. Ein Parameter besteht immer aus einem Namen, gefolgt von einem Gleichheitszeichen („=“) und einem alphanumerischen Wert. Fehlt das Gleichheitszeichen, so wird das Argument ignoriert.

```
*WM:
*WM: AutoConnect
*WM: Connecting as wifi client...
*WM: Using last saved values, should be faster
*WM: Connection result:
*WM: 3
*WM: IP Address:
*WM: 192.168.1.23
Connected to AirPort Extreme, IP: 192.168.1.23
HTTP server started

URL: 192.168.001.023/xml/json.php?cmd=play&id=1&key=ABCD
URI: /xml/json.php
Zeit: 01.01.1970 - 00:00:07
Methode: GET
Argumente: 3
cmd = play
id = 1
key = ABCD
```



Web-Sniffer!

Diese Seite zeigt alle HTTP-Anfragen an, die sich nicht auf die Startseite beziehen.

Datum: 20.07.2017

Uhrzeit: 16:18:34

Durch Neuladen der Website können die Werte jederzeit aktualisiert werden.

Letzte empfangene Anfrage:

URL: 192.168.001.023/xml/json.php?cmd=play&id=1&key=ABCD

URI: /xml/json.php

Zeit: 20.07.2017 - 16:18:34

Methode: GET

Argumente: 3

cmd = play

id = 1

key = ABCD

```
//  Beispiel 19 "Web-Sniffer"
//
//  Unter Verwendung der IoT Brick Library

#include <all_IoT.h>

int counter = 0;

String message = "";

void handleNotFound ()
{
    String argument = "";
    String msg = "";
    msg += "URI: ";
    msg += IoT_WebServer.uri();
    msg += "\nZeit: ";
    msg += IoT_NTPdate();
    msg += " - ";
    msg += IoT_NTPtime();
    msg += "\nMethode: ";
    msg += (IoT_WebServer.method() == HTTP_GET)?"GET":"POST";
    msg += "\nArgumente: ";
    msg += IoT_WebServer.args();
    msg += "\n";
    for (uint8_t i = 0; i < IoT_WebServer.args(); i++)
    {
        msg += IoT_WebServer.argName(i);
        msg += " = ";
        msg += IoT_WebServer.arg(i);
        msg += "\n";
        if (argument != "")
        {
            argument += "&";
        }
        argument += IoT_WebServer.argName(i);
        argument += "=";
        argument += IoT_WebServer.arg(i);
    }
    message = "URL: ";
    message += IoT_WLANaddress(true);
    message += IoT_WebServer.uri();
    if (argument != "")
    {
        message += "?";
        message += argument;
    }
    message += "\n";
    message += msg;
    handleRoot();
```

```
    Serial.println(replace(message, "\n", "\r"));                   // LineFeed in Return umwandeln (Terminal)
}

void setup ()
{
    IoT_Init(true);
    IoT_TerminalWaitInit(true);
    IoT_WLANautoConnect(true);                                      // Verbindung, ggf. gespeicherte, über WLAN herstellen
    IoT_NTPinit();                                                  // Sobald der Browser direkt auf das Stammverzeichnis zugreift,
    IoT_WebServer.onNotFound(handleNotFound);
    IoT_WebServer.on("/", handleRoot);                              // führe die Funktion handleRoot (siehe unten) aus.
    IoT_WebServer.begin();                                          // ab jetzt "hört" der Server auf HTTP-Anfragen
    Serial.println("HTTP server started");
    Serial.println("");
}

void loop ()
{
    IoT_Idle();

    IoT_WebServer.handleClient();                                   // Bediene die http Anfragen

    if (IoT_Keypress())                                             // Wenn Taster gedrückt wird, Konfiguration zeigen
    {
        IoT_DisplayClear(10);                                         // OLED Display löschen (10 Punkt Schrift)
        String state = IoT_WLANstatus();                              // Verbindungsstatus
        if (IoT_WLANinited)                                           // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());       // WLAN Netzwerk Name
            if (state == "Connected")                                   // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
            {
                state = state + " (" + str(IoT_WLANrssi()) + "dBm)";      // Signalstärke
            }
            IoT_DisplayDrawText(5, 10, state);                          // Verbindungsstatus & Signalstärke
            IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true));   // IP Adresse
            IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true));   // IP Gateway
            IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));    // IP Subnetzmaske
        }
        else                                                          // Falls WLAN Verbindung gescheitert ist, Status im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 10, state);
        }
    }
    else
    {
        IoT_Idle();
        if (counter % 100 == 0) //Infos seriell nicht zu oft ausgeben (ca. alle 10 Sekunden)
        {
            if (IoT_NTPeventAvail)                                      // Zeitevent triggern
            {
```

```
                // IoT_NTPprintEvent(IoT_NTPcurrentEvent);
                IoT_NTPeventAvail = false;
            }
        }

        IoT_DisplayClear(16);                                       // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                    // Zentrierte Darstellung des Textes
        if (IoT_NTPvalid())                                         // Prüfen, ob eine gültige Uhrzeit aus dem Internet empfangen wurde
        {
            IoT_DisplayDrawText(63, 0, IoT_NTPtime());
            IoT_DisplayDrawText(63, 15, IoT_NTPdate());
            IoT_DisplayDrawText(63, 45, "WLAN connected");
            IoT_DisplayAlignText(TEXT_ALIGN_LEFT);                    // Linksbündige Darstellung des Textes
            IoT_Idle();
        }
        else                                                        // "Bitte warten..." zentriert anzeigen
        {
            IoT_DisplayClear(24);                                   // OLED Display löschen (24 Punkt Schrift)
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                // Zentrierte Darstellung des Textes
            IoT_DisplayDrawText(64, 5, "Bitte");
            IoT_DisplayDrawText(64, 34, "warten...");
        }
    }
    IoT_DisplayUpdate();                                          // OLED-Anzeige aktualisieren
    delay(100);                                                   // 100 ms. warten
    counter++;
}

//Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot()
{
    bool AutoRefresh = false;
    String content;
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    content = "<!DOCTYPE html>";
    content += "<html>";
    content += "<head>";
    content += "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">";
    if (AutoRefresh)
    {
        content += "<meta http-equiv=\"refresh\" content=\"5; URL=http://" + IoT_WLANaddress(false) + "\">"; // Auto-Refresh
    }
    content += "<title>IoT Brick-Website</title>";
    content += "</head>";
    content += "<body>";
    content += "<h1> Web-Sniffer! </h1>";
    content += "<p>Diese Seite zeigt alle HTTP-Anfragen an, die sich nicht auf die Startseite beziehen.</p>";
    content += "<h2>Datum: "+date_web+"</h2>";
```

```
    content += "<h2>Uhrzeit: "+time_web+"</h2>";
    if (AutoRefresh)
    {
        content += "<p>Die Seite wird alle 5 Sekunden aktualisiert.</p>"; // Auto-Refresh
    }
    else
    {
        content += "<p>Durch Neuladen der Website können die Werte jederzeit aktualisiert werden.</p>";
    }
    content += "<p> </p>";
    content += "<p>Letzte empfangene Anfrage:</p>";
    content += "<p>";
    content += replace(message, "\n", "</p><p>");
    content += "</p>";
    content += "</body>";
    content += "</html>";
    IoT_WebServer.send(200, "text/html", content);   // HTTP-Statuscode 200 = OK (Anfrage wurde erfolgreich bearbeitet)
}
```

BRK-Clock (Deutsche und Englische Version) Listing

Das folgende Programm mit 656 Zeilen ist nicht Bestandteil des IoT-Handbuchs und beschaltet die BRK-Clock Matrix mit einem IoT-Brick. Dabei werden über das Webinterface des IoT-Bricks zahlreiche Funktionen für die Uhr (Farbanpassung mit vordefinierten Grundfarben sowie frei eingebbarer RGBW-Farbe, Wecker, Sekunden- und Minutentöne usw.) sowie einige Demos für die 8 x 8 RGBW-Matrix (Temperaturanzeige, Farbdemos, Kaleidoskop, Laufschrift usw.) zur Verfügung gestellt. Weiterhin kann man den IoT-Brick im Webinterface neu starten und die ganze Schaltung incl. der 8 x 8 RGBW-Matrix ausschalten. Darüber hinaus werden im Webinterface noch einige Informationen angezeigt wie Datum, Uhrzeit, Temperatur, Feuchtigkeit, die IP-Adresse, unter der der IoT-Brick im lokalen Netzwerk verfügbar ist sowie die öffentliche IP-Adresse, unter der der genutzte Internetanschluss im Internet angemeldet ist.

Das Webinterface ist farblich so gestaltet, dass es die aktuell angezeigte Wortfarbe der BRK-Clock annimmt und so optimiert, dass die Ladezeit unter eine Sekunde bleibt und auch mehrfache, parallele Browserzugriffe möglich sind, ohne dass der IoT-Brick sich dadurch aufhängt oder neu startet. Auf dem OLED-Display werden zusätzlich alle Parameter wie sekundengenaue Zeit, Datum, Temperatur und Luftfeuchtigkeit angezeigt. Weiterhin werden auf dem Display Hinweise für den Benutzer angezeigt, z.B. eine Aufforderung zum Drücken des Tasters, um eine einmal gestartete Demo wieder zu beenden.

Wahlweise können mit einer kleinen Verstärker-Schaltung Töne ausgegeben werden. Das sind zum Einen der Sekundenklick und der Minutenbeep und zum Anderen ein Sekundenbeep während des 60-sekündigen Alarms. Für die Soundausgabe wird der GPIO 13 benutzt. Allerdings wird das ganze Programm durch die standardmäßige Arduino-Soundausgabe instabil, vor allem dann, wenn der Sekundenton jede Sekunde abgespielt wird. Dabei kommt es dann nach einigen Minuten zu einem ungewollten Neustart des IoT-Bricks. Aus diesem Grund sind alle Töne standardmäßig ausgeschaltet. Das bedeutet, dass die Uhr beim Verzicht auf die Sekunden- und Minutenton Soundausgabe für einen Dauerbetrieb geeignet ist. Eine alternative Schaltung benutzt den GPIO 3 für die Soundausgabe. Dabei ist allerdings keine Terminal-Eingabe mehr möglich und der IoT-Brick lässt sich nur dann flashen, wenn die Verbindung zum GPIO 3 mit einem Schalter unterbrochen wurde.

Die Leitung GPIO 14 wird für die Ansteuerung der 8 x 8 RGBW-Matrix benötigt. Der GPIO 12 wird für die Ansteuerung des kombinierten Temperatur- und Luftfeuchtigkeits-Sensor verwendet. Sowohl der IoT-Brick als auch der Sensorbaustein und die Verstärkerschaltung können wahlweise mit 9 bis 12 Volt betrieben werden, ohne dass hierfür die Schaltung angepasst werden müsste.

Brick'R'knowledge BRK-Clock Internetseite

Informationen

Datum: 16.06.2017

Uhrzeit: 13:33:52

Weckzeit: (ausgeschaltet)

Temperatur: 23°C

Feuchtigkeit: 44%

IP-Addresse: 192.168.1.16

Öffentliche IP: 78.94.15.26

Einstellungen

- [ ] Wecker eingeschaltet H: 0 M: 00
- [ ] Weckton eingeschaltet

- [ ] Sekundenton eingeschaltet
- [ ] Minutenton eingeschaltet

- [x] OLED-Display eingeschaltet

- ( ) Weiße Schrift
- (•) Rote Schrift
- ( ) Grüne Schrift
- ( ) Blaue Schrift
- ( ) Türkise Schrift
- ( ) Violette Schrift
- ( ) Gelbe Schrift
- ( ) Orange Schrift
- ( ) RGBW Schrift R: 255 G: 0 B: 0 W: 0

[Einstellungen der BRK-Clock anwenden]

Demos

[BRK-Clock Selbsttest]

[BRK-Clock Temperatur]

[BRK-Clock Rechtecke]

[BRK-Clock Kaleidoskop mit 8 Farben]

[BRK-Clock Kaleidoskop mit 16 Farben]

[BRK-Clock Schlangenlinien]

[BRK-Clock Weiße Welle über Regenbogen]

[BRK-Clock Weißes Pulsieren]

[BRK-Clock Regenbogen wird weiß]

[BRK-Clock Ambientebeleuchtung]

[BRK-Clock Regenbogen]

[BRK-Clock Regenbogenzyklus]

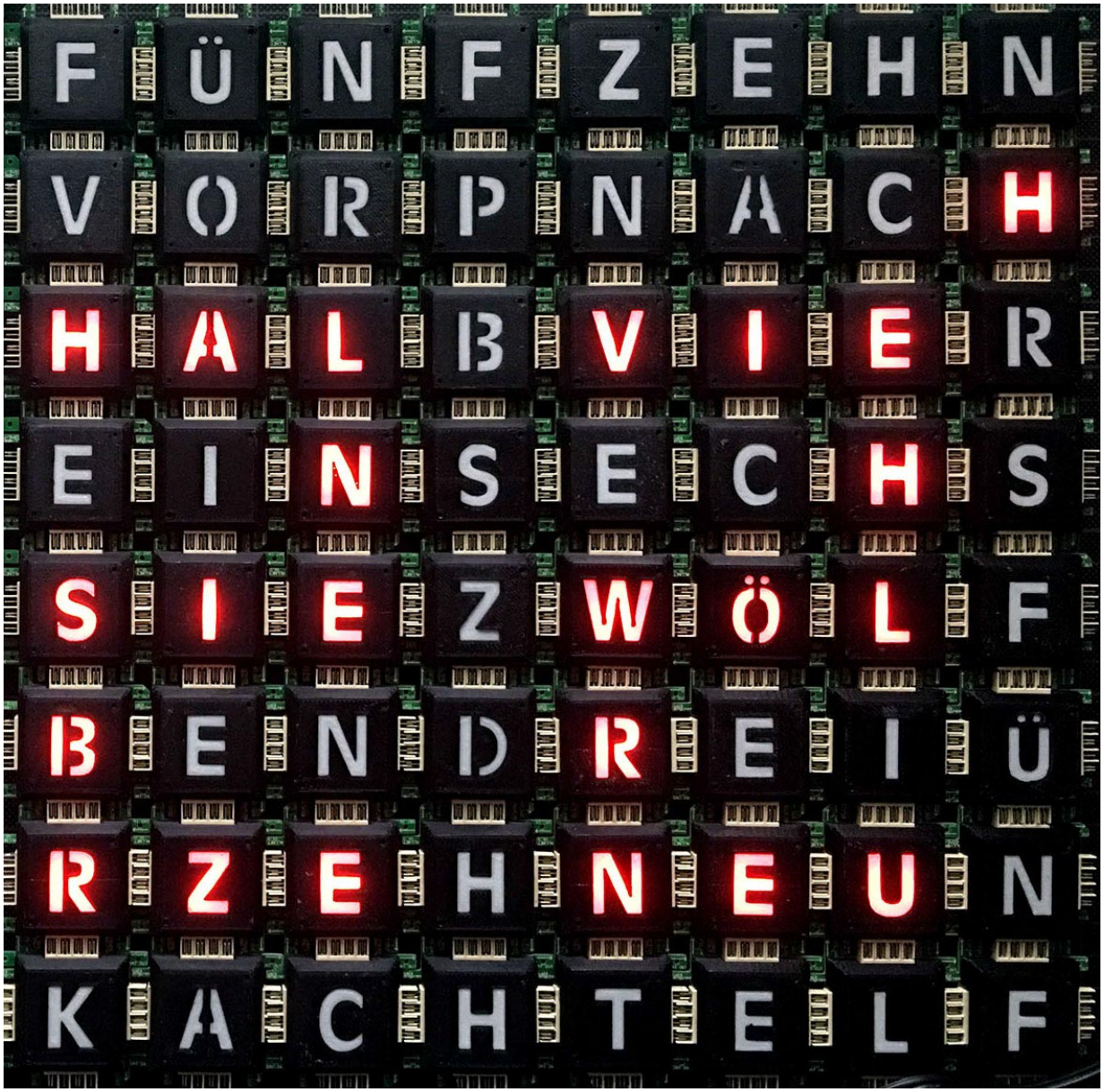
[BRK-Clock Laufschrift] [Brick'R'knowledge]

System

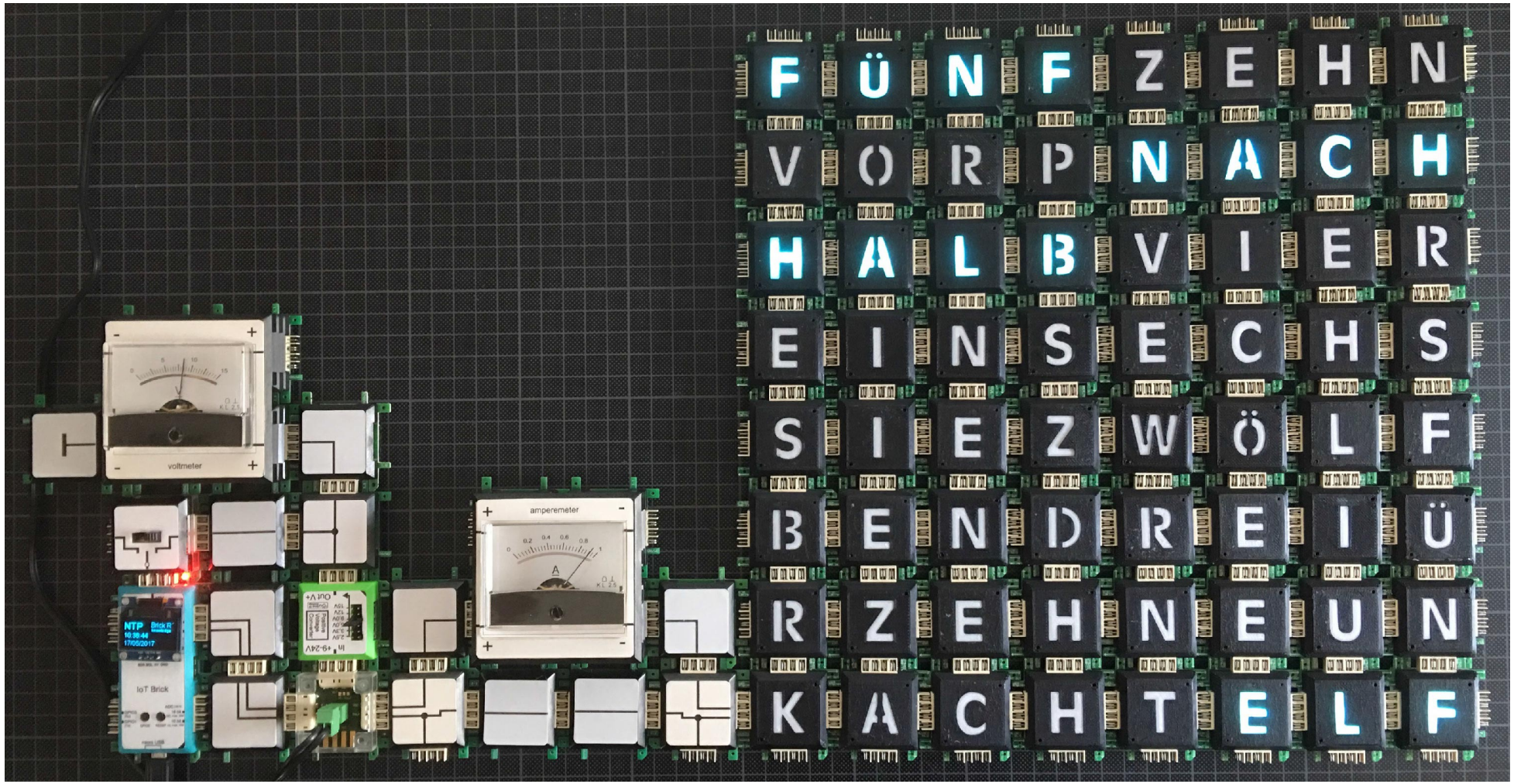
[BRK-Clock neu starten]

[BRK-Clock ausschalten]

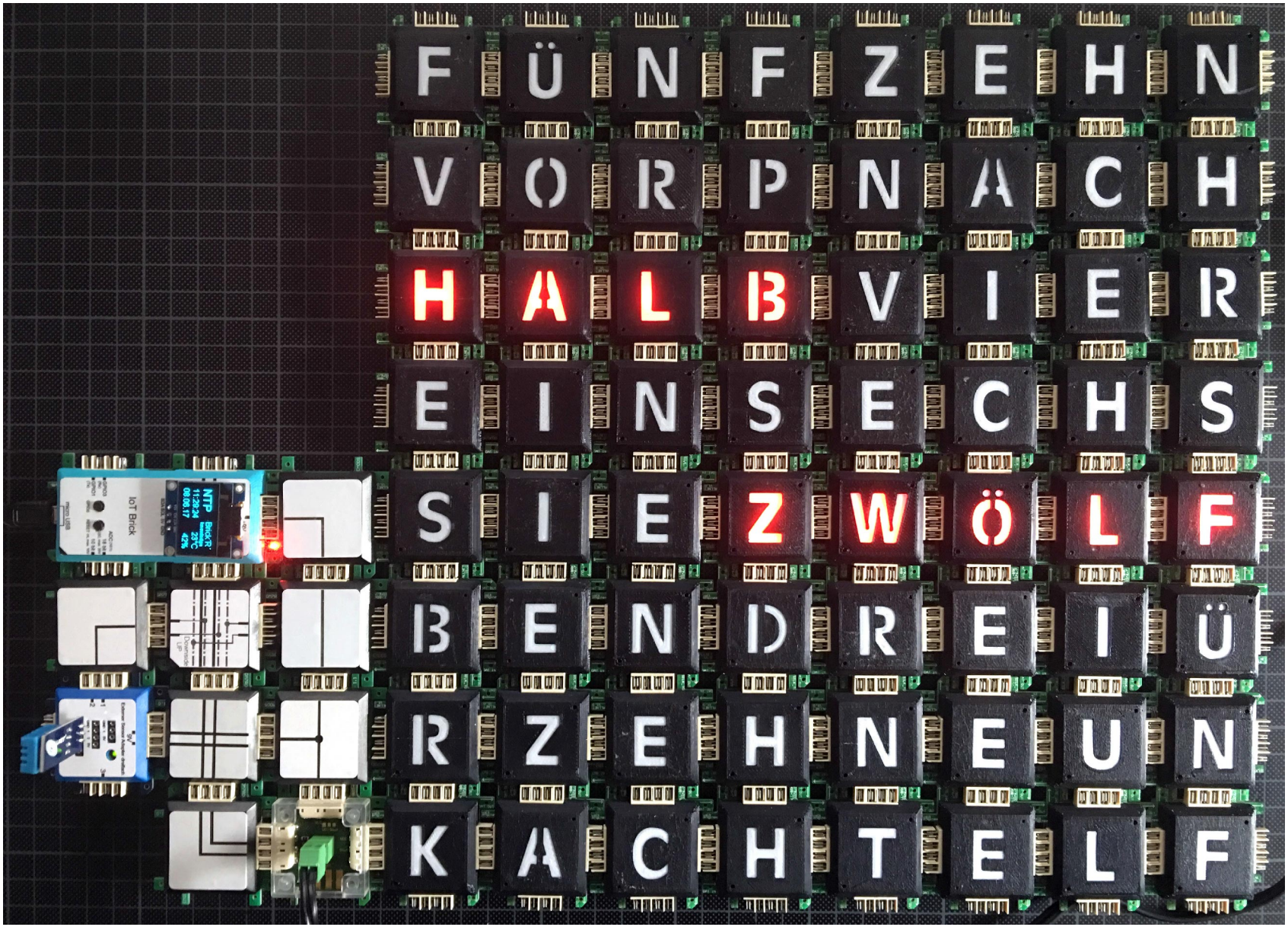
Temperaturanzeige mit der BRK-Clock:



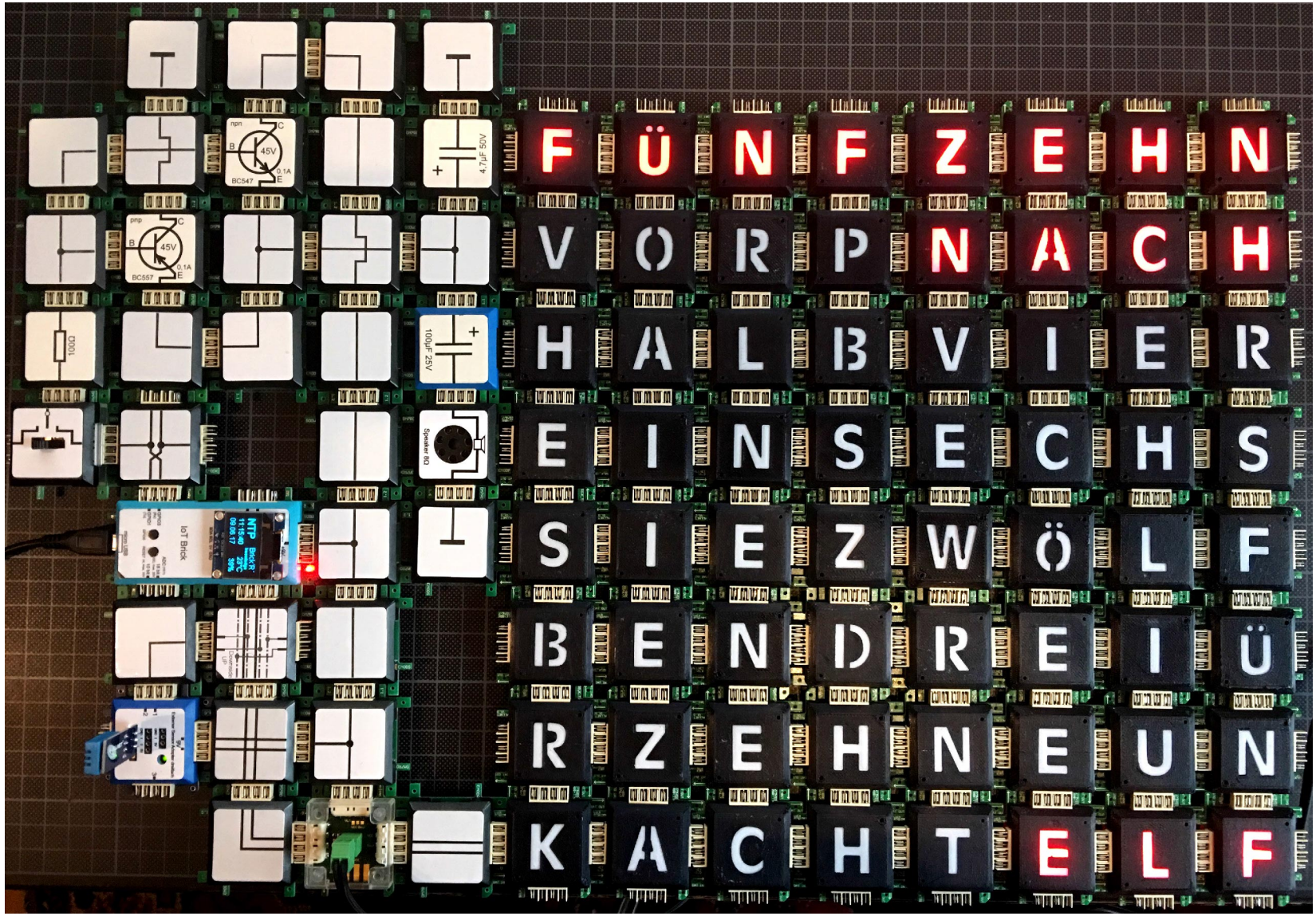
Ansteuerung der BRK-Clock mit einem IoT-Brick und Strommessung der Matrix sowie Spannungsmessung am Spannungswandler. Die Matrix verbraucht zwischen 0,8 und 1,5 A, je nachdem, wieviele RGBW-Bricks gerade eingeschaltet sind und in welcher Farbe diese leuchten. Der Selbttest braucht z.B. 1,3 A. Die RGBW-Bricks bestehen aus jeweils vier LEDs, nämlich rot, grün, blau und warmweiß.



Minimale Ansteuerung der BRK-Clock mit einem IoT-Brick und Temperatur/Feuchtigksits-Sensor:



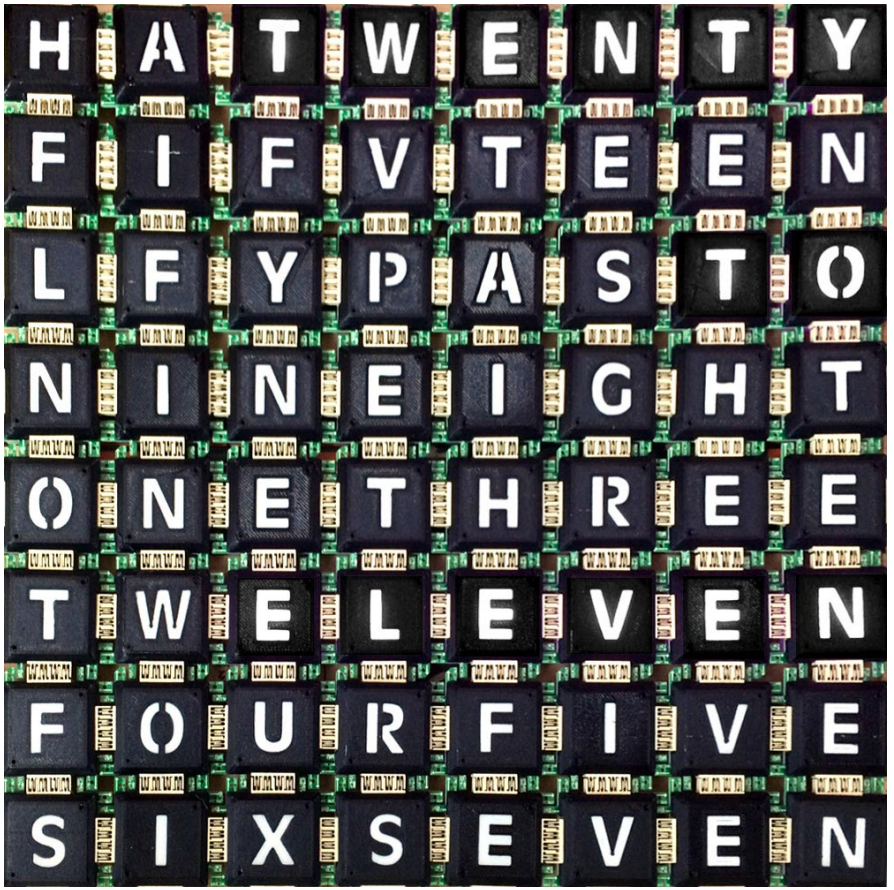
Voll ausgebaute Ansteuerung der BRK-Clock (ohne Terminal-Eingaben) mit einem IoT-Brick, Sensor und Sound-Verstärker-Endstufe.
Zum Flashen des IoT-Bricks muss der Schalter ausgeschaltet sein, damit das Programm auf den IoT-Brick übertragen werden kann.



Voll ausgebaute Ansteuerung der BRK-Clock mit einem IoT-Brick, Sensor und Sound-Verstärker-Endstufe. Hier wird alternativ der GPIO 13 benutzt. damit gibt es keinen Konflikt bei Terminal-Eingaben. Die Ansteuerung ist hier genau so hoch wie die Matrix.

Deutsche Belegung der RGBW-Matrix

Englische Belegung der RGBW-Matrix



Aktuell wird eine deutsche und eine englische Wort-Belegung unterstützt. Die oben angezeigten Buchstabenkombinationen für die RGBW-Matrix 8 x 8 sind in Originalgröße 32 x 32 cm groß. In der deutschen Belegung werden die Buchstaben "RK" links unten nur während der Startsequenz benutzt und das P in der zweiten Zeile überhaupt nicht. In der englischen Belegung wird nur das V in der zweiten Zeile überhaupt nicht benutzt,

```
//  Programm für eine Wort-Uhr aus 64 programmierbaren RGB-Bricks
//
//  Unter Verwendung der IoT Brick Library ab Version 1.03

#define DHT11_PIN 12                                                // Datenleitung des DHT11-Sensors an GPIO12 des IoT Bricks

#include <all_IoT.h>
#include <all_BRKclock.h>
#include <all_DHT11.h>


int    TemperatureVal  = 0;
String Temperature     = "";
String Humidity        = "";
String Lauf_Schrift    = "Brick'R'knowledge";

int    counter = 0;
int    m_alt = 0;
int    s_alt = 0;

int    BRKclock_displayMode     = 1;                                // 0 = Display aus,   1 = Display ein
int    BRKclock_secondTone      = 0;                                // 0 = Sekundenton aus,   1 = Sekundenton ein
int    BRKclock_minuteTone      = 0;                                // 0 = Minutenton aus,    1 = Minutenton ein
int    BRKclock_alarmTone       = 0;                                // 0 = Alarmton aus,      1 = Alarmton ein


void setup()
{
    t_TerminalInit();                                               // Terminal initialisieren
    IoT_Init(true);                                                 // IoT-Brick initialisieren
    BRKclock_Init();                                                // BRK-Clock initialisieren
    BRKclock_TestMatrix(10);                                        // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(2000, true);                                   // 2 Sekunden warten, kann mit Taster übersprungen werden
    Serial.println("");
    Serial.println("");
    if (IoT_Keypress())                                             // Wenn Taster gedrückt wird, Konfiguration im EEPROM löschen
    {
        IoT_EEPROMclear();                                            // Alle Parameter aus dem EEPROM löschen
        IoT_EEPROMupdate();                                           // EEPROM aktualisieren
        Serial.print("BRK-Clock Parameter aus EEPROM gelöscht. ");
        Serial.println("BRK-Clock wird neu gestartet...");
        delay(100);                                                   // 100 ms. warten
        BRKclock_Pixel.clear();                                       // Pixel löschen
        BRKclock_Pixel.show();                                        // Anzeige aktualisieren
        IoT_WaitNoKeypress();                                         // Warten bis der Taster losgelassen wurde
        t_Restart();                                                  // Neustart durchführen
    }
    BRKclock_WLANautoConnect(true);
    IoT_NTPinit();                                                  // NTP-Timeserver initialisieren
    DHT11_Init();
```

```
    if (BRKclock_ParameterLoad())                                   // Parameter-Block aus EEPROM laden
    {
        Serial.println("BRK-Clock Parameter aus EEPROM geladen.");
        BRKclock_ParameterApply();                                    // Parameter-Block in die aktuelle Konfiguration übernehmen
    }
    else
    {
        Serial.println("BRK-Clock Parameter im EEPROM nicht vorhanden.");
    }
    IoT_WebServer.on("/", handleRoot);                              // Browser-Handler (siehe unten) für das Stammverzeichnis
    IoT_WebServer.on("/form01", handleForm01);                      // Handler für die Eingabe der Namen in den Eingabefeldern
    IoT_WebServer.on("/form02", handleForm02);                      // Handler für "IoT-Brick neu starten"
    IoT_WebServer.on("/form03", handleForm03);                      // Handler für "IoT-Brick ausschalten"
    IoT_WebServer.on("/form04", handleForm04);                      // Handler für "BRK-Clock Selbsttest"
    IoT_WebServer.on("/form05", handleForm05);                      // Handler für "BRK-Clock Rechtecke"
    IoT_WebServer.on("/form06", handleForm06);                      // Handler für "BRK-Clock Kaleidoskop mit 8 Farben"
    IoT_WebServer.on("/form07", handleForm07);                      // Handler für "BRK-Clock Kaleidoskop mit 16 Farben"
    IoT_WebServer.on("/form08", handleForm08);                      // Handler für "BRK-Clock Laufschrift"
    IoT_WebServer.on("/form09", handleForm09);                      // Handler für "Temperatur anzeigen"
    IoT_WebServer.on("/form10", handleForm10);                      // Handler für "Schlangenlinien"
    IoT_WebServer.on("/form11", handleForm11);                      // Handler für "Weiße Welle über Regenbogen"
    IoT_WebServer.on("/form12", handleForm12);                      // Handler für "Weißes Pulsieren"
    IoT_WebServer.on("/form13", handleForm13);                      // Handler für "Regenbogen wird weiß"
    IoT_WebServer.on("/form14", handleForm14);                      // Handler für "Ambientebeleuchtung"
    IoT_WebServer.on("/form15", handleForm15);                      // Handler für "Regenbogenzyklus"
    IoT_WebServer.on("/form16", handleForm16);                      // Handler für "Regenbogen"
    IoT_WebServer.on("/form17", handleForm17);                      // Handler für "Einstellungen"
    IoT_WebServer.on("/form18", handleForm18);                      // Handler für "Startseite anzeigen"
    IoT_WebServer.on("/form19", handleForm19);                      // Handler für "Dynamische IP-Adresse"
    IoT_WebServer.on("/form20", handleForm20);                      // Handler für "Taster-Einstellung ändern"
    IoT_WebServer.on("/form21", handleForm21);                      // Handler für "Einstellungen speichern"
    IoT_WebServer.on("/form22", handleForm22);                      // Handler für "Standard Einstellungen wiederherstellen"
    IoT_WebServer.on("/form23", handleForm23);                      // Handler für "Dynamische IP-Adresse"
    IoT_WebServer.on("/form24", handleForm24);                      // Handler für "Demos"
    IoT_WebServer.begin();                                          // ab jetzt "horcht" der Server auf HTTP-Anfragen
    Serial.println("HTTP server started");
    BRKclock_SetClockRGBWColor(BRKclock_clockColorRed, BRKclock_clockColorGreen, BRKclock_clockColorBlue, BRKclock_clockColorWhite); // Schriftfarbe
    Serial.println("");
    snd_Init(13);                                                   // GPIO 13 für Sound-Ausgabe wählen
}

void BRKclock_ShowInfo (bool NTPvalid)
{
    IoT_Idle();
    IoT_WebServer.handleClient();                                   // Bediene die http Anfragen
    IoT_DisplayClear(24);                                           // OLED Display löschen (24 Punkt Schrift)
    IoT_DisplayDrawText(0, 0, "NTP");
    IoT_DisplayAlignText(TEXT_ALIGN_RIGHT);                         // Rechtsbündige Darstellung des Textes
    IoT_DisplayFontSize(16);
    IoT_DisplayDrawText(127, 0, "Brick´R´");
```

```
    IoT_DisplayFontSize(10);
    IoT_DisplayDrawText(121, 16, "knowledge");
    IoT_DisplayAlignText(TEXT_ALIGN_LEFT);                          // Linksbündige Darstellung des Textes
    if (NTPvalid)
    {
        IoT_DisplayFontSize(16);
        IoT_DisplayDrawText(0, 28, IoT_NTPtime());
        String d = IoT_NTPdate();
        IoT_DisplayDrawText(0, 48, left(d, 6) + right(d, 2));
        IoT_Idle();
        IoT_DisplayAlignText(TEXT_ALIGN_RIGHT);                      // Rechtsbündige Darstellung des Textes
        float t = DHT11_Temperature();                               // Temperatur auslesen (Celsius)
        if (not(isnan(t)))
        {
            TemperatureVal = int(t);
            Temperature = strrealform (t, 32, 1, 0, true, false) + "°C"; // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. Nullen nach dem Komma
            IoT_DisplayDrawText(127, 28, Temperature);
        }
        IoT_Idle();
        float h = DHT11_Humidity();                                  //Feuchtigkeit auslesen (Prozent)
        if (not(isnan(h)))
        {
            Humidity = strrealform (h, 32, 1, 0, true, false) + "%";    // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. Nullen nach dem Komma
            IoT_DisplayDrawText(127, 48, Humidity);
        }
    }
    else
    {
        IoT_DisplayFontSize(16);
        IoT_DisplayDrawText(0, 28, "Uhrzeit laden...");
        IoT_DisplayDrawText(0, 46, "Bitte warten.");
    }
}

void loop()
{
    IoT_Idle();
    IoT_WebServer.handleClient();                                   // Bediene die http Anfragen
    IoT_Idle();
    int h = hour();
    int m = minute();
    int s = second();
    bool NTPvalid = IoT_NTPvalid();

    if ((m_alt != m) && (NTPvalid))                                 // Minute hat sich geändert und NTP-Time wurde empfangen
    {
        if (BRKclock_minuteTone == 1)
        {
            snd_beep();
            delay(500);
```

```
        IoT_Idle();
    }
    m_alt = m;
    s_alt = s;
    BRKclock_International(h, m);
}
else if ((s_alt != s) && (NTPvalid))                            // Sekunde hat sich geändert und NTP-Time wurde empfangen
{
    if (BRKclock_secondTone == 1)
    {
        snd_click();
        delay(5);
    }
    s_alt = s;
}
else
{
    delay(100);
    IoT_Idle();
}

if (IoT_Keypress())                                             // Wenn Taster gedrückt wird, Konfiguration zeigen
{
    if (BRKclock_buttonMode == 0)                                 // IP-Adresse im Display anzeigen
    {
        IoT_DisplayWLANstatus();
        IoT_DisplayUpdate();                                        // OLED-Anzeige aktualisieren
        String ip = "IP: " + IoT_WLANaddress(true) + "  GW: " + IoT_WLANgateway(true) + "  NT: " + IoT_WLANsubnet(true);
        if (counter % 10 == 0) //Infos seriell nicht zu oft ausgeben (ca. jede Sekunden, nur wenn der Button gedrückt wird)
        {
            Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
            Serial.print("WiFi is ");
            Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
            Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
            Serial.println(ip);                                       // IP-Information
        }
        IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
        BRKclock_Print(ip, 250);                                    // Laufschrift-Text mit IP-Information, 250 ms. Pause
        IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    }
    else if (BRKclock_buttonMode == 1)                            // Temperatur anzeigen
    {
        IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
        BRKclock_PrintTemperature(TemperatureVal, 0);               // Temperatur anzeigen & 10 Sekunden warten
        IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    }
    else if (BRKclock_buttonMode == 2)                            // Rechtecke starten
    {
        BRKclock_Pixel.clear();                                     // Pixel löschen
        BRKclock_Pixel.show();                                      // Anzeige aktualisieren
```

```
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_Rectangles();                                // Rechtecke-Demo
    }
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
}
else if (BRKclock_buttonMode == 3)                         // Kaleidoskop mit 16 Farben starten
{
    BRKclock_Pixel.clear();                                 // Pixel löschen
    BRKclock_Pixel.show();                                  // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_Kaleidoscope(8, true);                       // Kaleidoskop mit 16 Farben
    }
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
}
else if (BRKclock_buttonMode == 4)                         // Schlangenlinien starten
{
    BRKclock_Pixel.clear();                                 // Pixel löschen
    BRKclock_Pixel.show();                                  // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_SnakeLines(50);                              // Schlangenlinien
    }
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
}
else if (BRKclock_buttonMode == 5)                         // Weiße Welle über Regenbogen starten
{
    BRKclock_Pixel.clear();                                 // Pixel löschen
    BRKclock_Pixel.show();                                  // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_WhiteOverRainbow(20, 75, 5);                 // Weiße Welle über Regenbogen
    }
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
}
else if (BRKclock_buttonMode == 6)                         // Ambientebeleuchtung starten
{
    BRKclock_Pixel.clear();                                 // Pixel löschen
    BRKclock_Pixel.show();                                  // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_AmbienceColors(15000);                       // Ambientebeleuchtung, 15 Sekunden Pause
    }
    IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
```

```
    }
    else if (BRKclock_buttonMode == 7)                  // Regenbogen starten
    {
        BRKclock_Pixel.clear();                           // Pixel löschen
        BRKclock_Pixel.show();                            // Anzeige aktualisieren
        IoT_WaitNoKeypress();                             // Warten bis der Taster losgelassen wurde
        while (not(IoT_Keypress()))
        {
            BRKclock_Rainbow(50);                           // Regenbogen
        }
        IoT_WaitNoKeypress();                             // Warten bis der Taster losgelassen wurde
    }
    else if (BRKclock_buttonMode == 8)                  // Regenbogenzyklus starten
    {
        BRKclock_Pixel.clear();                           // Pixel löschen
        BRKclock_Pixel.show();                            // Anzeige aktualisieren
        IoT_WaitNoKeypress();                             // Warten bis der Taster losgelassen wurde
        while (not(IoT_Keypress()))
        {
            BRKclock_RainbowCycle(50);                      // Regenbogenzyklus
        }
        IoT_WaitNoKeypress();                             // Warten bis der Taster losgelassen wurde
    }
    BRKclock_International(hour(), minute());           // Normale Zeitanzeige einschalten
}
else
{
    if ((counter % 25 == 0) && not(NTPvalid))           // NTP Event alle 2 Sekunden auslesen wenn noch nicht empfangen
    {
        if (IoT_NTPeventAvail)                            // Zeitevent-Infos im Terminal ausgeben
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }
    else if ((counter % 7500 == 0) && not(NTPvalid))    // NTP Event alle 10 Minuten auslesen wenn bereits empfangen
    {
        if (IoT_NTPeventAvail)                            // Zeitevent-Infos im Terminal ausgeben
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }
    else
    {
        BRKclock_ShowInfo(NTPvalid);
    }
}

IoT_Idle();
```

```
if (BRKclock_displayMode == 0)
{
    IoT_DisplayClear(24);                                       // OLED Display löschen (24 Punkt Schrift)
}
IoT_DisplayUpdate();                                           // OLED-Anzeige aktualisieren

if (BRKclock_alarmMode == 1)                                   // Wecker prüfen
{
    if ((h == BRKclock_alarmHour) && (m == BRKclock_alarmMinute)) // Weckzeit erreicht?
    {
        byte i = 0;
        while ((i < 60) && not(IoT_Keypress()))                    // 60 Sekunden Alarm oder bis zum Drücken des Tasters
        {
            if (not(IoT_Keypress()))
            {
                BRKclock_Pixel.clear();                                // Pixel löschen
                BRKclock_Pixel.show();                                 // Anzeige aktualisieren
                if (BRKclock_alarmTone)
                {
                    snd_beep();
                }
                IoT_Idle();
                IoT_WaitKeypress(500, false);                          // 500 ms. warten, Wartezeit wird durch den Button unterbrochen
                BRKclock_International(h, m);
                if (not(IoT_Keypress()))
                {
                    long t = millis();
                    BRKclock_ShowInfo(NTPvalid);                         // Informationen aktualisieren, kann bis zu 220 ms. dauern
                    IoT_DisplayUpdate();                                 // OLED-Anzeige aktualisieren
                    IoT_Idle();
                    t = 500 - (millis() - t);
                    if (t > 0)
                    {
                        IoT_WaitKeypress(t, false);                        // ≤500 ms. warten, Wartezeit wird durch den Button unterbrochen
                    }
                }
            }
            i++;
        }
        BRKclock_alarmMode = 2;                                    // Alarm beendet, verhindert dass der Alarm sofort wieder beginnt
    }
}
else if (BRKclock_alarmMode == 2)                              // Alarm beendet
{
    if (not((h == BRKclock_alarmHour) && (m == BRKclock_alarmMinute)))  // Weckzeit vorbei?
    {
        BRKclock_alarmMode = 1;                                    // Normalen Alarm-Modus wiederherstellen
    }
}
```

```
    IoT_Idle();
    counter++;
}

//Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot ()
{
    IoT_Idle();
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    String alarm_web = "";
    if (BRKclock_alarmMode == 1)                                                    // Wecker eingeschaltet
    {
        alarm_web = strform(BRKclock_alarmHour, 48, 2, false) + ":" + strform(BRKclock_alarmMinute, 48, 2, false) + ":00";
    }
    else
    {
        alarm_web = "(ausgeschaltet)";
    }
    String ipaddr_web  = IoT_WLANaddress(false);
    String title_web = "Brick'R'knowledge BRK-Clock Internetseite";
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String ColorHexVal = "";
    String staticInfo = " (dynamisch)";
    if (BRKclock_staticIP == 1)
    {
        staticInfo = " (statisch)";
    }
    switch (BRKclock_colorMode)
    {
        case 0: // Random
            ColorHexVal = "#000000";
            break;
        case 1: // Weiß
            ColorHexVal = "#000000";
            break;
        case 2: // Rot
            ColorHexVal = "#FF0000";
            break;
        case 3: // Grün
            ColorHexVal = "#00FF00";
            break;
        case 4: // Blau
            ColorHexVal = "#0000FF";
            break;
        case 5: // Türkis
            ColorHexVal = "#00FFFF";
```

```
        break;
    case 6: // Violett
        ColorHexVal = "#FF00FF";
        break;
    case 7: // Gelb
        ColorHexVal = "#FFFF00";
        break;
    case 8: // Orange
        ColorHexVal = "#FF8000";
        break;
    default: // Unbekannte Auswahl
        ColorHexVal = "#000000";
        break;
}
String content =                                                // Die HTML-Seite in lesbarer Formatierung,
// darf auch Variablen enthalten
"<html>\
<head>\
<title>" + title_web + "</title>\
<style>\
body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
</style>\
<style type='text/css'>\
h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
</style>\
</head>\
<body>\
<h1>" + title_web + "</h1>\
<p>" + cleanhtml(fillcenter(" Informationen ", "-", 64)) + "</p>\
<p> </p>\
<p>" + cleanhtml("Datum:          " + date_web)    + "</p>\
<p>" + cleanhtml("Uhrzeit:        " + time_web)    + "</p>\
<p>" + cleanhtml("Weckzeit:       " + alarm_web)   + "</p>\
<p>" + cleanhtml("Temperatur:     " + Temperature) + "</p>\
<p>" + cleanhtml("Feuchtigkeit:   " + Humidity)    + "</p>\
<p>" + cleanhtml("IP-Adresse:     " + ipaddr_web)  + staticInfo + "</p>\
<p>" + cleanhtml("Öffentliche IP: ") + IoT_WebClientIP() + "</p>\
<p>" + cleanhtml(fillcenter(" BRK-Clock Einstellungen ", "-", 64)) + "</p>\
<p> </p>\
" + IoT_WebFormOpen("form01") + "\
" + IoT_WebCheckBox(" Wecker eingeschaltet  ", "alarm", "chkd", sgn(BRKclock_alarmMode))
+ IoT_WebInput("H:", "wh", str(BRKclock_alarmHour), 5) + IoT_WebInput(" M:", "wm", strform(BRKclock_alarmMinute, 48, 2, false), 5) + "<br>\
" + IoT_WebCheckBox(" Weckton eingeschaltet  ", "BRKclock_alarmTone", "chkd", BRKclock_alarmTone) + "<br>\
<br>\
" + IoT_WebCheckBox(" Sekundenton eingeschaltet  ", "sectone", "chkd", BRKclock_secondTone) + "<br>\
" + IoT_WebCheckBox(" Minutenton eingeschaltet  ", "mintone", "chkd", BRKclock_minuteTone) + "<br>\
<br>\
" + IoT_WebCheckBox(" Temperaturanzeige jede Minute für 5 Sekunden", "display", "chkd", BRKclock_temperatureMode) + "<br>\
<br>\
" + IoT_WebCheckBox(" OLED-Display eingeschaltet", "display", "chkd", BRKclock_displayMode) + "<br>\
```

```
    <br>\
    " + IoT_WebRadioButton(" Zufällige Farbe",  0, "COLORgroup", "random",  BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Weiße Schrift",    1, "COLORgroup", "black",   BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Rote Schrift",     2, "COLORgroup", "red",     BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Grüne Schrift",    3, "COLORgroup", "green",   BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Blaue Schrift",    4, "COLORgroup", "blue",    BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Türkise Schrift",  5, "COLORgroup", "cyan",    BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Violette Schrift", 6, "COLORgroup", "magenta", BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Gelbe Schrift",    7, "COLORgroup", "yellow",  BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Orange Schrift",   8, "COLORgroup", "orange",  BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" RGBW Schrift    ", 9, "COLORgroup", "user",    BRKclock_colorMode)
    + IoT_WebInput("R:", "ur", str(BRKclock_clockColorRed), 5) + IoT_WebInput(" G:", "ug", str(BRKclock_clockColorGreen), 5)
    + IoT_WebInput(" B:", "ub", str(BRKclock_clockColorBlue), 5) + IoT_WebInput(" W:", "uw", str(BRKclock_clockColorWhite), 5) + "<br>\
    " + IoT_WebHtab (26) + cleanhtml("RGBW Temperatur ")
    + IoT_WebInput("R:", "tr", str(BRKclock_tempColorRed), 5) + IoT_WebInput(" G:", "tg", str(BRKclock_tempColorGreen), 5)
    + IoT_WebInput(" B:", "tb", str(BRKclock_tempColorBlue), 5) + IoT_WebInput(" W:", "tw", str(BRKclock_tempColorWhite), 5) + "<br>\
    <br>\
    " + IoT_WebFormSubmitButton("Einstellungen der BRK-Clock anwenden") + "<br>\
    " + IoT_WebFormClose() + "\
    <p>" + cleanhtml(fillcenter(" Demos ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form24", "Demos") + "\
    <p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form17", "System-Einstellungen") + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                                  // HTTP-Seite aktualisieren
}

void handleForm01 ()
{
    Serial.println("Button wurde betätigt: Form01");
    IoT_Idle();

    BRKclock_temperatureMode = boolval(IoT_WebTestField("display"));          // Testen, ob die Checkbox Display angekreuzt wurde
    BRKclock_secondTone  = boolval(IoT_WebTestField("sectone"));              // Testen, ob die Checkbox Sekundenton angekreuzt wurde
    BRKclock_minuteTone  = boolval(IoT_WebTestField("mintone"));              // Testen, ob die Checkbox Minutenton angekreuzt wurde
    BRKclock_displayMode = boolval(IoT_WebTestField("display"));              // Testen, ob die Checkbox Display angekreuzt wurde
    BRKclock_alarmTone   = boolval(IoT_WebTestField("BRKclock_alarmTone"));   // Testen, ob die Checkbox Alarmton angekreuzt wurde
    BRKclock_alarmMode   = boolval(IoT_WebTestField("alarm"));                // Testen, ob die Checkbox Wecker angekreuzt wurde
    BRKclock_alarmHour       = val(IoT_WebGetField("wh"));
    BRKclock_alarmMinute     = val(IoT_WebGetField("wm"));

    if (IoT_WebTestField("COLORgroup"))                                  // Testen, ob Felder mit dem Namen "COLORgroup" existieren
    {
        String selection = IoT_WebGetField("COLORgroup");
        if (selection == "random")                                         // Schwarzer Text im Web, zufällige Farbe auf der Uhr
        {
```

```
    BRKclock_colorMode = 0;                                     // Ab der nächsten Minute die Farbe ändern
    BRKclock_SetClockRandomColor();                             // Zufällige Farbe erzeugen
    BRKclock_SetTemperatureRGBWColor(BRKclock_clockColorRed, BRKclock_clockColorGreen, BRKclock_clockColorBlue, BRKclock_clockColorWhite);
}
else if (selection == "black")                                  // Schwarzer Text im Web, weiße Schrift auf der Uhr
{
    BRKclock_colorMode = 1;
    BRKclock_SetClockRGBWColor(0xFF, 0xFF, 0xFF, 0);
    BRKclock_SetTemperatureRGBWColor(0xFF, 0xFF, 0xFF, 0);
}
else if (selection == "red")                                    // Rot
{
    BRKclock_colorMode = 2;
    BRKclock_SetClockRGBWColor(0xFF, 0x00, 0x00, 0);
    BRKclock_SetTemperatureRGBWColor(0xFF, 0x00, 0x00, 0);
}
else if (selection == "green")                                  // Grün
{
    BRKclock_colorMode = 3;
    BRKclock_SetClockRGBWColor(0x00, 0xFF, 0x00, 0);
    BRKclock_SetTemperatureRGBWColor(0x00, 0xFF, 0x00, 0);
}
else if (selection == "blue")                                   // Blau
{
    BRKclock_colorMode = 4;
    BRKclock_SetClockRGBWColor(0x00, 0x00, 0xFF, 0);
    BRKclock_SetTemperatureRGBWColor(0x00, 0x00, 0xFF, 0);
}
else if (selection == "cyan")                                   // Türkis
{
    BRKclock_colorMode = 5;
    BRKclock_SetClockRGBWColor(0x00, 0xFF, 0xFF, 0);
    BRKclock_SetTemperatureRGBWColor(0x00, 0xFF, 0xFF, 0);
}
else if (selection == "magenta")                                // Violett
{
    BRKclock_colorMode = 6;
    BRKclock_SetClockRGBWColor(0xFF, 0x00, 0xFF, 0);
    BRKclock_SetTemperatureRGBWColor(0xFF, 0x00, 0xFF, 0);
}
else if (selection == "yellow")                                 // Gelb
{
    BRKclock_colorMode = 7;
    BRKclock_SetClockRGBWColor(0xFF, 0xFF, 0x00, 0);
    BRKclock_SetTemperatureRGBWColor(0xFF, 0xFF, 0x00, 0);
}
else if (selection == "orange")                                 // Orange
{
    BRKclock_colorMode = 8;
    BRKclock_SetClockRGBWColor(0xFF, 0x80, 0x00, 0);
```

```
            BRKclock_SetTemperatureRGBWColor(0xFF, 0x80, 0x00, 0);
        }
        else if (selection == "user")                                   // Benutzerdefinierte Farbe
        {
            BRKclock_colorMode = 9;
            BRKclock_SetClockRGBWColor      (val(IoT_WebGetField("ur")),
                                             val(IoT_WebGetField("ug")),
                                             val(IoT_WebGetField("ub")),
                                             val(IoT_WebGetField("uw")));
            BRKclock_SetTemperatureRGBWColor(val(IoT_WebGetField("tr")),
                                             val(IoT_WebGetField("tg")),
                                             val(IoT_WebGetField("tb")),
                                             val(IoT_WebGetField("tw")));
        }
        BRKclock_International(hour(), minute());
    }
    IoT_WaitNoKeypress();                                               // Warten bis der Taster losgelassen wurde
    handleRoot();                                                       // Wieder auf die Hauptseite zurück schalten
}

void handleForm02 ()                                                    // Neustart
{
    Serial.println("Button wurde betätigt: Form02");
    IoT_Idle();
    BRKclock_Clear(true);                                               // Pixel löschen & Anzeige aktualisieren
    IoT_WaitNoKeypress();                                               // Warten bis der Taster losgelassen wurde
    handleForm17();                                                     // Einstellungen anzeigen
    t_Restart();
}

void handleForm03 ()                                                    // Ausschalten
{
    Serial.println("Button wurde betätigt: Form03");
    BRKclock_Clear(true);                                               // Pixel löschen & Anzeige aktualisieren
    handleForm17();                                                     // Einstellungen anzeigen
    IoT_ShutDown();
}

void handleForm04 ()                                                    // Selbsttest
{
    Serial.println("Button wurde betätigt: Form04");
    handleForm24();                                                     // Demo-Seite anzeigen
    BRKclock_Clear(false);                                              // Pixel löschen & Anzeige aktualisieren
    BRKclock_TestMatrix(50);                                            // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(0, false);                                         // Auf Taster warten
    IoT_WaitNoKeypress();                                               // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                           // Normale Zeitanzeige einschalten
}

void handleForm05 ()                                                    // Rechtecke Demo
```

```
{
    Serial.println("Button wurde betätigt: Form05");
    handleForm24();                                          // Demo-Seite anzeigen
    BRKclock_Rectangles();                                   // Rechtecke-Demo
    IoT_WaitNoKeypress();                                    // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                // Normale Zeitanzeige einschalten
}

void handleForm06 ()                                        // Kaleidoskop mit 8 Farben
{
    Serial.println("Button wurde betätigt: Form06");
    handleForm24();                                          // Demo-Seite anzeigen
    BRKclock_Kaleidoscope(8, false);
    IoT_WaitNoKeypress();                                    // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                // Normale Zeitanzeige einschalten
}

void handleForm07 ()                                        // Kaleidoskop mit 16 Farben
{
    Serial.println("Button wurde betätigt: Form07");
    handleForm24();                                          // Demo-Seite anzeigen
    BRKclock_Kaleidoscope(8, true);
    IoT_WaitNoKeypress();                                    // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                // Normale Zeitanzeige einschalten
}

void handleForm08 ()                                        // Laufschrift
{
    Serial.println("Button wurde betätigt: Form08");
    handleForm24();                                          // Demo-Seite anzeigen
    Lauf_Schrift = replace(IoT_WebGetField("lauf"), chr(160), " ");   // Geschütztes Leerzeichen in Leerzeichen ersetzen
    BRKclock_Print(Lauf_Schrift, 100);                       // Laufschrift-Text von der Webseite laden
    IoT_WaitNoKeypress();                                    // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                // Normale Zeitanzeige einschalten
}

void handleForm09 ()                                        // Temperatur anzeigen
{
    Serial.println("Button wurde betätigt: Form09");
    handleForm24();                                          // Demo-Seite anzeigen
    BRKclock_PrintTemperature(TemperatureVal, 10000);        // Temperatur anzeigen & 10 Sekunden warten
    IoT_WaitNoKeypress();                                    // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                // Normale Zeitanzeige einschalten
}

void handleForm10 ()                                        // Schlangenlinien
{
    Serial.println("Button wurde betätigt: Form10");
    handleForm24();                                          // Demo-Seite anzeigen
    BRKclock_Clear(false);                                   // Pixel löschen & Anzeige aktualisieren
```

```
    BRKclock_SnakeLines(50);                                    // Schlangenlinien
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm11 ()                                          // Weiße Welle über Regenbogen
{
    Serial.println("Button wurde betätigt: Form11");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_WhiteOverRainbow(20, 75, 5);                       // Weiße Welle über Regenbogen
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm12 ()                                          // Weißes Pulsieren
{
    Serial.println("Button wurde betätigt: Form12");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_PulseWhite(5, 4);
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm13 ()                                          // Regenbogen wird weiß
{
    Serial.println("Button wurde betätigt: Form13");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_RainbowFade2White(3,3,1);
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm14 ()                                          // Ambientebeleuchtung
{
    Serial.println("Button wurde betätigt: Form14");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_AmbienceColors(1000);                              // Ambientebeleuchtung
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}
```

```
void handleForm15 ()                                                    // Regenbogenzyklus
{
    Serial.println("Button wurde betätigt: Form15");
    handleForm24();                                                       // Demo-Seite anzeigen
    BRKclock_Clear(false);                                                // Pixel löschen & Anzeige aktualisieren
    BRKclock_RainbowCycle(50);                                            // Regenbogenzyklus
    IoT_WaitNoKeypress();                                                 // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                             // Normale Zeitanzeige einschalten
}

void handleForm16 ()                                                    // Regenbogen
{
    Serial.println("Button wurde betätigt: Form16");
    handleForm24();                                                       // Demo-Seite anzeigen
    BRKclock_Clear(false);                                                // Pixel löschen & Anzeige aktualisieren
    BRKclock_Rainbow(50);                                                 // Regenbogen
    IoT_WaitNoKeypress();                                                 // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                             // Normale Zeitanzeige einschalten
}

void handleForm17 ()
{
    Serial.println("Button wurde betätigt: Form17");
    IoT_Idle();
    String ColorHexVal = "#000000";
    String ipaddr_web  = IoT_WLANaddress(false);
    String ipaddr_gateway  = IoT_WLANgateway(false);
    String ipaddr_subnet  = IoT_WLANsubnet(false);
    String title_web = "Brick'R'knowledge BRK-Clock System";
    String staticInfo = " (dynamisch)";
    if (BRKclock_staticIP == 1)
    {
        staticInfo = " (statisch)";
    }
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String content =                                                      // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
    "<html>\
    <head>\
    <title>" + title_web + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
```

```
</head>\
<body>\
<h1>" + title_web + "</h1>\
<p>" + cleanhtml(fillcenter(" Aktuelle Netzwerk-Einstellungen ", "-", 64)) + "</p>\
<p> </p>\
<p>" + cleanhtml("IP-Addresse   : " + ipaddr_web)      + "</p>\
<p>" + cleanhtml("Gateway      : " + ipaddr_gateway)  + "</p>\
<p>" + cleanhtml("Subnetzmaske  : " + ipaddr_subnet)   + "</p>\
<p> </p>\
<p>" + cleanhtml("Öffentliche IP: ") + IoT_WebClientIP() + "</p>\
<p>" + cleanhtml(fillcenter(" Netzwerk ", "-", 64)) + "</p>\
<p> </p>\
" + IoT_WebFormOpen("form19") + "\
<p>" + cleanhtml("         IP-Adresse     Gateway     Subnetzmaske") + "</p>\
" + IoT_WebFormSubmitButton("Ändern") + "\
" + IoT_WebInput(" ", "ip01", ipaddr_web, 15) + "\
" + IoT_WebInput(" ", "ip02", ipaddr_gateway, 15) + "\
" + IoT_WebInput(" ", "ip03", ipaddr_subnet, 15) + staticInfo + "\
" + IoT_WebFormClose() + "\
" + IoT_WebFormActionButton ("form23", "Dynamische IP-Adresse") + "\
<p> </p>\
<p>" + cleanhtml(fillcenter(" Sprache & Taster ", "-", 64)) + "</p>\
<p> </p>\
" + IoT_WebFormOpen("form20") + "\
" + IoT_WebRadioButton(" Deutsche Wort-Uhr",           0, "LANGgroup",   "DE",  BRKclock_language)   + "<br>\
" + IoT_WebRadioButton(" English Word-Clock",          1, "LANGgroup",   "EN",  BRKclock_language)   + "<br>\
<p> </p>\
" + IoT_WebRadioButton(" IP-Adresse anzeigen",         0, "BTgroup", "butnIP",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Temperatur anzeigen",         1, "BTgroup", "butnTP",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Rechtecke",                   2, "BTgroup", "butn02",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Kaleidoskop",                 3, "BTgroup", "butn03",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Schlangenlinien",             4, "BTgroup", "butn04",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Weiße Welle über Regenbogen", 5, "BTgroup", "butn05",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Ambientebeleuchtung",         6, "BTgroup", "butn06",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Regenbogen",                  7, "BTgroup", "butn07",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Regenbogenzyklus",            8, "BTgroup", "butn08",  BRKclock_buttonMode) + "<br>\
<br>\
" + IoT_WebFormSubmitButton("Sprache & Taster-Einstellung ändern") + "<br>\
" + IoT_WebFormClose() + "\
<p> </p>\
<p>" + cleanhtml(fillcenter(" Einstellungen ", "-", 64)) + "</p>\
<p> </p>\
" + IoT_WebFormActionButton ("form22", "Standard Einstellungen wiederherstellen") + "\
" + IoT_WebFormActionButton ("form21", "Aktuelle Einstellungen dauerhaft speichern") + "\
<p> </p>\
<p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
<p> </p>\
" + IoT_WebFormActionButton ("form18", "Startseite anzeigen")   + "\
" + IoT_WebFormActionButton ("form02", "BRK-Clock neu starten") + "\
" + IoT_WebFormActionButton ("form03", "BRK-Clock ausschalten") + "\
```

```
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                        // HTTP-Seite aktualisieren
}

void handleForm18 ()                                                // Startseite anzeigen
{
    Serial.println("Button wurde betätigt: Form18");
    handleRoot();                                                   // Wieder auf die Hauptseite zurück schalten
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                       // Normale Zeitanzeige einschalten
}

void handleForm19 ()                                                // Statische IP-Adresse
{
    Serial.println("Button wurde betätigt: Form19");
    BRKclock_staticIP = 1;
    IPAddress ip1 = IPval(IoT_WebGetField("ip01"));                 // Statische IP-Adresse
    IPAddress ip2 = IPval(IoT_WebGetField("ip02"));                 // Gateway/Router IP-Adresse
    IPAddress ip3 = IPval(IoT_WebGetField("ip03"));                 // Subnet IP-Adresse
    WiFi.config(ip1, ip2, ip3);                                     // Parameter: IP, Gateway, Subnet, DNS
    handleForm17();                                                 // Einstellungen anzeigen
}

void handleForm20 ()
{
    Serial.println("Button wurde betätigt: Form20");
    if (IoT_WebTestField("BTgroup"))                                // Testen, ob Felder mit dem Namen "BUTTONgroup" existieren
    {
        String selection = IoT_WebGetField("BTgroup");
        if (selection == "butnIP")
        {
            BRKclock_buttonMode = 0;                                    // IP-Adresse anzeigen
        }
        else if (selection == "butnTP")
        {
            BRKclock_buttonMode = 1;                                    // Temperatur anzeigen
        }
        else if (selection == "butn02")
        {
            BRKclock_buttonMode = 2;                                    // Rechtecke
        }
        else if (selection == "butn03")
        {
            BRKclock_buttonMode = 3;                                    // Kaleidoskop
        }
        else if (selection == "butn04")
        {
            BRKclock_buttonMode = 4;                                    // Schlangenlinien
```

```
        }
        else if (selection == "butn05")
        {
            BRKclock_buttonMode = 5;                                    // Weiße Welle über Regenbogen
        }
        else if (selection == "butn06")
        {
            BRKclock_buttonMode = 6;                                    // Ambientebeleuchtung
        }
        else if (selection == "butn07")
        {
            BRKclock_buttonMode = 7;                                   // Regenbogen
        }
        else if (selection == "butn08")
        {
            BRKclock_buttonMode = 8;                                    // Regenbogenzyklus
        }
    }
    if (IoT_WebTestField("LANGgroup"))                                // Testen, ob Felder mit dem Namen "LANGgroup" existieren
    {
        String selection = IoT_WebGetField("LANGgroup");
        if (selection == "DE")
        {
            BRKclock_language = 0;                                      // Deutsche Wort-Uhr
            BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
        }
        else if (selection == "EN")
        {
            BRKclock_language = 1;                                      // Englische Wort-Uhr
            BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
        }
    }
    IoT_WaitNoKeypress();                                             // Warten bis der Taster losgelassen wurde
    handleForm17();                                                   // Einstellungen anzeigen
}

void handleForm21 ()                                                 // Einstellungen speichern (in das EEPROM)
{
    Serial.println("Button wurde betätigt: Form21");
    BRKclock_ParameterCreate();                                       // Aktuelle Konfiguration in den Parameter-Block schreiben
    BRKclock_ParameterSave();                                         // Parameter-Block in das EEPROM sichern
    IoT_EEPROMupdate();                                               // EEPROM aktualisieren
    handleForm17();                                                   // Einstellungen anzeigen
}

void handleForm22 ()                                                 // Standard Einstellungen wiederherstellen
{
    Serial.println("Button wurde betätigt: Form22");
    BRKclock_ParameterNew();                                          // Aktuelle Konfiguration in den Parameter-Block schreiben
    BRKclock_ParameterApply();                                        // Parameter-Block in die aktuelle Konfiguration übernehmen
```

```
    handleForm17();                                                 // Einstellungen anzeigen
}

void handleForm23 ()                                               // Dynamische IP-Adresse
{
    Serial.println("Button wurde betätigt: Form23");
    BRKclock_staticIP = 0;
    IPAddress ip1 = IPAddress(IoT_dynamicIP_address0, IoT_dynamicIP_address1, IoT_dynamicIP_address2, IoT_dynamicIP_address3);
    IPAddress ip2 = IPAddress(IoT_dynamicIP_gateway0, IoT_dynamicIP_gateway1, IoT_dynamicIP_gateway2, IoT_dynamicIP_gateway3);
    IPAddress ip3 = IPAddress(IoT_dynamicIP_subnet0,  IoT_dynamicIP_subnet1,  IoT_dynamicIP_subnet2,  IoT_dynamicIP_subnet3);
    WiFi.config(ip1, ip2, ip3);                                    // Parameter: IP, Gateway, Subnet, DNS
    handleForm17();                                                 // Einstellungen anzeigen
}

void handleForm24 ()
{
    Serial.println("Button wurde betätigt: Form24");
    IoT_Idle();
    String ColorHexVal = "#000000";
    String ipaddr_web  = IoT_WLANaddress(false);
    String ipaddr_gateway  = IoT_WLANgateway(false);
    String ipaddr_subnet  = IoT_WLANsubnet(false);
    String title_web = "Brick'R'knowledge BRK-Clock Demos";
    String staticInfo = " (dynamisch)";
    if (BRKclock_staticIP == 1)
    {
        staticInfo = " (statisch)";
    }
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String content =                                                // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
    "<html>\
    <head>\
    <title>" + title_web + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p>" + cleanhtml(fillcenter(" Demos ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form04", "BRK-Clock Selbsttest") + "\
    " + IoT_WebFormActionButton ("form09", "BRK-Clock Temperatur") + "\
    " + IoT_WebFormActionButton ("form05", "BRK-Clock Rechtecke") + "\
```

```
    " + IoT_WebFormActionButton ("form06", "BRK-Clock Kaleidoskop mit 8 Farben") + "\
    " + IoT_WebFormActionButton ("form07", "BRK-Clock Kaleidoskop mit 16 Farben") + "\
    " + IoT_WebFormActionButton ("form10", "BRK-Clock Schlangenlinien") + "\
    " + IoT_WebFormActionButton ("form11", "BRK-Clock Weiße Welle über Regenbogen") + "\
    " + IoT_WebFormActionButton ("form12", "BRK-Clock Weißes Pulsieren") + "\
    " + IoT_WebFormActionButton ("form13", "BRK-Clock Regenbogen wird weiß") + "\
    " + IoT_WebFormActionButton ("form14", "BRK-Clock Ambientebeleuchtung") + "\
    " + IoT_WebFormActionButton ("form16", "BRK-Clock Regenbogen") + "\
    " + IoT_WebFormActionButton ("form15", "BRK-Clock Regenbogenzyklus") + "\
    " + IoT_WebFormOpen("form08") + "\
    " + IoT_WebFormSubmitButton("BRK-Clock Laufschrift") + "\
    " + IoT_WebInput(" ", "lauf", cleanhtml(Lauf_Schrift), 40) + "<br>\
    " + IoT_WebFormClose() + "\
    <p> </p>\
    <p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form18", "Startseite anzeigen")   + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                          // HTTP-Seite aktualisieren
}
```

BRK-Matrix Clock (Deutsche und Englische Version) Listing

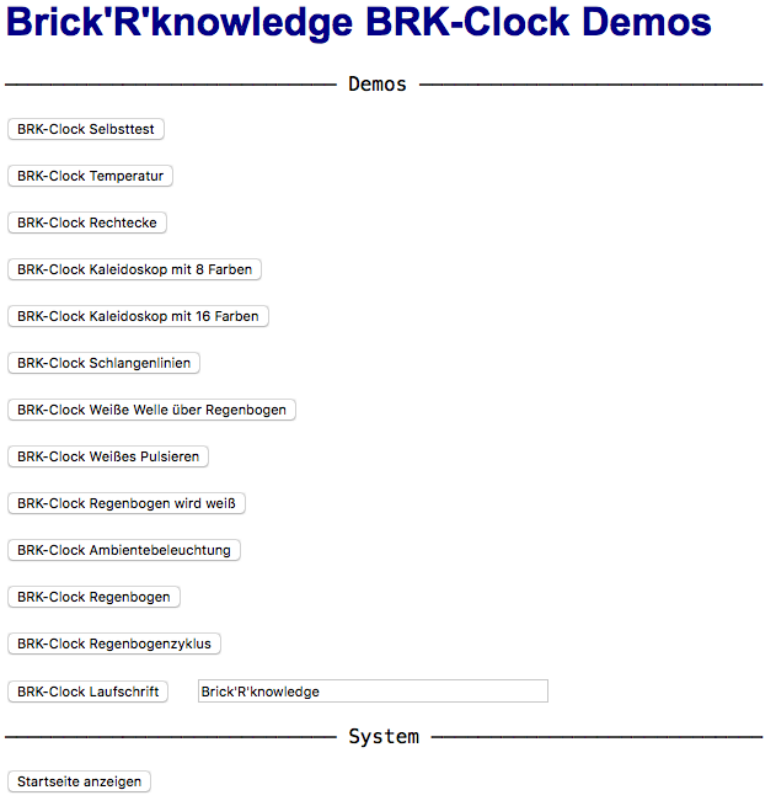
Das folgende Programm mit 655 Zeilen ist nicht Bestandteil des IoT-Handbuchs und beschaltet die 8 x 8 RGBW BRK-Matrix mit integriertem Microcontroller EPS8266. Dabei werden über das Webinterface des EPS8266 zahlreiche Funktionen für eine Wort-Uhr (Farbanpassung mit vordefinierten Grundfarben sowie frei eingebbarer RGBW-Farbe, Wecker, Sekunden- und Minutentöne usw.) sowie einige Demos für die 8 x 8 RGBW-Matrix (Temperaturanzeige, Farbdemos, Kaleidoskop, Laufschrift usw.) zur Verfügung gestellt. Weiterhin kann man den IoT-Brick im Webinterface neu starten und die ganze Schaltung incl. der 8 x 8 RGBW-Matrix ausschalten. Darüber hinaus werden im Webinterface noch einige Informationen angezeigt wie Datum, Uhrzeit, Temperatur, die IP-Adresse, unter der der IoT-Brick im lokalen Netzwerk verfügbar ist sowie die öffentliche IP-Adresse, unter der der genutzte Internetanschluss im Internet angemeldet ist.

Durch Drücken des Tasters kann man sich die aktuelle IP-Information als Laufschrift auf der Matrix ansehen.

Das Webinterface ist farblich so gestaltet, dass es die aktuell angezeigte Wortfarbe der BRK-Matrix annimmt und so optimiert, dass die Ladezeit unter eine Sekunde bleibt und auch mehrfache, parallele Browserzugriffe möglich sind, ohne dass der EPS8266 sich dadurch aufhängt oder neu startet.

Durch das Drücken des Tasters wird eine einmal gestartete Demo wieder zu beendet. Die Leitung GPIO 13 wird für die Ansteuerung der 8 x 8 RGBW-Matrix benötigt. Der GPIO 14 wird für die Ansteuerung des Temperatur-Sensor verwendet.

Durch Drücken auf den Button „Einstellungen“ gelangt man zu einer zweiten Webseite, auf der man die Netzwerkeinstellungen der BRK-Matrix Clock einsehen und verändern kann. An dieser Stelle kann man dem EPS8266 eine statische IP-Adresse zuweisen, diesen neu starten oder die ganze BRK-Matrix Clock ausschalten.



Brick'R'knowledge BRK-Clock System

Aktuelle Netzwerk-Einstellungen

IP-Addresse : 192.168.1.13

Gateway : 192.168.1.1

Subnetzmaske : 255.255.255.0

Öffentliche IP: 78.94.163.210

Netzwerk

IP-Adresse Gateway Subnetzmaske

Ändern 192.168.1.13 192.168.1.1 255.255.255.0 (dynamisch)

Dynamische IP-Adresse

Sprache & Taster

Temperatur-Offset +/-: 0,0 °C

- ○ Deutsche Wort-Uhr
- ◉ English Word-Clock

- ○ IP-Adresse anzeigen
- ◉ Temperatur anzeigen
- ○ Rechtecke
- ○ Kaleidoskop
- ○ Schlangenlinien
- ○ Weiße Welle über Regenbogen
- ○ Ambientebeleuchtung
- ○ Regenbogen
- ○ Regenbogenzyklus

Sprache & Taster-Einstellung ändern

Einstellungen

Standard Einstellungen wiederherstellen

Aktuelle Einstellungen dauerhaft speichern

System

Startseite anzeigen

BRK-Clock neu starten

BRK-Clock ausschalten

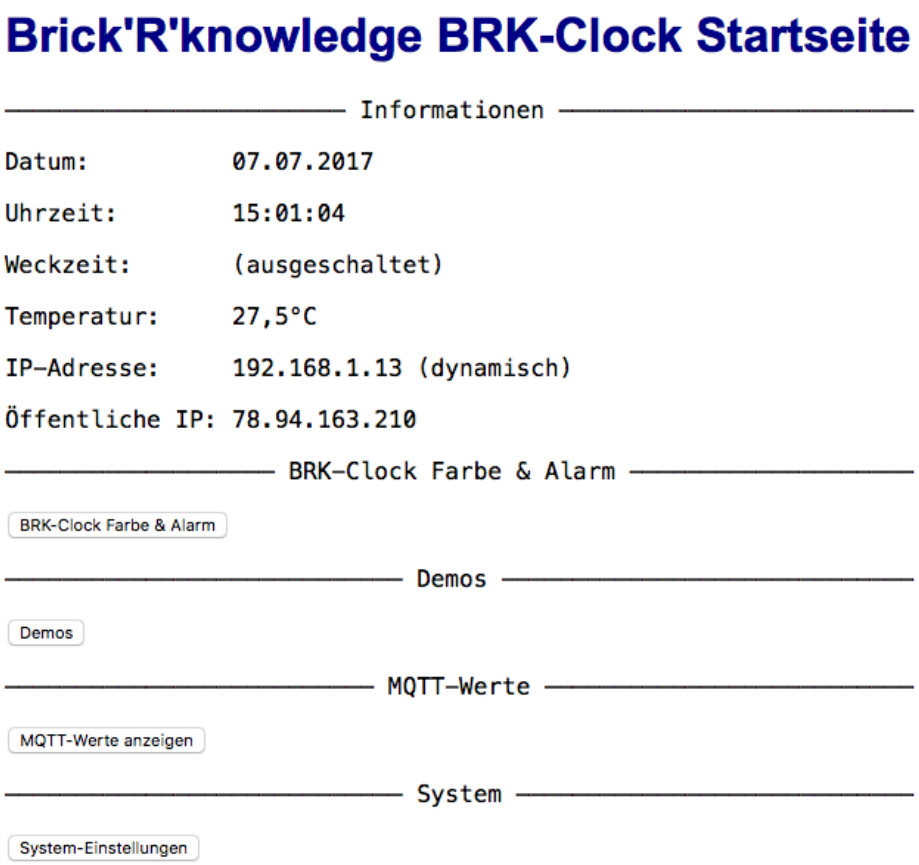
Zeitanzeige mit Papier-Schablone darübergelegt:

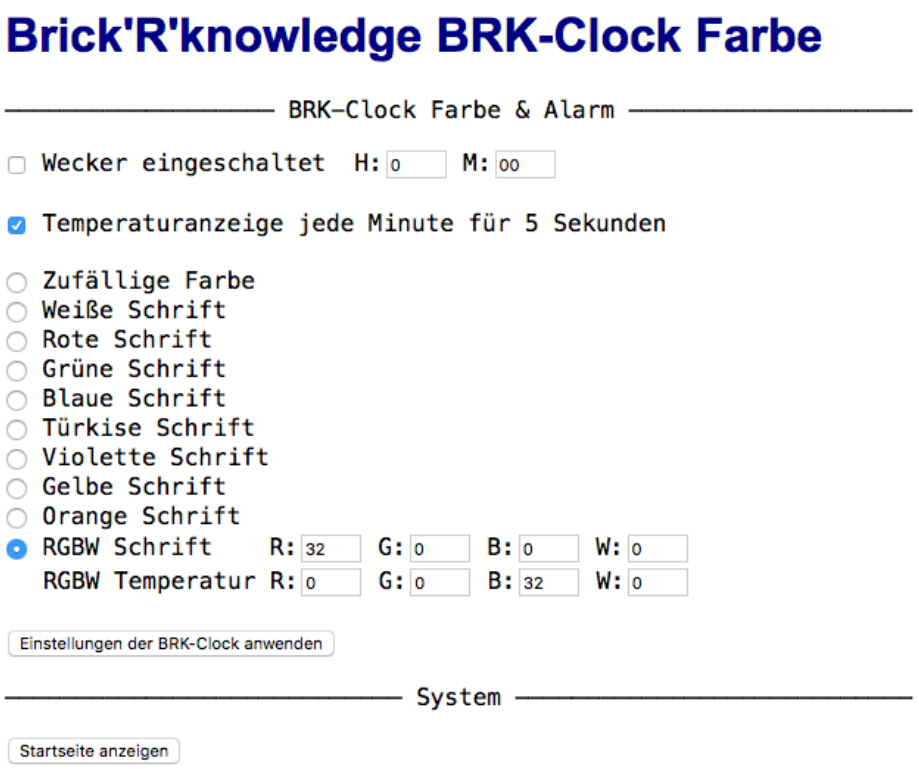
Deutsche Belegung der RGBW-Matrix

Englische Belegung der RGBW-Matrix

Aktuell wird eine deutsche und eine englische Wort-Belegung unterstützt. Die oben angezeigten Schablonen für die RGBW-Matrix 8 x 8 sind 25 x 25 cm groß und lassen sich im Original-Maßstab auf jeweils einem DIN A3 Blatt ausdrucken. In der deutschen Belegung werden die Buchstaben "RK" links unten nur während der Startsequenz benutzt und das P in der zweiten Zeile überhaupt nicht. In der englischen Belegung wird nur das V in der zweiten Zeile überhaupt nicht benutzt.

Die Web-Oberfläche der BRK-Clock besteht aus fünf Webseiten, von denen drei auf dieser Seite, die anderen zwei auf den vorherigen Seiten abgebildet sind. Auf jeder der Unterseiten befindet sich ein Button „Startseite anzeigen“, mit dem man zur Startseite zurückkehren kann.





Brick'R'knowledge BRK-Clock MQTT-Werte

Verbindungsstatus mit Server 'iot.allnet.de': Verbindung hergestellt

Temperaturen

Hamburg	= 19,7°C	Berlin	= 22,5°C
Frankfurt	= 25,4°C	Stuttgart	= 24,0°C
Hannover	= 20,8°C	München	= 25,7°C
Köln	= 21,8°C	Düsseldorf	= 21,7°C
Nürnberg	= 25,0°C	Dresden	= 24,0°C
Naumburg	= 23,7°C	Heidelberg	= 26,0°C
Bremen	= 21,0°C	Dortmund	= 21,0°C
Kiel	= 19,5°C	Leipzig	= 24,0°C
Regensburg	= 26,1°C	Rostock	= 20,0°C
Saarbrücken	= 24,7°C	Trier	= 21,2°C
Ulm	= 25,0°C	Würzburg	= 23,8°C
Oldenburg	= 20,4°C	Potsdam	= 22,5°C
Cottbus	= 25,1°C	Schwerin	= 20,6°C
Mainz	= 25,4°C	Wiesbaden	= 23,4°C

Aktien- und Währungskurse

DAX	= 12.344,50	Differenz	= -102,75 (-0,83%)
MDAX	= 24.699,27	Differenz	= -157,61 (-0,63%)
ESTX50	= 3.479,20	Differenz	= -20,29 (-0,58%)
Gold	= 1.247,10	Differenz	= +1,60 (+0,13%)
Öl	= 46,48	Differenz	= -0,44 (-0,94%)
EUR/USD	= 1,16	Differenz	= +0,00 (+0,09%)
BitCoin/USD	= 2.726,12	Differenz	= -51,42 (-1,85%)
Ethereum/USD	= 227,51	Differenz	= -7,29 (-3,10%)

Aktualisieren

Alle MQTT-Werte aktualisieren

System

Startseite anzeigen

```
//  Programm für eine Wort-Uhr für die BRK-Matrix Hardware
//  Nur mit einem EPS8266 Chip ohne den IoT-Brick
//
//  Unter Verwendung der IoT Brick Library ab Version 1.03


#define BRKclock_PIN 13                                          // Led Pin
#define DS18B20_PIN  14                                          // Datenleitung des DS18B20-Sensors am GPIO14 des EPS8266-Microcontrollers

#include <all_IoT.h>
#include <all_BRKclock.h>
#include <all_DS18B20.h>
#include <all_MQTT.h>


int    TemperatureVal  = 0;
String Temperature     = "";
String Humidity        = "";
String Lauf_Schrift    = "Brick'R'knowledge";
double tempOffset      = 0.0;

int    counter = 0;
int    m_alt = 0;
int    s_alt = 0;


void setup()
{
    t_TerminalInit();                                            // Terminal initialisieren
    Serial.print(t_TerminalClearScreen());
    IoT_Init(false);                                             // EPS8266 initialisieren (ohne OLED/ADC)
    BRKclock_Init();                                             // BRK-Clock initialisieren
    BRKclock_TestMatrix(10);                                     // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(2000, false);                               // 2 s warten, in der Zeit kann mit Taster die Konfig zurückgesetzt werden
    Serial.println("");
    Serial.println("");
    if (IoT_Keypress())                                          // Wenn Taster gedrückt wird, Konfiguration im EEPROM löschen
    {
        IoT_EEPROMclear();                                         // Alle Parameter aus dem EEPROM löschen
        IoT_EEPROMupdate();                                        // EEPROM aktualisieren
        Serial.print("BRK-Clock Parameter aus EEPROM gelöscht. ");
        Serial.println("BRK-Clock wird neu gestartet...");
        delay(100);                                                // 100 ms. warten
        BRKclock_Pixel.clear();                                    // Pixel löschen
        BRKclock_Pixel.show();                                     // Anzeige aktualisieren
        IoT_WaitNoKeypress();                                      // Warten bis der Taster losgelassen wurde
        t_Restart();                                               // Neustart durchführen
    }
    BRKclock_WLANautoConnect(true);
    IoT_NTPinit();                                               // NTP-Timeserver initialisieren
```

```
if (not(DS18B20_Init(false)))                                   // DS18B20 Temperatursensor initialisieren
{
    Serial.println("Temperature Sensor DS18B20 not found.");
}
if (BRKclock_ParameterLoad())                                   // Parameter-Block aus EEPROM laden
{
    Serial.println("BRK-Clock Parameter aus EEPROM geladen.");
    BRKclock_ParameterApply();                                    // Parameter-Block in die aktuelle Konfiguration übernehmen
}
else
{
    Serial.println("BRK-Clock Parameter im EEPROM nicht vorhanden.");
}

MQTT_Init("iot.allnet.de", 1883, "BrickRknowledge", "open", "Allnet"); // iot.allnet.de = IPval("212.18.29.166") ("open",  "Allnet")
MQTT_Connect("general/data/#");
MQTT_ConnectTrace = false;

IoT_WebServer.on("/", handleRoot);                              // Browser-Handler (siehe unten) für das Stammverzeichnis
IoT_WebServer.on("/form01", handleForm01);                      // Handler für die Eingabe der Namen in den Eingabefeldern
IoT_WebServer.on("/form02", handleForm02);                      // Handler für "IoT-Brick neu starten"
IoT_WebServer.on("/form03", handleForm03);                      // Handler für "IoT-Brick ausschalten"
IoT_WebServer.on("/form04", handleForm04);                      // Handler für "BRK-Clock Selbsttest"
IoT_WebServer.on("/form05", handleForm05);                      // Handler für "BRK-Clock Rechtecke"
IoT_WebServer.on("/form06", handleForm06);                      // Handler für "BRK-Clock Kaleidoskop mit 8 Farben"
IoT_WebServer.on("/form07", handleForm07);                      // Handler für "BRK-Clock Kaleidoskop mit 16 Farben"
IoT_WebServer.on("/form08", handleForm08);                      // Handler für "BRK-Clock Laufschrift"
IoT_WebServer.on("/form09", handleForm09);                      // Handler für "Temperatur anzeigen"
IoT_WebServer.on("/form10", handleForm10);                      // Handler für "Schlangenlinien"
IoT_WebServer.on("/form11", handleForm11);                      // Handler für "Weiße Welle über Regenbogen"
IoT_WebServer.on("/form12", handleForm12);                      // Handler für "Weißes Pulsieren"
IoT_WebServer.on("/form13", handleForm13);                      // Handler für "Regenbogen wird weiß"
IoT_WebServer.on("/form14", handleForm14);                      // Handler für "Ambientebeleuchtung"
IoT_WebServer.on("/form15", handleForm15);                      // Handler für "Regenbogenzyklus"
IoT_WebServer.on("/form16", handleForm16);                      // Handler für "Regenbogen"
IoT_WebServer.on("/form17", handleForm17);                      // Handler für "Einstellungen"
IoT_WebServer.on("/form18", handleForm18);                      // Handler für "Startseite anzeigen"
IoT_WebServer.on("/form19", handleForm19);                      // Handler für "Dynamische IP-Adresse"
IoT_WebServer.on("/form20", handleForm20);                      // Handler für "Taster-Einstellung ändern"
IoT_WebServer.on("/form21", handleForm21);                      // Handler für "Einstellungen speichern"
IoT_WebServer.on("/form22", handleForm22);                      // Handler für "Standard Einstellungen wiederherstellen"
IoT_WebServer.on("/form23", handleForm23);                      // Handler für "Dynamische IP-Adresse"
IoT_WebServer.on("/form24", handleForm24);                      // Handler für "Demos"
IoT_WebServer.on("/form25", handleForm25);                      // Handler für "MQTT-Werte"
IoT_WebServer.on("/form26", handleForm26);                      // Handler für "Farbeinstellung"
IoT_WebServer.begin();                                          // ab jetzt "horcht" der Server auf HTTP-Anfragen
Serial.println("HTTP server started");
IoT_WebServer.handleClient();                                   // Bediene die http Anfragen
BRKclock_SetClockRGBWColor(BRKclock_clockColorRed, BRKclock_clockColorGreen, BRKclock_clockColorBlue, BRKclock_clockColorWhite); // Schriftfarbe
Serial.println("");
```

```
}

void loop()
{
    IoT_Idle();
    IoT_WebServer.handleClient();                                   // Bediene die http Anfragen
    IoT_Idle();
    int h = hour();
    int m = minute();
    int s = second();
    bool NTPvalid = IoT_NTPvalid();

    if ((m_alt != m) && (NTPvalid))                                 // Minute hat sich geändert und NTP-Time wurde empfangen
    {
        if (BRKclock_colorMode == 0)                                  // Farbe jede Minute wechseln
        {
            BRKclock_SetClockRandomColor();                             // Zufällige Farbe erzeugen
        }
        m_alt = m;
        s_alt = s;
        if (BRKclock_temperatureMode == 1)
        {
            BRKclock_PrintTemperature(TemperatureVal, 5000);            // Temperatur anzeigen & 5 Sekunden warten
        }
        BRKclock_International(h, m);
    }
    else if ((s_alt != s) && (NTPvalid))                            // Sekunde hat sich geändert und NTP-Time wurde empfangen
    {
        s_alt = s;
    }
    else
    {
        delay(100);
        IoT_Idle();
    }

    if (IoT_Keypress())                                             // Wenn Taster gedrückt wird, entsprechende Aktion ausführen
    {
        if (BRKclock_buttonMode == 0)                                 // IP-Adresse als Laufschrift anzeigen
        {
            String ip = "IP: " + IoT_WLANaddress(true) + "  GW: " + IoT_WLANgateway(true) + "  NT: " + IoT_WLANsubnet(true);
            if (counter % 10 == 0) // Infos seriell nicht zu oft ausgeben (ca. jede Sekunden, nur wenn der Button gedrückt wird)
            {
                Serial.println(IoT_NTPdatetime() + " (" + IoT_NTPdaylightSavingTime(true) + ")");
                Serial.print("WiFi is ");
                Serial.print(IoT_WLANconnected() ? "connected" : "not connected");
                Serial.println(". Uptime: " + IoT_WLANuptime(true) + " seit " + IoT_WLANstartDatetime());
                Serial.println(ip);                                       // IP-Information
            }
            IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
```

```
    BRKclock_Print(ip, 250);                                  // Laufschrift-Text mit IP-Information, 250 ms. Pause
}
else if (BRKclock_buttonMode == 1)                           // Temperatur anzeigen
{
    BRKclock_PrintTemperature(TemperatureVal, 0);             // Temperatur anzeigen & 10 Sekunden warten
}
else if (BRKclock_buttonMode == 2)                           // Rechtecke starten
{
    BRKclock_Pixel.clear();                                   // Pixel löschen
    BRKclock_Pixel.show();                                    // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                     // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_Rectangles();                                  // Rechtecke-Demo
    }
}
else if (BRKclock_buttonMode == 3)                           // Kaleidoskop mit 16 Farben starten
{
    BRKclock_Pixel.clear();                                   // Pixel löschen
    BRKclock_Pixel.show();                                    // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                     // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_Kaleidoscope(8, true);                         // Kaleidoskop mit 16 Farben
    }
}
else if (BRKclock_buttonMode == 4)                           // Schlangenlinien starten
{
    BRKclock_Pixel.clear();                                   // Pixel löschen
    BRKclock_Pixel.show();                                    // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                     // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_SnakeLines(50);                                // Schlangenlinien
    }
}
else if (BRKclock_buttonMode == 5)                           // Weiße Welle über Regenbogen starten
{
    BRKclock_Pixel.clear();                                   // Pixel löschen
    BRKclock_Pixel.show();                                    // Anzeige aktualisieren
    IoT_WaitNoKeypress();                                     // Warten bis der Taster losgelassen wurde
    while (not(IoT_Keypress()))
    {
        BRKclock_WhiteOverRainbow(20, 75, 5);                   // Weiße Welle über Regenbogen
    }
}
else if (BRKclock_buttonMode == 6)                           // Ambientebeleuchtung starten
{
    BRKclock_Pixel.clear();                                   // Pixel löschen
    BRKclock_Pixel.show();                                    // Anzeige aktualisieren
```

```
        IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
        while (not(IoT_Keypress()))
        {
            BRKclock_AmbienceColors(15000);                       // Ambientebeleuchtung, 15 Sekunden Pause
        }
    }
    else if (BRKclock_buttonMode == 7)                          // Regenbogen starten
    {
        BRKclock_Pixel.clear();                                 // Pixel löschen
        BRKclock_Pixel.show();                                  // Anzeige aktualisieren
        IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
        while (not(IoT_Keypress()))
        {
            BRKclock_Rainbow(50);                                 // Regenbogen
        }
    }
    else if (BRKclock_buttonMode == 8)                          // Regenbogenzyklus starten
    {
        BRKclock_Pixel.clear();                                 // Pixel löschen
        BRKclock_Pixel.show();                                  // Anzeige aktualisieren
        IoT_WaitNoKeypress();                                   // Warten bis der Taster losgelassen wurde
        while (not(IoT_Keypress()))
        {
            BRKclock_RainbowCycle(50);                            // Regenbogenzyklus
        }
    }
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}
else
{
    if ((counter % 25 == 0) && not(NTPvalid))                   // NTP Event alle 2 Sekunden auslesen wenn noch nicht empfangen
    {
        if (IoT_NTPeventAvail)                                  // Zeitevent-Infos im Terminal ausgeben
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }
    else if ((counter % 7500 == 0) && not(NTPvalid))            // NTP Event alle 10 Minuten auslesen wenn bereits empfangen
    {
        if (IoT_NTPeventAvail)                                  // Zeitevent-Infos im Terminal ausgeben
        {
            IoT_NTPprintEvent(IoT_NTPcurrentEvent);
            IoT_NTPeventAvail = false;
        }
    }
    else
    {
        if (counter % 100 == 1)                                 // Temperatur alle 5 Sekunden auslesen
```

```
            {
                Serial.print("Temperatur lesen... ");
                double t = DS18B20_Temperature();
                if (not(isnan(t)))
                {
                    Serial.print("Erfolgreich. ");
                    TemperatureVal = int(t + tempOffset);
                    if (t == -127)
                    {
                        Temperature = "(Sensor nicht gefunden)";
                    }
                    else
                    {
                        Serial.print("Temperatur = " + String(t));
                        Temperature = strrealform(t + tempOffset, 32, 1, 1, true, false) + "°C"; // 1 Nachkommastelle, auch Null als Nachkommastelle
zulassen
                    }
                }
                Serial.println("");
            }
            else if (counter % 100 == 50)
            {
                int maxMQTT = 100;
                while ((MQTT_EventAvail) && (maxMQTT > 0))              // Maximal 100 MQTT Events auf einmal abfragen
                {
                    MQTT_GetNextEvent();
                    if (not(MQTT_AllnetParseEventCurrent())) // Unbekannten Event ausgeben
                    {
                        Serial.print("Time: "      + cleanasc(String(MQTT_EventCurrentTime)));
                        Serial.println(", Topic: " + cleanasc(String(MQTT_EventCurrentTopic)));
                        Serial.println("Message: " + String(MQTT_EventCurrentMessage));
                    }
                    bool success = MQTT_HandleClient();                   // Verbindung aufrecht erhalten, ggf. neu aufbauen
                    maxMQTT -= 1;
                }
            }
            IoT_Idle();
            IoT_WebServer.handleClient();                               // Bediene die http Anfragen
            bool success = MQTT_HandleClient();                         // Verbindung aufrecht erhalten, ggf. neu aufbauen
            Serial.print("*");                                          // loop() wurde einmal durchlaufen
        }
    }

    IoT_Idle();

    if (BRKclock_alarmMode == 1)                                    // Wecker prüfen
    {
        if ((h == BRKclock_alarmHour) && (m == BRKclock_alarmMinute)) // Weckzeit erreicht?
        {
            byte i = 0;
```

```
            while ((i < 60) && not(IoT_Keypress()))                  // 60 Sekunden Alarm oder bis zum Drücken des Tasters
            {
                if (not(IoT_Keypress()))
                {
                    BRKclock_Pixel.clear();                              // Pixel löschen
                    BRKclock_Pixel.show();                               // Anzeige aktualisieren
                    IoT_Idle();
                    IoT_WaitKeypress(500, false);                        // 500 ms. warten, Wartezeit wird durch den Button unterbrochen
                    BRKclock_International(h, m);
                    if (not(IoT_Keypress()))
                    {
                        long t = millis();
                        IoT_WebServer.handleClient();                      // Bediene die http Anfragen
                        IoT_Idle();
                        t = 500 - (millis() - t);
                        if (t > 0)
                        {
                            IoT_WaitKeypress(t, false);                      // ≤500 ms. warten, Wartezeit wird durch den Button unterbrochen
                        }
                    }
                }
                i++;
            }
            BRKclock_alarmMode = 2;                                  // Alarm beendet, verhindert dass der Alarm sofort wieder beginnt
        }
    }
    else if (BRKclock_alarmMode == 2)                              // Alarm beendet
    {
        if (not((h == BRKclock_alarmHour) && (m == BRKclock_alarmMinute)))  // Weckzeit vorbei?
        {
            BRKclock_alarmMode = 1;                                  // Normalen Alarm-Modus wiederherstellen
        }
    }

    IoT_Idle();
    counter++;
}

// Mit der Funktion handleRoot() wird die Website ausgeliefert sobald eine Anfrage von einem Browser eintrifft

void handleRoot ()
{
    IoT_Idle();
    String time_web = IoT_NTPtime();
    String date_web = IoT_NTPdate();
    String alarm_web = "";
    if (BRKclock_alarmMode == 1)                                   // Wecker eingeschaltet
    {
        alarm_web = strform(BRKclock_alarmHour, 48, 2, false) + ":" + strform(BRKclock_alarmMinute, 48, 2, false) + ":00";
    }
```

```
else
{
    alarm_web = "(ausgeschaltet)";
}
String ipaddr_web  = IoT_WLANaddress(false);
String title_web = "Brick'R'knowledge BRK-Clock Startseite";
int sec = millis() / 1000;
int min = sec / 60;
int hr = min / 60;
String ColorHexVal = "";
String staticInfo = " (dynamisch)";
if (BRKclock_staticIP == 1)
{
    staticInfo = " (statisch)";
}
switch (BRKclock_colorMode) // Farbe für Web-Interface
{
    case 0: // Random
        ColorHexVal = "#000000";
        break;
    case 1: // Weiß
        ColorHexVal = "#000000";
        break;
    case 2: // Rot
        ColorHexVal = "#FF0000";
        break;
    case 3: // Grün
        ColorHexVal = "#00FF00";
        break;
    case 4: // Blau
        ColorHexVal = "#0000FF";
        break;
    case 5: // Türkis
        ColorHexVal = "#00FFFF";
        break;
    case 6: // Violett
        ColorHexVal = "#FF00FF";
        break;
    case 7: // Gelb
        ColorHexVal = "#FFFF00";
        break;
    case 8: // Orange
        ColorHexVal = "#FF8000";
        break;
    default: // Unbekannte Auswahl
        ColorHexVal = "#000000";
        break;
}
String content =                                                // Die HTML-Seite in lesbarer Formatierung,
// darf auch Variablen enthalten
```

```
    "<html>\
    <head>\
    <title>" + cleanhtml(title_web) + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p>" + cleanhtml(fillcenter(" Informationen ", "-", 64)) + "</p>\
    <p> </p>\
    <p>" + cleanhtml("Datum:          " + date_web)    + "</p>\
    <p>" + cleanhtml("Uhrzeit:        " + time_web)    + "</p>\
    <p>" + cleanhtml("Weckzeit:       " + alarm_web)   + "</p>\
    <p>" + cleanhtml("Temperatur:     " + Temperature) + "</p>\
    <p>" + cleanhtml("IP-Adresse:     " + ipaddr_web)  + staticInfo + "</p>\
    <p>" + cleanhtml("Öffentliche IP: ") + IoT_WebClientIP() + "</p>\
    <p>" + cleanhtml(fillcenter(" BRK-Clock Farbe & Alarm ", "-", 64)) + "</p>\
    <p> </p>";
    content +=
    IoT_WebFormActionButton ("form26", "BRK-Clock Farbe & Alarm") + "\
    <p> </p>\
    <p>" + cleanhtml(fillcenter(" Demos ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form24", "Demos") + "\
    <p>" + cleanhtml(fillcenter(" MQTT-Werte ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form25", "MQTT-Werte anzeigen") + "\
    <p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form17", "System-Einstellungen") + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                                    // HTTP-Seite aktualisieren
    content = "";
}

void handleForm01 ()
{
    Serial.println("Button wurde betaetigt: Form01");
    // IoT_WebPrintAllFields();                                                 // Alle gefundenen Felder auf dem Terminal ausgeben
    IoT_Idle();

    BRKclock_temperatureMode = boolval(IoT_WebTestField("display"));  // Testen, ob die Checkbox Display angekreuzt wurde
    BRKclock_alarmMode   = boolval(IoT_WebTestField("alarm"));        // Testen, ob die Checkbox Wecker angekreuzt wurde
    BRKclock_alarmHour       = val(IoT_WebGetField("wh"));
```

```
BRKclock_alarmMinute     = val(IoT_WebGetField("wm"));

if (IoT_WebTestField("COLORgroup"))                                 // Testen, ob Felder mit dem Namen "COLORgroup" existieren
{
    String selection = IoT_WebGetField("COLORgroup");
    if (selection == "random")                                        // Schwarzer Text im Web, zufällige Farbe auf der Uhr
    {
        BRKclock_colorMode = 0;                                         // Ab der nächsten Minute die Farbe ändern
        BRKclock_SetClockRandomColor();                                 // Zufällige Farbe erzeugen
        BRKclock_SetTemperatureRGBWColor(BRKclock_clockColorRed, BRKclock_clockColorGreen, BRKclock_clockColorBlue, BRKclock_clockColorWhite);
    }
    else if (selection == "black")                                    // Schwarzer Text im Web, weiße Schrift auf der Uhr
    {
        BRKclock_colorMode = 1;
        BRKclock_SetClockRGBWColor(0xFF, 0xFF, 0xFF, 0);
        BRKclock_SetTemperatureRGBWColor(0xFF, 0xFF, 0xFF, 0);
    }
    else if (selection == "red")                                      // Rot
    {
        BRKclock_colorMode = 2;
        BRKclock_SetClockRGBWColor(0xFF, 0x00, 0x00, 0);
        BRKclock_SetTemperatureRGBWColor(0xFF, 0x00, 0x00, 0);
    }
    else if (selection == "green")                                    // Grün
    {
        BRKclock_colorMode = 3;
        BRKclock_SetClockRGBWColor(0x00, 0xFF, 0x00, 0);
        BRKclock_SetTemperatureRGBWColor(0x00, 0xFF, 0x00, 0);
    }
    else if (selection == "blue")                                     // Blau
    {
        BRKclock_colorMode = 4;
        BRKclock_SetClockRGBWColor(0x00, 0x00, 0xFF, 0);
        BRKclock_SetTemperatureRGBWColor(0x00, 0x00, 0xFF, 0);
    }
    else if (selection == "cyan")                                     // Türkis
    {
        BRKclock_colorMode = 5;
        BRKclock_SetClockRGBWColor(0x00, 0xFF, 0xFF, 0);
        BRKclock_SetTemperatureRGBWColor(0x00, 0xFF, 0xFF, 0);
    }
    else if (selection == "magenta")                                  // Violett
    {
        BRKclock_colorMode = 6;
        BRKclock_SetClockRGBWColor(0xFF, 0x00, 0xFF, 0);
        BRKclock_SetTemperatureRGBWColor(0xFF, 0x00, 0xFF, 0);
    }
    else if (selection == "yellow")                                   // Gelb
    {
        BRKclock_colorMode = 7;
```

```
            BRKclock_SetClockRGBWColor(0xFF, 0xFF, 0x00, 0);
            BRKclock_SetTemperatureRGBWColor(0xFF, 0xFF, 0x00, 0);
        }
        else if (selection == "orange")                                 // Orange
        {
            BRKclock_colorMode = 8;
            BRKclock_SetClockRGBWColor(0xFF, 0x80, 0x00, 0);
            BRKclock_SetTemperatureRGBWColor(0xFF, 0x80, 0x00, 0);
        }
        else if (selection == "user")                                   // Benutzerdefinierte Farbe
        {
            BRKclock_colorMode = 9;
            BRKclock_SetClockRGBWColor      (val(IoT_WebGetField("ur")),
                                             val(IoT_WebGetField("ug")),
                                             val(IoT_WebGetField("ub")),
                                             val(IoT_WebGetField("uw")));
            BRKclock_SetTemperatureRGBWColor(val(IoT_WebGetField("tr")),
                                             val(IoT_WebGetField("tg")),
                                             val(IoT_WebGetField("tb")),
                                             val(IoT_WebGetField("tw")));
        }
        BRKclock_International(hour(), minute());
    }
    IoT_WaitNoKeypress();                                               // Warten bis der Taster losgelassen wurde
    handleForm26();                                                     // Farb-Einstellungen anzeigen
}

void handleForm02 ()                                                    // Neustart
{
    Serial.println("Button wurde betaetigt: Form02");
    handleForm17();                                                     // Einstellungen anzeigen
    IoT_Idle();
    BRKclock_Clear(true);                                               // Pixel löschen & Anzeige aktualisieren
    IoT_WaitNoKeypress();                                               // Warten bis der Taster losgelassen wurde
    t_Restart();
}

void handleForm03 ()                                                    // Ausschalten
{
    Serial.println("Button wurde betaetigt: Form03");
    handleRoot();                                                       // Wieder auf die Hauptseite zurück schalten
    BRKclock_Clear(true);                                               // Pixel löschen & Anzeige aktualisieren
    IoT_ShutDown();
}

void handleForm04 ()                                                    // Selbsttest
{
    Serial.println("Button wurde betaetigt: Form04");
    handleForm24();                                                     // Demo-Seite anzeigen
    BRKclock_Clear(false);                                              // Pixel löschen & Anzeige aktualisieren
```

```
    BRKclock_TestMatrix(50);                                    // Alle LEDs mit 50 ms. Verzögerung einschalten
    IoT_WaitKeypress(0, false);                                 // Auf Taster warten
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm05 ()                                            // Rechtecke Demo
{
    Serial.println("Button wurde betaetigt: Form05");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Rectangles();                                      // Rechtecke-Demo
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm06 ()                                            // Kaleidoskop mit 8 Farben
{
    Serial.println("Button wurde betaetigt: Form06");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Kaleidoscope(8, false);
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm07 ()                                            // Kaleidoskop mit 16 Farben
{
    Serial.println("Button wurde betaetigt: Form07");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Kaleidoscope(8, true);
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm08 ()                                            // Laufschrift
{
    Serial.println("Button wurde betaetigt: Form08");
    Lauf_Schrift = replace(IoT_WebGetField("lauf"), chr(160), " ");   // Geschütztes Leerzeichen in Leerzeichen ersetzen
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Print(Lauf_Schrift, 100);                          // Laufschrift-Text von der Webseite laden
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm09 ()                                            // Temperatur anzeigen
{
    Serial.println("Button wurde betaetigt: Form09");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_PrintTemperature(TemperatureVal, 10000);           // Temperatur anzeigen & 10 Sekunden warten
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
```

```
}

void handleForm10 ()                                    // Schlangenlinien
{
    Serial.println("Button wurde betaetigt: Form10");
    handleForm24();                                       // Demo-Seite anzeigen
    BRKclock_Clear(false);                                // Pixel löschen & Anzeige aktualisieren
    BRKclock_SnakeLines(50);                              // Schlangenlinien
    IoT_WaitNoKeypress();                                 // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());             // Normale Zeitanzeige einschalten
}

void handleForm11 ()                                    // Weiße Welle über Regenbogen
{
    Serial.println("Button wurde betaetigt: Form11");
    handleForm24();                                       // Demo-Seite anzeigen
    BRKclock_Clear(false);                                // Pixel löschen & Anzeige aktualisieren
    BRKclock_WhiteOverRainbow(20, 75, 5);                 // Weiße Welle über Regenbogen
    IoT_WaitNoKeypress();                                 // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());             // Normale Zeitanzeige einschalten
}

void handleForm12 ()                                    // Weißes Pulsieren
{
    Serial.println("Button wurde betaetigt: Form12");
    handleForm24();                                       // Demo-Seite anzeigen
    BRKclock_Clear(false);                                // Pixel löschen & Anzeige aktualisieren
    BRKclock_PulseWhite(5, 4);
    IoT_WaitNoKeypress();                                 // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());             // Normale Zeitanzeige einschalten
}

void handleForm13 ()                                    // Regenbogen wird weiß
{
    Serial.println("Button wurde betaetigt: Form13");
    handleForm24();                                       // Demo-Seite anzeigen
    BRKclock_Clear(false);                                // Pixel löschen & Anzeige aktualisieren
    BRKclock_RainbowFade2White(3,3,1);
    IoT_WaitNoKeypress();                                 // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());             // Normale Zeitanzeige einschalten
}

void handleForm14 ()                                    // Ambientebeleuchtung
{
    Serial.println("Button wurde betaetigt: Form14");
    handleForm24();                                       // Demo-Seite anzeigen
```

```
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_AmbienceColors(1000);                              // Ambientebeleuchtung
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm15 ()                                          // Regenbogenzyklus
{
    Serial.println("Button wurde betaetigt: Form15");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_RainbowCycle(50);                                  // Regenbogenzyklus
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm16 ()                                          // Regenbogen
{
    Serial.println("Button wurde betaetigt: Form16");
    handleForm24();                                             // Demo-Seite anzeigen
    BRKclock_Clear(false);                                      // Pixel löschen & Anzeige aktualisieren
    BRKclock_Rainbow(50);                                       // Regenbogen
    IoT_WaitNoKeypress();                                       // Warten bis der Taster losgelassen wurde
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
}

void handleForm17 ()
{
    Serial.println("Button wurde betaetigt: Form17");
    IoT_Idle();
    String ColorHexVal = "#000000";
    String ipaddr_web  = IoT_WLANaddress(false);
    String ipaddr_gateway  = IoT_WLANgateway(false);
    String ipaddr_subnet  = IoT_WLANsubnet(false);
    String title_web = "Brick'R'knowledge BRK-Clock System";
    String staticInfo = " (dynamisch)";
    if (BRKclock_staticIP == 1)
    {
        staticInfo = " (statisch)";
    }
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String content =                                            // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
    "<html>\
    <head>\
```

```
<title>" + title_web + "</title>\
<style>\
body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
</style>\
<style type='text/css'>\
h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
</style>\
</head>\
<body>\
<h1>" + title_web + "</h1>\
<p>" + cleanhtml(fillcenter(" Aktuelle Netzwerk-Einstellungen ", "-", 64)) + "</p>\
<p> </p>";
content +=
"<p>" + cleanhtml("IP-Addresse   : " + ipaddr_web)      + "</p>\
<p>" + cleanhtml("Gateway       : " + ipaddr_gateway)  + "</p>\
<p>" + cleanhtml("Subnetzmaske  : " + ipaddr_subnet)   + "</p>\
<p> </p>\
<p>" + cleanhtml("Öffentliche IP: ") + IoT_WebClientIP() + "</p>\
<p>" + cleanhtml(fillcenter(" Netzwerk ", "-", 64)) + "</p>\
<p> </p>";
content +=
IoT_WebFormOpen("form19") +
"<p>" + cleanhtml("         IP-Adresse     Gateway     Subnetzmaske") + "</p>\
" + IoT_WebFormSubmitButton("Ändern") + "\
" + IoT_WebInput(" ", "ip01", ipaddr_web, 15) + "\
" + IoT_WebInput(" ", "ip02", ipaddr_gateway, 15) + "\
" + IoT_WebInput(" ", "ip03", ipaddr_subnet, 15) + staticInfo + "\
" + IoT_WebFormClose() + "\
" + IoT_WebFormActionButton ("form23", "Dynamische IP-Adresse") + "\
<p> </p>";
content +=
"<p>" + cleanhtml(fillcenter(" Sprache & Taster ", "-", 64)) + "</p>\
<p> </p>\
" + IoT_WebFormOpen("form20") + "\
" + IoT_WebInput("Temperatur-Offset +/-: ", "tempOffset", strrealform(tempOffset, 32, 1, 1, true, false), 5) + cleanhtml(" °C") + "<br>\
<p> </p>\
" + IoT_WebRadioButton(" Deutsche Wort-Uhr",            0, "LANGgroup",   "DE",  BRKclock_language)   + "<br>\
" + IoT_WebRadioButton(" English Word-Clock",          1, "LANGgroup",   "EN",  BRKclock_language)   + "<br>\
<p> </p>";
content +=
IoT_WebRadioButton(" IP-Adresse anzeigen",          0, "BTgroup", "butnIP",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Temperatur anzeigen",          1, "BTgroup", "butnTP",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Rechtecke",                    2, "BTgroup", "butn02",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Kaleidoskop",                  3, "BTgroup", "butn03",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Schlangenlinien",              4, "BTgroup", "butn04",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Weiße Welle über Regenbogen",  5, "BTgroup", "butn05",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Ambientebeleuchtung",          6, "BTgroup", "butn06",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Regenbogen",                   7, "BTgroup", "butn07",  BRKclock_buttonMode) + "<br>\
" + IoT_WebRadioButton(" Regenbogenzyklus",             8, "BTgroup", "butn08",  BRKclock_buttonMode) + "<br>\
<p> </p>";
```

```
    content +=
    IoT_WebFormSubmitButton("Sprache & Taster-Einstellung ändern") + "<br>\
    " + IoT_WebFormClose() + "\
    <p> </p>";
    content +=
    "<p>" + cleanhtml(fillcenter(" Einstellungen ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form22", "Standard Einstellungen wiederherstellen") + "\
    " + IoT_WebFormActionButton ("form21", "Aktuelle Einstellungen dauerhaft speichern") + "\
    <p> </p>\
    <p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form18", "Startseite anzeigen")   + "\
    " + IoT_WebFormActionButton ("form02", "BRK-Clock neu starten") + "\
    " + IoT_WebFormActionButton ("form03", "BRK-Clock ausschalten") + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                                    // HTTP-Seite aktualisieren
    content = "";
}

void handleForm18 ()                                                        // Startseite anzeigen
{
    Serial.println("Button wurde betaetigt: Form18");
    handleRoot();                                                             // Wieder auf die Hauptseite zurück schalten
    BRKclock_Pixel.setBrightness(255);
    BRKclock_International(hour(), minute());                                 // Normale Zeitanzeige einschalten
}

void handleForm19 ()                                                        // Statische IP-Adresse
{
    Serial.println("Button wurde betaetigt: Form19");
    BRKclock_staticIP = 1;
    IPAddress ip1 = IPval(IoT_WebGetField("ip01"));                           // Statische IP-Adresse
    IPAddress ip2 = IPval(IoT_WebGetField("ip02"));                           // Gateway/Router IP-Adresse
    IPAddress ip3 = IPval(IoT_WebGetField("ip03"));                           // Subnet IP-Adresse
    WiFi.config(ip1, ip2, ip3);                                               // Parameter: IP, Gateway, Subnet, DNS
    handleForm17();                                                           // Einstellungen anzeigen
}

void handleForm20 ()
{
    Serial.println("Button wurde betaetigt: Form20");
    if (IoT_WebTestField("BTgroup"))                                          // Testen, ob Felder mit dem Namen "BUTTONgroup" existieren
    {
        String selection = IoT_WebGetField("BTgroup");
        if (selection == "butnIP")
        {
            BRKclock_buttonMode = 0;                                              // IP-Adresse anzeigen
```

```
    }
    else if (selection == "butnTP")
    {
        BRKclock_buttonMode = 1;                                    // Temperatur anzeigen
    }
    else if (selection == "butn02")
    {
        BRKclock_buttonMode = 2;                                    // Rechtecke
    }
    else if (selection == "butn03")
    {
        BRKclock_buttonMode = 3;                                    // Kaleidoskop
    }
    else if (selection == "butn04")
    {
        BRKclock_buttonMode = 4;                                    // Schlangenlinien
    }
    else if (selection == "butn05")
    {
        BRKclock_buttonMode = 5;                                    // Weiße Welle über Regenbogen
    }
    else if (selection == "butn06")
    {
        BRKclock_buttonMode = 6;                                    // Ambientebeleuchtung
    }
    else if (selection == "butn07")
    {
        BRKclock_buttonMode = 7;                                   // Regenbogen
    }
    else if (selection == "butn08")
    {
        BRKclock_buttonMode = 8;                                    // Regenbogenzyklus
    }
}
if (IoT_WebTestField("LANGgroup"))                                 // Testen, ob Felder mit dem Namen "LANGgroup" existieren
{
    String selection = IoT_WebGetField("LANGgroup");
    if (selection == "DE")
    {
        BRKclock_language = 0;                                      // Deutsche Wort-Uhr
        BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
    }
    else if (selection == "EN")
    {
        BRKclock_language = 1;                                      // Englische Wort-Uhr
        BRKclock_International(hour(), minute());                   // Normale Zeitanzeige einschalten
    }
}
tempOffset = realval(IoT_WebGetField("tempOffset"));               // Temperatur-Offset auslesen
IoT_WaitNoKeypress();                                              // Warten bis der Taster losgelassen wurde
```

```
    handleForm17();                                             // Einstellungen anzeigen
}

void handleForm21 ()                                           // Einstellungen speichern (in das EEPROM)
{
    Serial.println("Button wurde betaetigt: Form21");
    BRKclock_ParameterCreate();                                 // Aktuelle Konfiguration in den Parameter-Block schreiben
    BRKclock_ParameterSave();                                   // Parameter-Block in das EEPROM sichern
    IoT_EEPROMupdate();                                         // EEPROM aktualisieren
    handleForm17();                                             // Einstellungen anzeigen
}

void handleForm22 ()                                           // Standard Einstellungen wiederherstellen
{
    Serial.println("Button wurde betaetigt: Form22");
    BRKclock_ParameterNew();                                    // Aktuelle Konfiguration in den Parameter-Block schreiben
    BRKclock_ParameterApply();                                  // Parameter-Block in die aktuelle Konfiguration übernehmen
    handleForm17();                                             // Einstellungen anzeigen
}

void handleForm23 ()                                           // Dynamische IP-Adresse
{
    Serial.println("Button wurde betaetigt: Form23");
    BRKclock_staticIP = 0;
    IPAddress ip1 = IPAddress(IoT_dynamicIP_address0, IoT_dynamicIP_address1, IoT_dynamicIP_address2, IoT_dynamicIP_address3);
    IPAddress ip2 = IPAddress(IoT_dynamicIP_gateway0, IoT_dynamicIP_gateway1, IoT_dynamicIP_gateway2, IoT_dynamicIP_gateway3);
    IPAddress ip3 = IPAddress(IoT_dynamicIP_subnet0,  IoT_dynamicIP_subnet1,  IoT_dynamicIP_subnet2,  IoT_dynamicIP_subnet3);
    WiFi.config(ip1, ip2, ip3);                                 // Parameter: IP, Gateway, Subnet, DNS
    handleForm17();                                             // Einstellungen anzeigen
}

void handleForm24 ()
{
    Serial.println("Button wurde betaetigt: Form24");
    IoT_Idle();
    String ColorHexVal = "#000000";
    String ipaddr_web  = IoT_WLANaddress(false);
    String ipaddr_gateway  = IoT_WLANgateway(false);
    String ipaddr_subnet  = IoT_WLANsubnet(false);
    String title_web = "Brick'R'knowledge BRK-Clock Demos";
    String staticInfo = " (dynamisch)";
    if (BRKclock_staticIP == 1)
    {
        staticInfo = " (statisch)";
    }
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    String content =                                            // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
```

```
    "<html>\
    <head>\
    <title>" + title_web + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p>" + cleanhtml(fillcenter(" Demos ", "-", 64)) + "</p>\
    <p> </p>";
    content +=
    IoT_WebFormActionButton ("form04", "BRK-Clock Selbsttest")
    + IoT_WebFormActionButton ("form09", "BRK-Clock Temperatur")
    + IoT_WebFormActionButton ("form05", "BRK-Clock Rechtecke")
    + IoT_WebFormActionButton ("form06", "BRK-Clock Kaleidoskop mit 8 Farben")
    + IoT_WebFormActionButton ("form07", "BRK-Clock Kaleidoskop mit 16 Farben")
    + IoT_WebFormActionButton ("form10", "BRK-Clock Schlangenlinien");
    content +=
    IoT_WebFormActionButton ("form11", "BRK-Clock Weiße Welle über Regenbogen")
    + IoT_WebFormActionButton ("form12", "BRK-Clock Weißes Pulsieren")
    + IoT_WebFormActionButton ("form13", "BRK-Clock Regenbogen wird weiß")
    + IoT_WebFormActionButton ("form14", "BRK-Clock Ambientebeleuchtung")
    + IoT_WebFormActionButton ("form16", "BRK-Clock Regenbogen")
    + IoT_WebFormActionButton ("form15", "BRK-Clock Regenbogenzyklus");
    content +=
    IoT_WebFormOpen("form08")
    + IoT_WebFormSubmitButton("BRK-Clock Laufschrift")
    + IoT_WebInput(" ", "lauf", cleanhtml(Lauf_Schrift), 40) + "<br>\
    " + IoT_WebFormClose() + "\
    <p> </p>";
    content +=
    "<p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form18", "Startseite anzeigen")   + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                                    // HTTP-Seite aktualisieren
    content = "";
}

void handleForm25 ()
{
    Serial.println("Button wurde betaetigt: Form25");
    IoT_Idle();
    String ColorHexVal = "#000000";
```

```
String title_web = "Brick'R'knowledge BRK-Clock MQTT-Werte";
String stat = "";
if (MQTT_Client.connected())
{
    stat = "Verbindung hergestellt";
}
else
{
    stat = "Server wird gesucht";
}

String content =                                                  // Die HTML-Seite in lesbarer Formatierung,
// darf auch Variablen enthalten
"<html>\
<head>\
<title>" + cleanhtml(title_web) + "</title>\
<style>\
body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
</style>\
<style type='text/css'>\
h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
</style>\
</head>\
<body>\
<h1>" + title_web + "</h1>\
<p> </p>\
<p>" + cleanhtml("Verbindungsstatus mit Server '" + MQTT_ServerURL + "': " + stat)    + "\
<p> </p>\
<p>" + cleanhtml(fillcenter(" Temperaturen ", "-", 80)) + "</p>\
<p> </p>";
content +=
"<p>" + cleanhtml("Hamburg      = " + strrealform(MQTT_AllnetTemperaturHamburg,      32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Berlin       = " + strrealform(MQTT_AllnetTemperaturBerlin,       32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Frankfurt    = " + strrealform(MQTT_AllnetTemperaturFrankfurt,    32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Stuttgart    = " + strrealform(MQTT_AllnetTemperaturStuttgart,    32, 2, 1, false, false) + "°C  ")            + "</p>";
content +=
"<p>" + cleanhtml("Hannover     = " + strrealform(MQTT_AllnetTemperaturHannover,     32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("München      = " + strrealform(MQTT_AllnetTemperaturMuenchen,     32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Köln          = " + strrealform(MQTT_AllnetTemperaturKoeln,        32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Düsseldorf   = " + strrealform(MQTT_AllnetTemperaturDuesseldorf,  32, 2, 1, false, false) + "°C  ")            + "</p>";
content +=
"<p>" + cleanhtml("Nürnberg     = " + strrealform(MQTT_AllnetTemperaturNuernberg,    32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Dresden      = " + strrealform(MQTT_AllnetTemperaturDresden,      32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Naumburg     = " + strrealform(MQTT_AllnetTemperaturNaumburg,     32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Heidelberg   = " + strrealform(MQTT_AllnetTemperaturHeidelberg,   32, 2, 1, false, false) + "°C  ")            + "</p>";
content +=
"<p>" + cleanhtml("Bremen       = " + strrealform(MQTT_AllnetTemperaturBremen,       32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Dortmund     = " + strrealform(MQTT_AllnetTemperaturDortmund,     32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Kiel         = " + strrealform(MQTT_AllnetTemperaturKiel,         32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Leipzig      = " + strrealform(MQTT_AllnetTemperaturLeipzig,      32, 2, 1, false, false) + "°C  ")            + "</p>";
```

```
content +=
"<p>" + cleanhtml("Regensburg   = " + strrealform(MQTT_AllnetTemperaturRegensburg,   32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Rostock      = " + strrealform(MQTT_AllnetTemperaturRostock,      32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Saarbrücken  = " + strrealform(MQTT_AllnetTemperaturSaarbruecken, 32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Trier        = " + strrealform(MQTT_AllnetTemperaturTrier,        32, 2, 1, false, false) + "°C  ")            + "</p>";
content +=
"<p>" + cleanhtml("Ulm          = " + strrealform(MQTT_AllnetTemperaturUlm,          32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Würzburg     = " + strrealform(MQTT_AllnetTemperaturWuerzburg,    32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Oldenburg    = " + strrealform(MQTT_AllnetTemperaturOldenburg,    32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Potsdam      = " + strrealform(MQTT_AllnetTemperaturPotsdam,      32, 2, 1, false, false) + "°C  ")            + "</p>";
content +=
"<p>" + cleanhtml("Cottbus      = " + strrealform(MQTT_AllnetTemperaturCottbus,      32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Schwerin     = " + strrealform(MQTT_AllnetTemperaturSchwerin,     32, 2, 1, false, false) + "°C  ")            + "</p>\
<p>" + cleanhtml("Mainz         = " + strrealform(MQTT_AllnetTemperaturMainz,        32, 2, 1, false, false) + "°C" + spc(19))    + \
cleanhtml("Wiesbaden    = " + strrealform(MQTT_AllnetTemperaturWiesbaden,    32, 2, 1, false, false) + "°C  ")            + "</p>\
<p> </p>";
content +=
"<p>" + cleanhtml(fillcenter(" Aktien- und Währungskurse ", "-", 81)) + "</p>\
<p> </p>";
content +=
"<p>" + cleanhtml("DAX          = " + strrealform(MQTT_AllnetFinanceDaxVal,          32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceDaxDifVal,       32, 1, 2, false, false, true) + " (") + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceDaxDifPerc,      32, 1, 2, false, false, true) + "%)") + "</p>";
content +=
"<p>" + cleanhtml("MDAX         = " + strrealform(MQTT_AllnetFinanceMDaxVal,         32, 6, 2, true, false)  + spc(16))    + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceMDaxDifVal,      32, 1, 2, false, false, true) + " (") + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceMDaxDifPerc,     32, 1, 2, false, false, true) + "%)") + "</p>";
content +=
"<p>" + cleanhtml("ESTX50       = " + strrealform(MQTT_AllnetFinanceEStx50Val,       32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceEStx50DifVal,    32, 1, 2, false, false, true) + " (")  + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceEStx50DifPerc,   32, 1, 2, false, false, true) + "%)")  + "</p>";
content +=
"<p>" + cleanhtml("Gold         = " + strrealform(MQTT_AllnetFinanceGoldVal,         32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceGoldDifVal,      32, 1, 2, false, false, true) + " (")  + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceGoldDifPerc,     32, 1, 2, false, false, true) + "%)")  + "</p>";
content +=
"<p>" + cleanhtml("Öl           = " + strrealform(MQTT_AllnetFinanceOelVal,          32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceOelDifVal,       32, 1, 2, false, false, true) + " (")  + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceOelDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>";
content +=
"<p>" + cleanhtml("EUR/USD      = " + strrealform(MQTT_AllnetFinanceEURVal,          32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceEURDifVal,       32, 1, 2, false, false, true) + " (")  + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceEURDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>";
content +=
"<p>" + cleanhtml("BitCoin/USD  = " + strrealform(MQTT_AllnetFinanceBTCVal,          32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceBTCDifVal,       32, 1, 2, false, false, true) + " (")  + \
cleanhtml(                    strrealform(MQTT_AllnetFinanceBTCDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>";
content +=
"<p>" + cleanhtml("Ethereum/USD = " + strrealform(MQTT_AllnetFinanceETHVal,          32, 6, 2, true, false) + spc(16))     + \
cleanhtml("Differenz    = " + strrealform(MQTT_AllnetFinanceETHDifVal,       32, 1, 2, false, false, true) + " (")  + \
```

```
    cleanhtml(                    strrealform(MQTT_AllnetFinanceETHDifPerc,      32, 1, 2, false, false, true) + "%)")  + "</p>\
    <p> </p>";
    content +=
    "<p>" + cleanhtml(fillcenter(" Aktualisieren ", "-", 80)) + "</p>";
    content +=
    "<p> </p>\
    " + IoT_WebFormActionButton ("form25", "Alle MQTT-Werte aktualisieren")   + "\
    <p> </p>";
    content +=
    "<p>" + cleanhtml(fillcenter(" System ", "-", 80)) + "</p>";
    content +=
    "<p> </p>\
    " + IoT_WebFormActionButton ("form18", "Startseite anzeigen")   + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                                  // HTTP-Seite aktualisieren
    content = "";
}

void handleForm26 ()
{
    IoT_Idle();
    String title_web = "Brick'R'knowledge BRK-Clock Farbe";
    String ColorHexVal = "#000000";
    String content =                                                  // Die HTML-Seite in lesbarer Formatierung,
    // darf auch Variablen enthalten
    "<html>\
    <head>\
    <title>" + cleanhtml(title_web) + "</title>\
    <style>\
    body { background-color: #FFFFFF; font-family: Menlo, Monaco, Courier, monospace; Color: " + ColorHexVal + "; }\
    </style>\
    <style type='text/css'>\
    h1 { font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif; Color: #000088; }\
    </style>\
    </head>\
    <body>\
    <h1>" + title_web + "</h1>\
    <p>" + cleanhtml(fillcenter(" BRK-Clock Farbe & Alarm ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormOpen("form01") + "\
    " + IoT_WebCheckBox(" Wecker eingeschaltet  ", "alarm", "chkd", sgn(BRKclock_alarmMode))
    + IoT_WebInput("H:", "wh", str(BRKclock_alarmHour), 5)
    + IoT_WebInput(" M:", "wm", strform(BRKclock_alarmMinute, 48, 2, false), 5) + "<br>\
    <br>\
    " + IoT_WebCheckBox(" Temperaturanzeige jede Minute für 5 Sekunden", "display", "chkd", BRKclock_temperatureMode) + "<br>\
    <br>";
    content +=
    IoT_WebRadioButton(" Zufällige Farbe",  0, "COLORgroup", "random",  BRKclock_colorMode) + "<br>\
```

```
    " + IoT_WebRadioButton(" Weiße Schrift",    1, "COLORgroup", "black",   BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Rote Schrift",     2, "COLORgroup", "red",     BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Grüne Schrift",    3, "COLORgroup", "green",   BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Blaue Schrift",    4, "COLORgroup", "blue",    BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Türkise Schrift",  5, "COLORgroup", "cyan",    BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Violette Schrift", 6, "COLORgroup", "magenta", BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Gelbe Schrift",    7, "COLORgroup", "yellow",  BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" Orange Schrift",   8, "COLORgroup", "orange",  BRKclock_colorMode) + "<br>\
    " + IoT_WebRadioButton(" RGBW Schrift    ", 9, "COLORgroup", "user",    BRKclock_colorMode);
    content +=
    IoT_WebInput("R:", "ur", str(BRKclock_clockColorRed), 5) + IoT_WebInput(" G:", "ug", str(BRKclock_clockColorGreen), 5)
    + IoT_WebInput(" B:", "ub", str(BRKclock_clockColorBlue), 5) + IoT_WebInput(" W:", "uw", str(BRKclock_clockColorWhite), 5) + "<br>\
    " + IoT_WebHtab (26) + cleanhtml("RGBW Temperatur ")
    + IoT_WebInput("R:", "tr", str(BRKclock_tempColorRed), 5) + IoT_WebInput(" G:", "tg", str(BRKclock_tempColorGreen), 5)
    + IoT_WebInput(" B:", "tb", str(BRKclock_tempColorBlue), 5) + IoT_WebInput(" W:", "tw", str(BRKclock_tempColorWhite), 5) + "<br>\
    <br>";
    content +=
    IoT_WebFormSubmitButton("Einstellungen der BRK-Clock anwenden") + "<br>\
    " + IoT_WebFormClose() + "\
    <p>" + cleanhtml(fillcenter(" System ", "-", 64)) + "</p>\
    <p> </p>\
    " + IoT_WebFormActionButton ("form18", "Startseite anzeigen")   + "\
    </body>\
    </html>";

    IoT_WebUpdate(&content);                                                // HTTP-Seite aktualisieren
    content = "";
}
```

BRK-Matrix Color Demo Listing

Das folgende Programm mit 167 Zeilen ist nicht Bestandteil des IoT-Handbuchs und beinhaltet zahlreiche Farbdemos für die 8 x 8 RGBW BRK-Matrix mit integriertem Microcontroller EPS8266. Diese werden über das Webinterface des EPS8266 zur Auswahl gestellt. Jede Demo kann in Farbton, Helligkeit und Geschwindigkeit variiert werden.

```
BRK-Matrix Color Demonstration
------------------------------

WLAN wird gestartet...
*WM:
*WM: AutoConnect
*WM: Connecting as wifi client...
*WM: Using last saved values, should be faster
*WM: Connection result:
*WM: 3
*WM: IP Address:
*WM: 192.168.1.26
Connected to AirPort Extreme, IP: 192.168.1.26
Konfiguration wird vorbereitet...
HTTP Server wird eingerichtet...HTTP Server wurde gestartet.
WLAN-Verbindung wird geprüft... OK
WLAN-Verbindung wird geprüft... OK
WLAN-Verbindung wird geprüft... OK
```

```
//  BRK-Matrix Color Demonstration
//  Nur mit einem EPS8266 Chip ohne den IoT-Brick
//
//  Unter Verwendung der IoT Brick Library ab Version 1.01


#define BRKclock_PIN 13                                         // Led Pin

#include <all_IoT.h>
#include <all_BRKclock.h>
#include "index.html.h"
#include "main.js.h"
#include <WS2812FX.h>                                           // Library mit all den Demos

WS2812FX ws2812fx = WS2812FX(64, BRKclock_PIN, NEO_GRBW + NEO_KHZ800);

#define WIFI_TIMEOUT       30000                                // WLAN Verbindung jede ... ms. prüfen
#define BRIGHTNESS_STEP    15                                   // Helligkeit um diesen Wert erhöhen/verringern
#define SPEED_STEP         10                                   // Geschwindigkeit um diesen Wert erhöhen/verringern

unsigned long LastWLANtime = 0;
String        modes        = "";

void setup ()
{
    IoT_Init(false);                                            // EPS8266 initialisieren (ohne OLED/ADC)
    t_TerminalInit();                                           // Terminal initialisieren
    BRKclock_Init();                                            // BRK-Clock initialisieren
    t_TerminalClearScreen();
    Serial.println("BRK-Matrix Color Demonstration");
    Serial.println("-----------------------------");
    Serial.println("");

    Serial.println("WLAN wird gestartet...");
    BRKclock_WLANautoConnect(true);

    Serial.println("Konfiguration wird vorbereitet...");
    modes.reserve(5000);                                        // String-Speicherplatz reservieren
    modes_setup();                                              // String für Demo-Auswahl einrichten

    ws2812fx.init();
    ws2812fx.setMode(FX_MODE_STATIC);
    ws2812fx.setColor(0xFF5900);
    ws2812fx.setSpeed(200);
    ws2812fx.setBrightness(255);
    ws2812fx.start();

    Serial.print("HTTP Server wird eingerichtet...");
    IoT_WebServer.on("/", srv_handle_index_html);
    IoT_WebServer.on("/main.js", srv_handle_main_js);
```

```
    IoT_WebServer.on("/modes", srv_handle_modes);
    IoT_WebServer.on("/set", srv_handle_set);
    IoT_WebServer.onNotFound(srv_handle_not_found);
    IoT_WebServer.begin();
    Serial.println("HTTP Server wurde gestartet.");
}


void loop ()
{
    unsigned long now = millis();
    IoT_WebServer.handleClient();
    ws2812fx.service();
    if (now - LastWLANtime > WIFI_TIMEOUT)
    {
        Serial.print("WLAN-Verbindung wird geprüft... ");
        if (IoT_WLANstatus() != "Connected")
        {
            Serial.println("Verbindung muss neu aufgebaut werden...");
            BRKclock_WLANautoConnect(true);
        }
        else
        {
            Serial.println("OK");
        }
        LastWLANtime = now;
    }
}



// ######################################################
// #  Build <li> string for all modes                   #
// ######################################################

void modes_setup()
{
    modes = "";
    for (byte i = 0; i < ws2812fx.getModeCount(); i++)
    {
        modes += "<li><a href='#' class='m' id='";
        modes += i;
        modes += "'>";
        modes += ws2812fx.getModeName(i);
        modes += "</a></li>";
    }
}
```

```
// ##################################################
// #  Webserver Functions                           #
// ##################################################

void srv_handle_not_found()
{
    IoT_WebServer.send(404, "text/plain", "File Not Found");
}

void srv_handle_index_html()
{
    IoT_WebServer.send_P(200,"text/html", index_html);
}

void srv_handle_main_js()
{
    IoT_WebServer.send_P(200,"application/javascript", main_js);
}

void srv_handle_modes()
{
    IoT_WebServer.send(200,"text/plain", modes);
}

void srv_handle_set()
{
    for (byte i = 0; i < IoT_WebServer.args(); i++)
    {
        if (IoT_WebServer.argName(i) == "c")
        {
            uint32_t tmp = (uint32_t) strtol(&IoT_WebServer.arg(i)[0], NULL, 16);
            if(tmp >= 0x000000 && tmp <= 0xFFFFFF)
            {
                ws2812fx.setColor(tmp);
            }
        }

        if (IoT_WebServer.argName(i) == "m")
        {
            uint8_t tmp = (uint8_t) strtol(&IoT_WebServer.arg(i)[0], NULL, 10);
            ws2812fx.setMode(tmp % ws2812fx.getModeCount());
        }

        if (IoT_WebServer.argName(i) == "b")
        {
            if (IoT_WebServer.arg(i)[0] == '-')
            {
                ws2812fx.decreaseBrightness(BRIGHTNESS_STEP);
            }
            else
```

```
            {
                ws2812fx.increaseBrightness(BRIGHTNESS_STEP);
            }
        }

        if (IoT_WebServer.argName(i) == "s")
        {
            if (IoT_WebServer.arg(i)[0] == '-')
            {
                ws2812fx.decreaseSpeed(SPEED_STEP);
            }
            else
            {
                ws2812fx.increaseSpeed(SPEED_STEP);
            }
        }
    }
    IoT_WebServer.send(200, "text/plain", "OK");
}
```

Type Library Listing

Die Type-Library definiert je nach verwendeter Handware die fehlenden Integer-Datentypen. Weiterhin werden Datentypen deklariert, die für alle anderen Libraries benötigt werden.

```
// Type Library
// 1.00 - 2017-05-23
// 1.01 - 2017-06-23
// (c) Copyright 2017
//
// Zur Verwendung mit allen Allnet Libraries


// *** Datentypen ***

#if defined(ESP8266)    // ESP 8266 (32 Bit)
typedef unsigned char uint8_t;
typedef unsigned char uint8;
typedef signed   char int8_t;
typedef signed   char int8;
#endif
#if defined(__MK20DX256__) || defined(__SAM3X8E__) || defined(ESP8266) || defined(ESP32)    // Teensy, Arduino Due, ESP 8266 (32 Bit)
#else
typedef uint16_t uint;
typedef uint16_t uint16;
typedef int16_t  int16;
#endif
#if defined(ESP8266) || defined(ESP32)                                                      // ESP 8266 (32 Bit)
#else
typedef uint32_t ulong;
typedef uint32_t uint32;
typedef int32_t  int32;
#endif
typedef int64_t  int64;
typedef uint64_t uint64;

struct t_DateTime
{
    byte day = 0;
    byte month = 0;
    uint year = 0;
    byte second = 0;
    byte minute = 0;
    byte hour = 0;
};
```

```
struct t_Color
{
    ulong red   = 0;
    ulong green = 0;
    ulong blue  = 0;
    ulong white = 0;
};
```

IoT-Brick Library Listing

Die IoT Brick Library deckt aktuell die im IoT-Handbuch vorhandenen Programme 6.5.x und 6.6.x ab. In der IoT Brick Library werden alle benötigten anderen Libraries geladen. Die String-Library stellt dabei alle möglichen String-Befehle zur Verfügung, die das Arbeiten mit Strings erheblich leichter und übersichtlicher macht.

```
// IoT Brick Library
// 1.00 - 2017-05-17
// 1.01 - 2017-06-15
// 1.02 - 2017-06-23
// (c) Copyright 2017
//
// Zur Verwendung mit dem Allnet IoT-Brick


#include <MCP3421.h>
#include <ESP8266WiFi.h>
#include <SSD1306Wire.h>
#include <TimeLib.h>
#include <NtpClientLib.h>
#include <WiFiManager.h>
#include <EEPROM.h>
#include "ESP8266WebServer.h"

#include "all_Type.h"
#include "all_String.h"
#include "all_Terminal.h"


// *** Globale Konstanten ***

const bool t_ShowMessages               = true;      // Status-Mitteilungen anzeigen
const bool t_HideMessages               = false;     // Status-Mitteilungen nicht anzeigen
const int  IoT_Analog10bit              = 0;         // ADC im ESP8266 hat den Index 0


// *** Globale Variablen ***

MCP3421          IoT_ADC18bit           = MCP3421(); // Interner 18 Bit Wandler vom Typ MCP3421
SSD1306Wire      IoT_Display(0x3c, 4, 5);            // Variable für das OLED Display, wird hier initialisiert
ESP8266WebServer IoT_WebServer(80);                  // Webserver starten auf Port 80

bool             IoT_WLANinited         = false;     // Variable für den erfolgreichen (oder nicht) WLAN Connect
bool             IoT_OLEDinited         = false;     // Variable für die Initialisierung der OLED-Display Benutzung
bool             IoT_NTPinited          = false;     // Variable für die Initialisierung der NTP Benutzung
bool             IoT_NTPeventAvail      = false;     // Variable für das Auftreten eines NTP Events
NTPSyncEvent_t   IoT_NTPcurrentEvent;                // Der zuletzt vom TimeServer empfangene Event
```

```
long             IoT_EEPROMsize         = 0;         // Größe des verbauten EPROMs


// *** Variablen für verwendete DHCP-IP-Adressen ***

byte             IoT_dynamicIP_address0 = 0;         // 4 Bytes IP-Adresse
byte             IoT_dynamicIP_address1 = 0;
byte             IoT_dynamicIP_address2 = 0;
byte             IoT_dynamicIP_address3 = 0;
byte             IoT_dynamicIP_gateway0 = 0;         // 4 Bytes Gateway
byte             IoT_dynamicIP_gateway1 = 0;
byte             IoT_dynamicIP_gateway2 = 0;
byte             IoT_dynamicIP_gateway3 = 0;
byte             IoT_dynamicIP_subnet0  = 0;         // 4 Bytes Subnetzmaske
byte             IoT_dynamicIP_subnet1  = 0;
byte             IoT_dynamicIP_subnet2  = 0;
byte             IoT_dynamicIP_subnet3  = 0;


// *** Forward-Deklarationen ***

void IoT_Idle ();
void IoT_WatchDog (bool active);
void IoT_DisplayClear (int fontSize);
void IoT_DisplayFontSize (int fontSize);
void IoT_DisplayUpdate ();
void IoT_DisplayDrawText (int x, int y, String text);
void IoT_DisplayAlignText (OLEDDISPLAY_TEXT_ALIGNMENT alignment);


// *** Initialisierung des IoT-Bricks ***

void IoT_Init (bool IoTbrick)                       // Standard-Initialisierung, muss als Erstes aufgerufen werden
{
    IoT_WatchDog(true);                              // IoT-Brick-Watchdog auf 5 Sekunden festlegen
    pinMode(0, INPUT);                               // Eingebauten Taster benutzbar machen
    switch (t_CPUtype())
    {
        case t_ESP8266:
            Serial.begin(115200);
            IoT_EEPROMsize = 4096;
            EEPROM.begin(IoT_EEPROMsize);                // Daten aus dem EEPROM auslesen
            break;
        case t_ArduinoMega2560:
            Serial.begin(921600);
            break;
        case t_Teensy31:
            Serial.begin(921600);                        // Beherrscht alle Baudraten, macht bis zu 14 MBit.
            break;
        default: // t_ArduinoNano, t_ArduinoDue u.a.
```

```
            Serial.begin(115200);
            break;
    }

    if (IoTbrick)
    {
        IoT_Display.init();                             // Initialisiert die OLED graphische Anzeige
        IoT_OLEDinited = true;                          // Variable für OLED-Initialisierung setzen
        //IoT_Display.flipScreenVertically();           // Anzeige spiegeln (optional, hier nur zur Veranschaulichung)
        IoT_DisplayClear(10);                           // Eingebautes OLED-Display löschen
        pinMode(13, OUTPUT);                            // GPIO 13 aus Ausgabe schalten
        pinMode(14, OUTPUT);                            // GPIO 14 aus Ausgabe schalten
        digitalWrite(13, LOW);                          // GPIO 13 auf 0V schalten
        digitalWrite(14, LOW);                          // GPIO 14 auf 0V schalten
        IoT_ADC18bit.init(0x68,3,0);                    // Init MCP3421: I2C-Adresse, 18 Bit Modus, keine Verstärkung
    }
}


// *** WLAN des IoT-Bricks ***

String IoT_WLANstatus ()
{
    switch (WiFi.status())
    {
        case 0:
            return ("Idle");
            break;
        case 1:
            return ("No SSID available");
            break;
        case 2:
            return ("Scan completed");
            break;
        case 3: // WL_CONNECTED
            return ("Connected");
            break;
        case 4:
            return ("Connect failed");
            break;
        case 5:
            return ("Connection lost");
            break;
        case 6:
            return ("Disconnected");
            break;
        default:
            return ("N/A");
            break;
    }
```

```
    IoT_Idle();
}

bool IoT_WLANconnected ()                                       // Ist eine WLAN-Verbindung erfolgreich hergestellt?
{
    return (WiFi.isConnected());
}

bool IoT_WLANconnect (String WLANssid, String WLANpassword)     // Funktion um die WLAN Verbindung aufzubauen
{                                                               // incl. Fortschrittsbalken
    IoT_WLANinited = false;
    int progress = 0;

    Serial.print(chr(12));

    // Da manche Access Points langsamer antworten als andere, muss man ein paar Sekunden auf die Antwort warten.
    // Solange WLAN nicht verbunden ist, Fortschrittsbalken im Display und Punkte in der Console anzeigen.
    while (not(IoT_WLANinited) && (progress < 100))
    {
        if ((progress % 25) == 0)
        {
            if (progress == 50)                                 // Wenn nach 50 versuchen noch keine Wifi-Verbindung
            {                                                   // zu Stande kommt, dann alle Konfigurationsdaten
                Serial.println("");                               // aus dem Flash-Speicher löschen
                ESP.eraseConfig();
                Serial.print("Resetting Configuration. ");
            }
            if ((progress == 0) || (progress == 50))
            {
                Serial.println("WLAN access point " + WLANssid);
                Serial.print("Connecting ");
            }
            IoT_Idle();
            WiFi.persistent(false);                             // Räumt mit alten RF-Daten auf und sorgt dafür, dass
            WiFi.mode(WIFI_OFF);                                // zuverlässig eine WiFi-Verbindung aufgebaut wird,
            WiFi.mode(WIFI_STA);                                // auch dann, wenn WiFi erst später startet. WiFi erst
            WiFi.begin(WLANssid.c_str(), WLANpassword.c_str());  // ausschalten und dann wieder einschalten.
            delay(500);                                         // 500 ms. warten
        }
        IoT_WLANinited = (IoT_WLANstatus() == "Connected");
        //Serial.println(String(progress) + " Status = " + IoT_WLANstatus() + " = " + boolstr(IoT_WLANinited, "True", "False"));
        Serial.print(".");

        if (IoT_OLEDinited)
        {
            IoT_Display.clear();
            IoT_Display.drawProgressBar(0, 32, 120, 10, progress);
            IoT_Display.setTextAlignment(TEXT_ALIGN_CENTER);
            IoT_Display.drawString(64, 15, String(progress) + "%");
            IoT_Display.display();
```

```
        }
        IoT_Idle();

        if (not(IoT_WLANinited))                // Verbindung noch nicht hergestellt?
        {
            delay(500);                           // 500 ms. warten
            progress++;                           // Durchgang um 1 erhöhen
        }
    }
    Serial.println("");
    if (IoT_WLANinited) // Falls Verbindung erfolgreich, zeige WLAN Namen und die DHCP IP Adresse in der Console
    {
        Serial.print("Connected to ");
        Serial.print(WiFi.SSID());
        Serial.print(", IP: ");
        Serial.println(WiFi.localIP());
        IoT_dynamicIP_address0 = WiFi.localIP()[0];                 // 4 Bytes IP-Adresse
        IoT_dynamicIP_address1 = WiFi.localIP()[1];
        IoT_dynamicIP_address2 = WiFi.localIP()[2];
        IoT_dynamicIP_address3 = WiFi.localIP()[3];
        IoT_dynamicIP_gateway0 = WiFi.gatewayIP()[0];               // 4 Bytes Gateway
        IoT_dynamicIP_gateway1 = WiFi.gatewayIP()[1];
        IoT_dynamicIP_gateway2 = WiFi.gatewayIP()[2];
        IoT_dynamicIP_gateway3 = WiFi.gatewayIP()[3];
        IoT_dynamicIP_subnet0  = WiFi.subnetMask()[0];              // 4 Bytes Subnetzmaske
        IoT_dynamicIP_subnet1  = WiFi.subnetMask()[1];
        IoT_dynamicIP_subnet2  = WiFi.subnetMask()[2];
        IoT_dynamicIP_subnet3  = WiFi.subnetMask()[3];
    }
    else // gebe Fehlermeldung in der Console aus, falls Verbindung fehlgeschlagen
    {
        Serial.println("Connection failed");
    }

    return (IoT_WLANinited);
}

String IoT_WLANaddress (bool leadingZero) // Lokale IP-Adresse des WLAN-Netzwerkes (wahlweise mit führenden Nullen)
{
    return (strIP(WiFi.localIP(), leadingZero));
}

String IoT_WLANgateway (bool leadingZero) // Gateway (Router) IP-Adresse des WLAN-Netzwerkes (wahlweise mit führenden Nullen)
{
    return (strIP(WiFi.gatewayIP(), leadingZero));
}

String IoT_WLANsubnet (bool leadingZero) // Subnetzmaske des WLAN-Netzwerkes (wahlweise mit führenden Nullen)
{
    return (strIP(WiFi.subnetMask(), leadingZero));
```

```
}

String IoT_WLANssid () // Name des WLAN-Netzwerkes
{
    return (WiFi.SSID());
}

int IoT_WLANrssi () // Signalstärke in dBm (immer ein negativer Wert, z.B. -70 dBm)
{
    return (WiFi.RSSI());
}

String IoT_WLANuptime (bool germanLanguage)                         // Dauer der aktuellen WLAN-Verbindung
{
    String d = NTP.getUptimeString();                                 // Zeitdauer im Format "T days hh:mm:ss"
    if (germanLanguage)
    {
        return (replace(d, "days", "Tage"));                            // Zeitdauer im Format "T Tage hh:mm:ss"
    }
    else
    {
        return (d);                                                     // Zeitdauer im Format "T days hh:mm:ss"
    }
}

String IoT_WLANstartDatetime ()                                     // Startzeit der aktuellen WLAN-Verbindung
{
    String d = NTP.getTimeDateString(NTP.getFirstSync()).c_str();     // Datum & Uhrzeit im Format tt/mm/jjjj hh:mm:ss
    return (replace(d, "/", "."));                                    // Datum & Uhrzeit im Format tt.mm.jjjj hh:mm:ss
}

void IoT_WLANautoConnect (bool useSavedSettings)                    // Verbindung mit WiFi-Manager herstellen
{
    WiFiManager wifiManager;
    if (not(useSavedSettings))
    {
        wifiManager.resetSettings();
    }
    if (IoT_OLEDinited)
    {
        IoT_DisplayClear(16);                                         // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayAlignText(TEXT_ALIGN_CENTER);
        IoT_DisplayDrawText(63, 0, "WLAN wählen:");
        IoT_DisplayFontSize(24);
        IoT_DisplayDrawText(63, 20, "IoT-Brick");
        IoT_DisplayFontSize(10);
        IoT_DisplayDrawText(63, 50, "Danach HotSpot wählen");
        IoT_DisplayUpdate();                                          // Display aktualisieren
        IoT_DisplayClear(16);                                         // OLED Display löschen (16 Punkt Schrift)
    }
```

```
    wifiManager.autoConnect("IoT-Brick");                               // Verbindung ggf. mit gespeicherten Einstellungen
    if (IoT_OLEDinited)
    {
        IoT_DisplayUpdate();                                             // Display aktualisieren
    }
    Serial.print("Connected to ");
    Serial.print(WiFi.SSID());
    Serial.print(", IP: ");
    Serial.println(WiFi.localIP());
    IoT_WLANinited = true;
    IoT_dynamicIP_address0 = WiFi.localIP()[0];                         // 4 Bytes IP-Adresse
    IoT_dynamicIP_address1 = WiFi.localIP()[1];
    IoT_dynamicIP_address2 = WiFi.localIP()[2];
    IoT_dynamicIP_address3 = WiFi.localIP()[3];
    IoT_dynamicIP_gateway0 = WiFi.gatewayIP()[0];                       // 4 Bytes Gateway
    IoT_dynamicIP_gateway1 = WiFi.gatewayIP()[1];
    IoT_dynamicIP_gateway2 = WiFi.gatewayIP()[2];
    IoT_dynamicIP_gateway3 = WiFi.gatewayIP()[3];
    IoT_dynamicIP_subnet0  = WiFi.subnetMask()[0];                      // 4 Bytes Subnetzmaske
    IoT_dynamicIP_subnet1  = WiFi.subnetMask()[1];
    IoT_dynamicIP_subnet2  = WiFi.subnetMask()[2];
    IoT_dynamicIP_subnet3  = WiFi.subnetMask()[3];
}


// *** Web-Server (HTTP) des IoT-Bricks ***

void IoT_WebUpdate (String *html)
{
    IoT_Idle();
    IoT_WebServer.handleClient();                                       // Bediene die http Anfragen
    IoT_WebServer.send(200, "text/html", *html);                        // HTTP-Statuscode 200 = OK (Anfrage erfolgreich)
    IoT_WebServer.handleClient();                                       // Bediene die http Anfragen
    IoT_Idle();
}

void IoT_WebPrintAllFields ()
{
    String s = "";
    byte argumentCount = IoT_WebServer.args();                          // Anzahl der Felder auf der Web-Seite
    if (argumentCount > 0)                                              // Ist überhaupt eine Eingabe vorhanden?
    {
        for (byte i = 0; i < argumentCount; i++)                        // Alle erlaubten Eingabefelder durchsuchen
        {
            Serial.print("Field #" + strform(i, 48, 2, true) + ": ");   // Die Ausgabe von Unicodes stört die Terminal-Ausgabe,
            s = cleanasc(IoT_WebServer.argName(i));                     // daher hier alle Unicode-Zeichen sinnvoll ersetzen
            Serial.print(left(s + spc(16), 16));                        // Feldname ausgeben, max. 16 Zeichen (Asc II)
            Serial.print(" = " + chr(34));                              // Feldinhalt in Anführungszeichen ausgeben,
            Serial.print(cleanasc(IoT_WebServer.arg(i)));               // Alle Unicode-Zeichen sinnvoll ersetzen
            Serial.println(chr(34));                                    // max. 32 Zeichen (Asc II)
```

```
        }
    }
    IoT_Idle();
}

bool IoT_WebTestField (String varName)                          // Ergibt true, wenn ein Web-Feld mit dem Namen existiert
{
    byte argumentCount = IoT_WebServer.args();                    // Anzahl der Felder auf der Web-Seite
    if (argumentCount > 0)                                        // Ist überhaupt eine Eingabe vorhanden?
    {
        for (byte i = 0; i < argumentCount; i++)                    // Alle erlaubten Eingabefelder durchsuchen
        {
            if (IoT_WebServer.argName(i) == varName)                  // Das Feld mit dem angegebenen varName suchen
            {
                return (true);                                         // Feld auslesen und als Funktionsergebnis übergeben
            }
        }
    }
    return (false);
}

String IoT_WebGetField (String varName)                         // Ergibt den Inhalt des Web-Feldes
{
    byte argumentCount = IoT_WebServer.args();                    // Anzahl der Felder auf der Web-Seite
    if (argumentCount > 0)                                        // Ist überhaupt eine Eingabe vorhanden?
    {
        for (byte i = 0; i < argumentCount; i++)                    // Alle erlaubten Eingabefelder durchsuchen
        {
            if (IoT_WebServer.argName(i) == varName)                  // Das Feld mit dem angegebenen varName suchen
            {
                return (IoT_WebServer.arg(i));                         // Feld auslesen und als Funktionsergebnis übergeben
            }
        }
    }
    return ("");
}


// *** Web-Seitenaufbau ***

String IoT_WebFormOpen (String formName)
{
    String s = "<form action = 'http://" + IoT_WLANaddress(false) + "/" + formName + "' method='POST'>";
    return (s);
}

String IoT_WebFormClose ()
{
    String s = "</form>";
    return (s);
```

```
}

String IoT_WebFormSubmitButton (String formName)
{
    String s = "<input type='submit' value='" + cleanhtml(formName) + "'>";
    return (s);
}

String IoT_WebFormActionButton (String actionName, String title)
{
    String s = IoT_WebFormOpen(actionName) + IoT_WebFormSubmitButton(title) + IoT_WebFormClose();
    return (s);
}

String IoT_WebCheckBox (String title, String varName, String varValue, bool checked)
{
    String s = "<input type='checkbox' name='" + varName + "' value='" + varValue + "'";
    if (checked)
    {
        s += " checked";
    }
    s += ">";
    s += cleanhtml(title);
    return (s);
}

String IoT_WebRadioButton (String title, int varIndex, String varName, String varValue, int selectedIndex)
{
    String s = "<input type='radio' id='" + varName + strform(varIndex, 48, 3, false) + "' name='" + varName + "' value='" + varValue + "'";
    if (varIndex == selectedIndex)
    {
        s += " checked";
    }
    s += ">";
    s += cleanhtml(title);
    return (s);
}

String IoT_WebInput (String title, String varName, String varValue, int width)
{
    String s = cleanhtml(title) + "<input type='text' name='" + varName + "' value='" + varValue + "' size='" + str(width) + "'>";
    return (s);
}

String IoT_WebClientIP ()
{
    String s = "<script type='text/javascript' src='https://l2.io/ip.js'></script>";
    return (s);
}
```

```
String IoT_WebHtab (int h)
{
    String s = "<span style='padding-left:" + String(h) + "px;'></span>";
    return (s);
}


// *** OLED-Display Funktionen ***

void IoT_DisplayFontSize (int fontSize)           // Schriftgröße setzen
{
    if (IoT_OLEDinited)
    {
        switch (fontSize)
        {
            case 10:
                IoT_Display.setFont(ArialMT_Plain_10);      // Schriftart setzen. Hiermit sind 6 Zeilen Text möglich
                break;
            case 16:
                IoT_Display.setFont(ArialMT_Plain_16);      // Schriftart setzen. Hiermit sind 4 Zeilen Text möglich
                break;
            case 24:
                IoT_Display.setFont(ArialMT_Plain_24);      // Schriftart setzen. Hiermit sind 2 Zeilen Text möglich
                break;
            default:
                IoT_Display.setFont(ArialMT_Plain_10);      // Schriftart setzen. Hiermit sind 6 Zeilen Text möglich
                break;
        }
    }
    IoT_Idle();
}

void IoT_DisplayClear (int fontSize)              // Bildschirm löschen und Schriftgröße setzen
{
    if (IoT_OLEDinited)
    {
        IoT_Display.clear();
        IoT_Display.setTextAlignment(TEXT_ALIGN_LEFT);  // Linksbündige Darstellung des Textes
        IoT_DisplayFontSize(fontSize);
    }
}

void IoT_DisplayUpdate ()                         // Bildschirm aktualisieren
{
    if (IoT_OLEDinited)
    {
        IoT_Display.display();
    }
    IoT_Idle();
}
```

```
void IoT_DisplayDrawText (int x, int y, String text) // Text an der angegebenen Koordinate schreiben
{
    if (IoT_OLEDinited)
    {
        IoT_Display.drawString(x, y, text);
    }
    IoT_Idle();
}

void IoT_DisplayAlignText (OLEDDISPLAY_TEXT_ALIGNMENT alignment)
{
    if (IoT_OLEDinited)
    {
        IoT_Display.setTextAlignment(alignment);
    }
}

void IoT_DisplayWLANstatus ()
{
    if (IoT_OLEDinited)
    {
        IoT_DisplayClear(10);                                       // OLED Display löschen (10 Punkt Schrift)
        String state = IoT_WLANstatus();                            // Verbindungsstatus
        if (IoT_WLANinited)                                         // Falls WLAN Verbindung erfolgreich, Status & IP im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 0, "WLAN: " + IoT_WLANssid());     // WLAN Netzwerk Name
            if (state == "Connected")                               // Wenn erfolgreich verbunden, dann Signalstärke hinzufügen
            {
                state = state + " (" + str(IoT_WLANrssi()) + "dBm)";    // Signalstärke
            }
            IoT_DisplayDrawText(5, 10, state);                      // Verbindungsstatus & Signalstärke
            IoT_DisplayDrawText(5, 20, "IP: " + IoT_WLANaddress(true));   // IP Adresse
            IoT_DisplayDrawText(5, 30, "GW: " + IoT_WLANgateway(true));   // IP Gateway
            IoT_DisplayDrawText(5, 40, "NT: " + IoT_WLANsubnet(true));    // IP Subnetzmaske
        }
        else                                                        // Falls WLAN Verbindung gescheitert ist, Status im OLED Display anzeigen
        {
            IoT_DisplayDrawText(5, 10, state);
        }
    }
}

// *** NTP Funktionen ***

bool IoT_NTPinit ()                             // Funktion um die NTP Verbindung aufzubauen
{
    if (IoT_WLANinited)                             // Nur wenn WLAN verbunden ist
    {
        NTP.begin("pool.ntp.org", 1, true);           // Verbindung zu NTP Zeit-Server herstellen
```

```
        NTP.setInterval(63);                            // Aktualisierungsintervall setzen
        IoT_NTPinited = true;

        NTP.onNTPSyncEvent                              // Event-Handler starten
        (
         [ ] (NTPSyncEvent_t event)                    // Variable "event" enthält den aktuellen NTPSyncEvent_t
         {
             IoT_NTPcurrentEvent = event;               // "event" Variable global zur Verfügung stellen
             IoT_NTPeventAvail = true;                  // Anzeigen, dass ein Event empfangen wurde
         }
         );

    }
    else
    {
        IoT_NTPinited = false;                          // Ohne WLAN ist kein NTP möglich
    }
    IoT_Idle();
    return (IoT_NTPinited);
}

String IoT_NTPtime ()
{
    return (NTP.getTimeStr());                      // Uhrzeit im Format hh:mm:ss
}

String IoT_NTPdate ()
{
    String d = NTP.getDateStr();                    // Datum im Format tt/mm/jjjj
    return (replace(d, "/", "."));                  // Datum im Format tt.mm.jjjj
}

String IoT_NTPdatetime ()
{
    String d = NTP.getTimeDateString();             // Datum & Uhrzeit im Format tt/mm/jjjj hh:mm:ss
    return (replace(d, "/", "."));                  // Datum & Uhrzeit im Format tt.mm.jjjj hh:mm:ss
}

bool IoT_NTPvalid ()
{
    return (IoT_NTPdate() != "01.01.1970");
}

String IoT_NTPdaylightSavingTime (bool germanLanguage)
{
    if (germanLanguage)
    {
        return (NTP.isSummerTime() ? "Sommerzeit" : "Winterzeit");
    }
    else
```

```
    {
        return (NTP.isSummerTime() ? "Summer Time" : "Winter Time");
    }
}

void IoT_NTPprintEvent (NTPSyncEvent_t event)
{
    if (event)                                                  // Fehlermeldung überprüfen
    {
        Serial.print("Time Sync error");
        if (event == noResponse)                                // NTP-Server antwortet nicht
        {
            Serial.print(": NTP server not reachable");
        }
        else if (event == invalidAddress)                       // Fehlerhafte IP-Adresse für den NTP-Server
        {
            Serial.print(": Invalid NTP server address");
        }
        Serial.println("");
    }
    else                                                        // Zeit wurde erfolgreich empfangen
    {
        Serial.print("Got NTP time: ");
        Serial.println(NTP.getTimeDateString(NTP.getLastNTPSync()));
    }
}


// *** Utilites für den IoT-Brick ***

void IoT_Idle ()                                // Verhindert das Festfrieren und Neustarten bei komplexen Aufgaben
{
    ESP.wdtFeed();                                // Watchdog Timer zurücksetzen
    //delay(1);                                     // Alternativ kann man auch 1 ms. warten, kostet aber CPU-Zeit
}

void IoT_WatchDog (bool active)                 // Schaltet den IoT-Brick-Watchdog ein (true) oder aus (false)
{
    ESP.wdtFeed();                                // Watchdog Timer zurücksetzen
    if (active)
    {
        ESP.wdtEnable(5000);                        // Watchdog Timer einschalten auf 5 Sekunden (5000 ms.)
    }
    else
    {
        ESP.wdtDisable();                           // Watchdog Timer ausschalten
    }
    ESP.wdtFeed();                                // Watchdog Timer zurücksetzen
}
```

```
bool IoT_Keypress ()                            // Eingebauten Taster abfragen
{
    return ((digitalRead(0) == LOW));
}

void IoT_WaitMessage ()
{
    IoT_DisplayClear(16);                       // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);    // Zentrierte Darstellung des Textes
    IoT_DisplayDrawText(64,  5, "Den Taster am");
    IoT_DisplayDrawText(64, 25, "IoT-Brick drücken");
    IoT_DisplayDrawText(64, 45, "um fortzufahren.");
    IoT_DisplayUpdate();
    ESP.wdtFeed();                              // Watchdog Timer zurücksetzen
}

void IoT_WaitKeypress (uint64 timeout, bool message)
{
    long startTime = millis();
    if (timeout == 0)                             // Timeout 0 bedeutet: Kein Timeout
    {
        timeout = 20000000000000;                   // 20 Milliarden Sekunden = mehr als 630 Jahre
    }
    if (message)
    {
        IoT_WaitMessage ();
    }
    while (not(IoT_Keypress()) && (abs(millis() - startTime) < timeout))
    {                                             // Wenn Taster nicht gedrückt wird und noch kein Timeout
        delay(1);                                   // 1 ms. warten
        ESP.wdtFeed();                              // Watchdog Timer zurücksetzen
        IoT_WebServer.handleClient();               // Bediene die http Anfragen
    }
    if (message)
    {
        IoT_DisplayClear(16);                       // OLED Display löschen (16 Punkt Schrift)
        IoT_DisplayUpdate();
    }
}

void IoT_WaitNoKeypress ()
{
    while (IoT_Keypress())                        // Warten, bis der Taster losgelassen wurde
    {
        delay(1);
        IoT_Idle();
        IoT_WebServer.handleClient();               // Bediene die http Anfragen
    }
}
```

```
void IoT_TerminalWaitInit (bool showMessage)
{
    if (showMessage == t_ShowMessages)
    {
        if (IoT_OLEDinited)
        {
            IoT_DisplayClear(16);
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);
            IoT_DisplayDrawText(63, 2, "Bitte mit");
            IoT_DisplayDrawText(63, 22, "Terminal");
            IoT_DisplayDrawText(63, 42, "verbinden");
            IoT_DisplayUpdate();
        }
    }

    t_TerminalRespond = "";
    while (t_TerminalWidth == 0)
    {
        t_TerminalInit();
        if (t_TerminalWidth == 0)
        {
            delay(500); // 500 ms. warten
        }
    }
    t_TerminalRespond = "";

    if (showMessage == t_ShowMessages)
    {
        if (IoT_OLEDinited)
        {
            IoT_DisplayClear(10);
            IoT_DisplayAlignText(TEXT_ALIGN_CENTER);
            IoT_DisplayDrawText(63, 0, "Verbindung");
            IoT_DisplayDrawText(63, 12, "ist hergestellt");
            IoT_DisplayDrawText(63, 24, "Protokoll:");
            IoT_DisplayDrawText(63, 36, t_TerminalType);
            IoT_DisplayDrawText(63, 48, String(t_TerminalWidth) + " x " + String(t_TerminalHeight) + " Zeichen");
            IoT_DisplayUpdate();
        }
    }
}

void IoT_EEPROMclear ()                         // Die Konfiguration des WiFi-Managers wird hiervon nicht berührt
{
    bool success = true;
    const byte zero = 0;
    for (int i = 0; i <= IoT_EEPROMsize; i++)
    {
        EEPROM.put(i, zero);
    }
```

```
}

void IoT_EEPROMupdate ()
{
    EEPROM.commit();                                // Daten in das EEPROM zurückschreiben
}

void IoT_ShutDown ()
{
    if (IoT_OLEDinited)
    {
        IoT_Display.clear();                        // Display löschen
        IoT_Display.display();                      // Display aktualisieren
    }
    if (IoT_EEPROMsize > 0)
    {
        EEPROM.end();                               // Daten in das EEPROM zurückschreiben und Speicherplatz freigeben
    }
    while (true)
    {
        ESP.wdtFeed();                              // Watchdog Timer zurücksetzen
        t_TerminalGetKey();
        delay(100);                                 // 100 ms. warten
    }
}
}
```

IoT-Brick Library Dokumentation

Bibliotheken

Folgende Bibliotheken müssen im Arduino installiert sein, damit die IoT-Brick Library funktioniert:

```
<ESP8266WiFi.h>
<SSD1306Wire.h>
<TimeLib.h>
<NtpClientLib.h>
"ESP8266WebServer.h"
```

Folgende Bibliotheken werden mitgeliefert und befinden sich im selben Verzeichnis wie die IoT-Brick Library:

```
"all_Type.h"
"all_String.h"
"all_Terminal.h"
```

Globale Konstanten

```
const bool t_ShowMessages             = true;    // Status-Mitteilungen anzeigen
const bool t_HideMessages             = false;   // Status-Mitteilungen nicht anzeigen
const int  IoT_Analog10bit            = 0;       // ADC im ESP8266 hat den Index 0
```

Globale Variablen

Variablen für die Verwendung des im IoT-Brick integrierten OLED-Displays:

```
MCP3421          IoT_ADC18bit          = MCP3421(); // Interner 18 Bit Wandler vom Typ MCP3421
SSD1306Wire      display(0x3c, 4, 5);               // Variable für das OLED Display, wird hier initialisiert
ESP8266WebServer IoT_WebServer(80);                 // Webserver starten auf Port 80
```

Variablen für die Verwendung des IoT-Brick WLANs:

```
bool             IoT_WLANinited        = false;     // Variable für den erfolgreichen (oder nicht) WLAN Connect
```

Variablen für die DHCP-Adresse während des ersten Verbindungsaufbaus des IoT-Brick WLANs:

```
byte             IoT_dynamicIP_address0 = 0;        // 4 Bytes IP-Adresse
byte             IoT_dynamicIP_address1 = 0;
byte             IoT_dynamicIP_address2 = 0;
byte             IoT_dynamicIP_address3 = 0;
byte             IoT_dynamicIP_gateway0 = 0;        // 4 Bytes Gateway
byte             IoT_dynamicIP_gateway1 = 0;
byte             IoT_dynamicIP_gateway2 = 0;
byte             IoT_dynamicIP_gateway3 = 0;
byte             IoT_dynamicIP_subnet0  = 0;        // 4 Bytes Subnetzmaske
byte             IoT_dynamicIP_subnet1  = 0;
byte             IoT_dynamicIP_subnet2  = 0;
byte             IoT_dynamicIP_subnet3  = 0;
```

Variablen für die Verwendung des IoT-Brick WLANs:

```
bool             IoT_OLEDinited        = false;     // Variable für die Initialisierung der OLED-Display Benutzung
```

Variablen für die Verwendung des NTP Time Servers mit dem IoT-Brick:

```
bool             IoT_NTPinited         = false;     // Variable für die Initialisierung der NTP Benutzung
bool             IoT_NTPeventAvail     = false;     // Variable für das Auftreten eines NTP Events
NTPSyncEvent_t   IoT_NTPcurrentEvent;               // Der zuletzt vom TimeServer empfangene Event
```

Variablen für die Größe des Benutzer-EEPROMs im IoT-Brick:

```
long            IoT_EEPROMsize        = 0;        // Größe des verbauten EPROMs
```

Funktionen für die Initialisierung

```
void IoT_Init (Bool IoTbrick)
```

Initialisiert den IoT-Brick für die Verwendung mit der Library. Dieser Befehl muss im `setup()` als erstes aufgerufen werden. Dabei wird die serielle Schnittstelle an Hand der möglichen Geschwindigkeit der verwendeten CPU initialisiert. Die Digitale Leitung 0 für den eingebauten Taster wird als Eingabe definiert. Wenn als Parameter für `IoTbrick` ein `true` angegeben wird, so wird darüber hinaus das aufgesteckte OLED Display initialisiert und anschließend gelöscht sowie für linksbündige 10 Punkt Schrift eingestellt. Die beiden GPIOs 13 und 14 werden als Ausgabe definiert und auf 0V gesetzt. Der integrierte 18 Bit ADC wird inistialisiert. Wenn man einen IoT-Brick benutzt, sollte als Parameter für `IoTbrick` ein `true` angegeben werden. Wenn ein allgemeines EPS82600-Board verwendet wird, sollte ein `false` angegeben werden.

```
bool IoT_NTPinit ()
```

Stellt die Verbindung zum NTP-Timeserver her, installiert einen Event-Handler und übergibt `true` oder `false`, je nachdem, ob eine Verbindung hergestellt werden konnte oder nicht.

Funktionen für die Verwendung von WLAN

```
bool IoT_WLANconnect (String WLANssid, String WLANpassword)
```

Stellt die Verbindung zum WLAN-Hotspot her. Der WLAN-Name und das WLAN-Password werden beim Aufruf der Funktion als `String` angegeben. Die Funktion übergibt `true` oder `false`, je nachdem, ob eine Verbindung hergestellt werden konnte oder nicht. Die globale Variable `IoT_WLANinited` enthält nach dem Aufruf ebenfalls das Funktionsergebnis. Während des Verbindungsaufbaus wird ein Fortschrittsbalken im OLED-Display angezeigt. Gleichzeitig werden Informationen dazu im Terminal ausgegeben. Der Verbindungsaufbau passiert dabei in vier Schritten: Zuerst wird die Verbindung zum WLAN-Hotspot initialisiert und 25 mal versucht, eine Verbindung herzustellen. Ist das nicht erfolgreich, so wird die Verbindung ein zweites mal initialisiert und 25 mal versucht, eine Verbindung herzustellen. Spätestens an diesem Punkt ist in der Regel eine Verbindung zu Stande gekommen. Ist das nicht passiert, so wird die Konfiguration des IoT-Bricks zurückgesetzt und die Verbindung ein drittes mal initialisiert sowie 25 mal versucht, eine Verbindung herzustellen. War

auch das nicht erfolgreich, so wird die Verbindung ein viertes mal initialisiert und 25 mal versucht, eine Verbindung herzustellen. Ist jetzt noch keine Verbindung hergestellt, so kann man davon ausgehen, dass ein Problem vorliegt wie z.B. ein falscher Name für den WLAN-Hotspot, ein falsches Passwort oder eine ungenügende Empfangsstärke. Beim Initialisieren der Verbindung wird zuerst die alte RF-Konfiguration gelöscht (`WiFi.persistent(false)`), danach das WiFi-Modul einmal ausgeschaltet (`WiFi.mode(WIFI_OFF)`) und wieder eingeschaltet (`WiFi.mode(WIFI_STA)`). Erst nach dieser Sequenz wird die Verbindung mit `WiFi.begin` hergestellt. Das Weglassen dieser dreiteiligen Initialisierungssequenz vor dem `WiFi.begin` hat zur Folge, dass eine WLAN-Verbindung nur dann zu Stande kommt, wenn man diese unmittelbar nach dem Neustart des IoT-Bricks herstellt. Sind bereits einige Sekunden vergangen oder möchte man die Verbindung erst zu einem viel späteren Zeitpunkt herstellen, so ist das in der Regel ohne diese Sequenz zu diesem Zeitpunkt gar nicht mehr möglich.

`String IoT_WLANstatus ()`

Ergibt den aktuellen Status der WLAN-Verbindung als `String`. Wenn eine WLAN-Verbindung erfolgreich hergestellt wurde,, so ist der Status gleich „Connected“.

`bool IoT_WLANconnected ()`

Ergibt den aktuellen Status der WLAN-Verbindung als `bool`. Wenn eine WLAN-Verbindung erfolgreich hergestellt wurde,, so ist der Status `true`.

`String IoT_WLANaddress (bool leadingZero)`

Ergibt die aktuelle, lokale WLAN-IP-Adresse des IoT-Bricks als `String` in der Form „000.000.000.000“ (`leadingZero` = `true`) oder in der Form „0.0.0.0“ (`leadingZero` = `false`).

`String IoT_WLANgateway (bool leadingZero)`

Ergibt die aktuelle Gateway-IP-Adresse (Router-IP-Adresse) des IoT-Bricks als `String` in der Form „000.000.000.000“ (`leadingZero` = `true`) oder in der Form „0.0.0.0“ (`leadingZero` = `false`).

`String IoT_WLANsubnet (bool leadingZero)`

Ergibt die aktuelle Subnetz-IP-Adresse des IoT-Bricks als `String` in der Form „000.000.000.000“ (`leadingZero` = `true`) oder in der Form „0.0.0.0“ (`leadingZero` = `false`).

`String IoT_WLANssid ()`

Ergibt den Namen des verbundenen WLAN-Netzwerkes als `String`.

`int IoT_WLANrssi ()`

Ergibt die aktuelle Signalstärke des verbundenen WLAN-Netzwerkes als `int`. Die Einheit ist dBm und der Wert stets eine negative Zahl.

`String IoT_WLANuptime (bool germanLanguage)`

Ergibt die Zeitdauer der bestehenden WLAN-Verbindung in der Form „T days hh:mm:ss“ (`germanLanguage` = `false`) oder in der Form „T Tage hh:mm:ss“ (`germanLanguage` = `true`) als `String`.

`String IoT_WLANstartDatetime ()`

Ergibt die Startzeit für die bestehende WLAN-Verbindung in der Form „tt.mm.jjjj hh:mm:ss“ als `String`.

`void IoT_WLANautoConnect (bool useSavedSettings)`

Stellt eine Verbindung zum zuletzt benutzen WLAN-Hotspot her, wenn für `useSavedSettings` als Wert `true` übergeben wurde. Falls die Verbindung nicht hergestellt werden konnte oder für `useSavedSettings` als Wert `false` übergeben wurde, dann wird der Benutzer durch eine Meldung im OLED-Display dazu aufgefordert, den „IoT-Brick“ Hotspot in den Einstellungen z.B. eines Smartphones auszuwählen. Sobald diese Auswahl erfolgt ist, öffnet sich im Smartphone der Browser und danach kann der gewünschte Hotspot ausgewählt und das Password eingegeben werden. Danach wird das OLED-Display gelöscht und das Programm fortgesetzt.

Funktionen für die Verwendung des Web-Servers (HTTP)

Bevor eine der folgenden Funktionen benutzt werden darf, muss der WebServer zuerst mit `IoT_WebServer.begin()` initialisiert worden sein und es muss zumindest ein Handler für den root-Level installiert sein mit `IoT_WebServer.on("/", handleRoot)`.

`void IoT_WebUpdate (String *html)`

Aktualisiert die Web-Seite, die gerade vom IoT-Brick angezeigt wird. Der Pointer auf die komplette HTML-Seite wird in `*html` übergeben. Der Aufruf der Funktion erfolgt dadurch mit `IoT_WebUpdate(&html)`, d.h. es wird ein Pointer auf den String übergeben, wodurch

keine neue lokale String-Variable auf dem Stack angelegt werden muss. Bei großen HTML-Seiten beschleunigt sich dadurch der Aufruf dieser Funktion erheblich und es wird kein zusätzlicher Speicherplatz benötigt.

```
void IoT_WebPrintAllFields ()
```

Gibt eine Liste mit allen auf der Webseite des IoT-Bricks vorhandenen Feldern aus. Dies umfasst Eingabefelder, angekreuzte Checkboxen und der ausgewählte Button von Radio-Button-Gruppen. Die Liste für das Programm 7 sieht z.B. wie folgt aus:

```
Field #00: vname            = ""
Field #01: nname            = ""
Field #02: unicode          = "chkd"
Field #03: display          = "chkd"
Field #04: LEDgroup         = "none"
```

Checkboxen, die nicht angekreuzt sind, erscheinen auch nicht in dieser Liste ebenso keine Radio-Buttons, die nicht ausgewählt sind. Eingabefelder dagegen erscheinen auch dann in der Liste, wenn sie leer sind. In der ersten Spalte wird die Feldnummer angezeigt, eine Zahl zwischen 0 und 255. In der zweiten Spalte wird der Feldname angezeigt und in der dritten Spalte der Wert bzw. Inhalt des Feldes. Der Inhalt des Feldes ist dabei immer ein String, auch bei Checkboxen und Radio-Buttons. Die folgenden Befehle benutzen den Feldnamen für weitere Operationen. Der Feldname muss eindeutig sein, damit alle Felder benutzt werden können. Die Feldnummer wird vom Webserver erzeugt und stellt keine zuverlässige Identifizierung dar sondern lediglich einen Index in der Gesamtliste, die sich je nachdem, ob Checkboxen angeklickt sind oder nicht, mal kürzer oder mal länger darstellt.

```
bool IoT_WebTestField (String varName)
```

Testet das Vorhandensein eines Feldes mit dem angegebenen Namen. Bei Eingabefeldern ergibt die Funktion `true`, wenn ein Eingabefeld mit dem angegeben Namen vorhanden ist und zwar unabhängig davon, ob in dem Feld etwas eingetragen ist oder nicht. Bei Checkboxen ergibt die Funktion `true`, wenn die Checkbox angekreuzt ist. Bei einem `false` ergeben sich dadurch zwei mögliche Ursachen: Die Checkbox ist nicht angekreuzt oder es existiert kein Feld mit dem angegebene Namen. Bei Radio-Buttons wird kein Feldname im eigentlichen Sinne sondern ein Gruppenname angegeben. In diesem Fall ergibt die Funktion `true`, wenn eine Gruppe mit dem angegeben Namen existiert.

```
String IoT_IoT_WebGetField (String varName)
```

Liest den Wert des Feldes mit dem angegebenen Namen aus. Bei Eingabefeldern ergibt die Funktion den Text, der in das Feld eingetragen ist oder einen leeren String, falls das Feld gar nicht existiert. Dabei ist zu beachten, dass ein leerer String auch bedeuten kann, dass in dem Eingabefeld gar nichts eingetragen ist. Wenn man sicherstellen möchte, dass ein Eingabefeld tatsächlich existiert, muss man zvor mit dem Befehl `IoT_WebTestField` die Existenz des Eingabefeldes prüfen. Bei Checkboxen ergibt die Funktion den Wert der Checkbox, wenn die Checkbox angekreuzt ist oder einen leeren String, wenn diese nicht angekreuzt ist. Bei Radio-Buttons wird kein Feldname im eigentlichen Sinne sondern ein Gruppenname angegeben. In diesem Fall ergibt die Funktion den Wert desjenigen Radio-Buttons, der innerhalb der Gruppe aktuell angekreuzt ist.

Funktionen für die Erzeugung von HTTP-Elementen

Steuerelemente sind Eingabefelder, Form-Buttons, Action-Buttons, Checkboxen und Radio Buttons. Die folgenden Funktionen erzeugen den benötigten HTL_Code der dann als String in die HTML-Seite eingebunden wird. Die Titel von Elementen dürfen Unicode-Zeichen enthalten und werden automatisch in die entsprechenden HTML-Steuerzeichen umgewandelt.

```
String IoT_WebFormOpen (String formName)
```

Erzeugt den HTML-Code zum Anfang eines Web-Formulars. Der Formularname entspricht dem virtuellen Unterverzeichnis, für das man mit dem Befehl `IoT_WebServer.on("/formName", formHandlerFunc);` eine .Verknüpfung zwischen dem Formularnamen/Unterverzeichnis (`formName`) und der Handler-Funktion (`formHandlerFunc`) im `setup()` herstellen muss. Am Ende des Formulars muss dieses mit dem Befehl `IoT_WebFormClose ()` abgeschlossen werden.

```
String IoT_WebFormClose ()
```

Erzeugt den HTML-Code zum Ende eines Web-Formulars, welches mit `IoT_WebFormOpen(String formName)` begonnen wurde.

```
String IoT_WebFormSubmitButton (String formName)
```

Erzeugt den HTML-Code zum Erstellen eines Submit-Buttons für das Web-Formular mit dem Namen `formName`. Wird dieser Submit-Button auf der HTML-Seite angeklickt, führt der IoT-Brick die mit diesem Formular verknüpfte Handler-Funktion aus.

```
String IoT_WebFormActionButton (String actionName, String title)
```

Erzeugt den HTML-Code zum Erstellen eines Action-Buttons mit dem Namen `actionName` und dem angegebenen Titel. Wird dieser Action-Button auf der HTML-Seite angeklickt, führt der IoT-Brick die mit diesem Button verknüpfte Handler-Funktion aus. Dabei wird ein Formular mit dem Namen (`actionName`) angelegt, welches nur einen einzigen Submit-Button mit dem angegeben Titel besitzt. Der Name entspricht dem virtuellen Unterverzeichnis, für das mit dem Befehl `IoT_WebServer.on("/formName", formHandlerFunc);` eine .Verknüpfung zwischen dem Action-Button/Unterverzeichnis (`actionName`) und der Handler-Funktion (`formHandlerFunc`) im `setup()` hergestellt sein muss.

```
String IoT_WebCheckBox (String title, String varName, String varValue, bool checked)
```

Erzeugt den HTML-Code zum Erstellen eines Checkbox-Buttons für das aktuell begonnene Web-Formular. Der Titel erscheint rechts neben der Checkbox. Der Name der Checkbox wird in `varName` übergeben, der alphanumerische Wert der Checkbox in `varValue`. Wenn die Checkbox angekreuzt sein soll, muss im Parameter `checked` ein `true` übergeben werden.

```
String IoT_WebRadioButton (String title, int varIndex, String varName, String varValue,
                           int selectedIndex)
```

Erzeugt den HTML-Code zum Erstellen eines Radio-Buttons für das aktuell begonnene Web-Formular. Der Titel erscheint rechts neben dem Radio-Button. Der Name des Radio-Buttons wird in `varName` übergeben, der Index innerhalb einer Gruppe, die zur selben Variable gehören, wird in `varIndex` übergeben. Bei Radio-Buttons innerhalb derselben Gruppe wird immer derselbe `varName` übergeben. Der alphanumerische Wert des Radio-Buttons mit dem angegebenen Index wird in `varValue` übergeben. Im Parameter `selectedIndex` wird der Index desjenigen Radio-Buttons angegeben, welcher angekreuzt sein soll. Radio-Buttons mit derselben Gruppe, d.h. mit dem selben `varName` haben auch immer den gleichen Wert in `selectedIndex`.

```
String IoT_WebInput (String title, String varName, String varValue, int width)
```

Erzeugt den HTML-Code zum Erstellen eines Eingabefeldes für das aktuell begonnene Web-Formular. Der Titel erscheint links neben dem Eingabefeld. Der Name des Eingabefeldes wird in `varName` übergeben, der alphanumerische Wert, d.h. der sichtbare Inhalt des Eingabefeldes, wird in `varValue` übergeben. Die Breite des Feldes (in Einheiten der Breite einer Zahl) wird in `width` übergeben.

```
String IoT_WebClientIP ()
```

Erzeugt den HTML-Code zur Abfrage der aktuellen Client ID, d.h. der öffentlichen IP-Adresse, unter welcher der Benutzer, der die Webseite des IoT-Bricks aufruft, selbst im Internet erreichbar ist. Dazu bedient sich diese Funktion eines kleinen Java-Scriptes, welches über die Internetseite „l2.io“ die IP-Adresse des Cleint-Browsers bestimmt. Dieser Vorgang dauert beim Aufbau der Internetseite im browser bis zu 500 Millisekunden.

```
String IoT_WebHtab (int h)
```

Erzeugt den HTML-Code für die horizontale Einrückung des nachfolgenden Inhaltes um die angegebene Anzahl von Pixeln.

Funktionen für die Verwendung des OLED-Displays

Bevor eine der folgenden Funktionen benutzt werden darf, muss das OLED-Diplay zuerst mit `IoT_Init()` initialisiert worden sein.

```
void IoT_DisplayFontSize (int fontSize)
```

Legt die Schriftgröße für das OLED-Display fest. Erlaubt sind 10 Punkt (6 Zeilen Text), 16 Punkt (4 Zeilen Text) oder 24 Punkt (2 Zeilen Text). Jede andere Größenangabe setzt die Größe ebenfalls auf 10 Punkt.

```
void IoT_DisplayClear (int fontSize)
```

Löscht das OLED-Display und legt die Schriftgröße fest. Erlaubt sind 10 Punkt (5 Zeilen Text), 16 Punkt (3 Zeilen Text) oder 24 Punkt (2 Zeilen Text). Jede andere Größenangabe setzt die Größe ebenfalls auf 10 Punkt. Die Textausrichting wird auf linksbündig gesetzt. Um die Anzeige sichtbar zu machen, muss wie bei jeder OLED-Display-Ausgabe zum Schluss der Befehl `IoT_DisplayUpdate` aufgerufen werden, damit das OLED-Display aktualisiert wird.

```
void IoT_DisplayUpdate ()
```

Aktualisiert das OLED-Display, d.h. alle Änderungen seit dem letzten Update werden angezeigt. Dadurch ist es möglich, den Bildschirmimhalt flimmerfrei aufzubauen und erst dann anzeigen zu lassen, wenn alles fertig ist.

```
void IoT_DisplayDrawText (int x, int y, String text)
```

Zeigt einen String an der angegebenen Koordinate (x = horizontal, y = vertikal) an. Die angegebene y-Koordinate ist dabei die obere linke Ecke der Schrift, d.h. die erste Zeile beginnt bei 0. Die x-Koordinate ist der Startpunkt für die zuvor gewählte Ausrichtugn des Textes. Um die Anzeige sichtbar zu machen, muss wie bei jeder OLED-Display-Ausgabe zum Schluss der Befehl `IoT_DisplayUpdate` aufgerufen werden, damit das OLED-Display aktualisiert wird.

```
void IoT_DisplayAlignText (OLEDDISPLAY_TEXT_ALIGNMENT alignment)
```

Legt die horizontale Ausrichtung für den nächsten Aufruf von `IoT_DisplayDrawText` fest. Erlaubte Parameter für den Aufruf sind: `TEXT_ALIGN_LEFT` = linksbündige Ausrichtung, `TEXT_ALIGN_CENTER` = zentrierte Ausrichtung und `TEXT_ALIGN_RIGHT` = rechtsbündige Ausrichtung.

```
void IoT_DisplayWLANstatus ()
```

Zeigt den aktuellen WLAN-Status auf dem OLED-Display an. Dabei wird das OLED-Display zuerst gelöscht und dann der WLAN-Name, der Verbindungsstatus, die Feldstärke sowie die IP-Nummern für die lokale Adresse, die Teilnetzmaske sowie das Gateway angezeigt. Um die Anzeige sichtbar zu machen, muss wie bei jeder OLED-Display-Ausgabe zum Schluss der Befehl `IoT_DisplayUpdate()` aufgerufen werden, damit das OLED-Display aktualisiert und der Inhalt angezeigt wird.

Funktionen für die Verwendung des NTP Timeservers

Bevor eine der folgenden Funktionen benutzt werden darf, muss der NTP Timeserver zuerst mit `IoT_NTPinit()` initialisiert worden sein.

```
String IoT_NTPtime ()
```

Ergibt die aktuelle Zeit in der Form „hh:mm:ss“ als `String`.

```
String IoT_NTPdate ()
```

Ergibt das aktuelle Datum in der Form „tt.mm.jjjj“ als `String`.

`String IoT_NTPdatetime ()`

Ergibt das aktuelle Datum und die aktuelle Zeit in der Form „tt.mm.jjjj hh:mm:ss“ als `String`.

`bool IoT_NTPvalid ()`

Ergibt `true`, falls eine gültige Uhrzeit vom NTP Timeserver empfangen wurde, sonst `false`.

`String IoT_NTPdaylightSavingTime (bool germanLanguage)`

Ergibt den aktuellen Staus von Sommer- oder Winterzeit als `String`. Dabei wird „Summer Time“ oder „Winter Time“ (`germanLanguage` = `false`) bzw. „Sommerzeit“ oder „Winterzeit“ (`germanLanguage` = `true`) übergeben.

`void IoT_NTPprintEvent (NTPSyncEvent_t event)`

Gibt den aktuellen Status der NTP Timeservers und die Uhrzeit (wenn eine Uhrzeit empfangen wurde) auf dem Terminal aus.

Allgemeine Funktionen zur Steuerung und Verwaltung des IoT-Bricks

`void IoT_Idle ()`

Diese Funktion verzögert den Programmablauf um 1 Millisekunde. Dadurch wird verhindert, dass bei komplexen Prozessen der im IoT-Brick integrierte Watch-Dog der Meinung ist, das Programm habe sich aufgehangen und einen reset des IoT-Bricks auslöst. Komplexe Funktionen innerhalb der Library rufen diese Funktion bereits auf. Bei Schleifen, die mehrere Sekunden dauern, sollte man diese Funktion immer zwischendurch aufrufen.

`void IoT_WatchDog (bool active)`

Schaltet den IoT-Brick-Watchdog ein (`true`) oder aus (`false`). Der IoT-Brick-Watchdog überwacht, ob sich das aktuell laufende Programm aufgehangen hat oder in einer Endlosschleife befindet. In diesen Zuständen wird er neu gestartet. Das hat in einigen Fällen zur Folge, dass Rechenoperationen, die länger als eine Sekunde dauern, zum Reset des IoT-Bricks führen, wenn diese keine `IoT_Idle()`oder `delay()` Aufrufe enthalten. Für Benchmarks oder andere Fälle, in denen Rechenoerationen mehr als 1 Sekunde dauern, sollte man den IoT-Brick-Watchdog vorher ausschalten.

```
bool IoT_Keypress ()
```

Ergibt `true`, falls der im IoT-Brick eingebaute Taster aktuell gedrückt wird, sonst `false`.

```
void IoT_WaitKeypress (uint64 timeout, bool message)
```

Wenn `message` = `true` ist, wird `IoT_WaitMessage()` aufgerufen und im OLED_Display eine Mitteilung angezeigt, dass der Benutzer den im IoT-Brick eingebauten Taster drücken soll. Die Funktion wartet dabei die mit `timeout` angegebene Anzahl von Millisekunden oder bis der Benutzer den Taster auf dem IoT-Brick gedrückt hat. Wird eine Zeit von 0 Millisekunden angegeben, so wartet der Befehl, bis der Taster tatsächlich vom Benutzer gedrückt wurde, ohne die Berücksichtigung eines Timeouts. Danach wird das OLED-Display gelöscht, Wenn `message` = `true` ist. Danach wird das Programm fortgesetzt.

```
void IoT_WaitMessage ()
```

Zeigt im OLED_Display eine Mitteilung an, dass der Benutzer den im IoT-Brick eingebauten Taster drücken soll um fortzufahren.

```
void IoT_WaitNoKeypress ()
```

Wartet, bis der im IoT-Brick eingebauten Taster losgelassen wurde.

```
void IoT_EEPROMclear ()
```

Löscht alle Daten, die in sich im EEPROM befinden und schreibt in alle Speicherzellen des EEPROMs Nullen. Um diesen Löschvorgang auch nach einem Reset oder Ausschalten zu erhalten, muss vorher der Befehl `IoT_EEPROMupdate()` oder wahlweise der Befehl `IoT_ShutDown()` benutzt werden. Die Konfiguration des WiFi-Managers wird hiervon nicht berührt, d.h. die Einwahldaten für das WLAN befinden sich nicht in dem hier zugänglichen EEPROM.

```
void IoT_EEPROMupdate ()
```

Aktualisiert alle Daten, die in das EEPROM geschrieben wurden. Wenn dieser Befehl nicht verwendet wurde, oder wahlweise der Befehl `IoT_ShutDown()`, dann gehen bei einem Reset oder Ausschalten des EPS8266 alle seit dem letzten `IoT_EEPROMupdate()` in das EEPROM geschriebenen Daten verloren.

```
void IoT_TerminalWaitInit (bool showMessage)
```

Diese Funktion wartet so lange, bis mit einem Terminalprogramm eine Verbindung über USB mit dem IoT-Brick hergestellt wurde. Wenn man als Parameter `t_ShowMessages` angibt, so wird vorher das OLED-Display gelöscht und die Meldung „Bitte mit Terminal verbinden“ angezeigt. Sobald eine Verbindung hergestellt wurde, wird das OLED-Display erneut gelöscht und die Meldung „Verbindung ist hergestellt“ sowie das verwendete Protokoll und die Terminal-Bildschirmgröße in Breite x Höhe angegeben.

Die Übertragung des tatsächlich verwendeten Protokolls sowie der tatsächlichen Bildschirmgröße erfolgt nur bei einer Verbindung mit dem Programm UniTerminal, welches aktuell nur für macOS verfügbar ist.

Bei anderen Terminalprogrammen wie dem Arduino IDE Seriellen Monitor oder Zterm wird bei Eingabe von "`ttttt`" das TTY-Protokoll angenommen sowie bei Eingabe von "`vvvvv`" das DEC VT52 Protokoll und die Bildschirmgröße auf 80 x 24 Zeichen gesetzt.

```
void IoT_ShutDown ()
```

Schaltet das Display des IoT-Bricks aus und führt eine Endlosschleife aus, ohne dass es zu einem Watch-Dog-bedingten Reset des IoT-Bricks kommt. Der Brick kann jederzeit durch Drücken der Reset-Taste oder über das Terminalprogramm neu gestartet werden. Alle Daten, die in das EEPROM geschrieben wurden, werden vorher aktualisiert.

String Library Listing

Die String-Library stellt alle möglichen String-Befehle zur Verfügung, die das Arbeiten mit Strings erheblich leichter und übersichtlicher macht.

```
// String Library
// 1.00 - 2017-05-17
// 1.01 - 2017-06-23
// 1.02 - 2017-07-18
// (c) Copyright 2017
//
// Zur Verwendung mit allen Allnet Libraries


// *** Globale Variablen ***

int64  rnd_0 = 0;        // Die zuletzt erzeugte ganzzahlige Zufallszahl
double rndreal_0 = 0.0;  // Die zuletzt erzeugte Fließkomma Zufallszahl


// *** Math-Funktionen ***

double minimum (double x, double y)
{
    if (x < y)
    {
        return x;
    }
    else
    {
        return y;
    }
}

double maximum (double x, double y)
{
    if (x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}
```

```
bool odd (long n)
{
    return ((n & 0x01) == 1);
}

bool even (long n)
{
    return ((n & 0x01) == 0);
}

byte sgn (double n)
{
    if (n > 0)
    {
        return 1;
    }
    else if (n < 0)
    {
        return -1;
    }
    else
    {
        return 0;
    }
}

byte boolval (bool b)
{
    if (b)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int64 exponent2 (byte n)
{
    if ((n < 0) || (n > 63))
    {
        return 0;
    }
    else
    {
        int64 i = (int64)1 << n;
        return (i);                 // exp2(63) = -9223372036854775808 wegen Vorzeichen
    }
```

```
}

int64 exponent10 (byte n)
{
    if ((n < 0) || (n > 18))
    {
        return 0;
    }
    else
    {
        long result = 1;
        for (byte i = 1; i <= n; i++)
        {
            result *= 10;
        }
        return result;
    }
}

double frac (double n)
{
    return n - trunc(n);
}


// *** Mathematische Funktionen für hexadezimale und BCD-Verarbeitung von Zahlen ***

byte lobyte (uint64 n)
{
    return (n & 0xFF);
}

byte hibyte (uint64 n)
{
    return ((n & 0xFF00) >> 8);
}

uint16 loword (uint64 n)
{
    return (n & 0xFFFF);
}

uint16 hiword (uint64 n)
{
    return (n >> 16);
}

byte bcd2bin (byte val)
{
    return val - 6 * (val >> 4);
```

```
}

byte bin2bcd (byte val)
{
    return val + 6 * (val / 10);
}


// *** Zufallszahlen- & Kryptographie-Funktionen ***

int64 crc64 (int64 Old64BitCRC, byte AddThis)
{
    int64 hash = Old64BitCRC;
    int64 crc = hash;
    for (int i = 1; i <= 8; i++)
    {
        if (hash < 0) // 0001 0000 0010 0001 0001 0000 0001 0001 0001 0000 0010 0001 0001 0000 0000 0001
        {             // = 116.222.784.430.938.931 = 0x1021101110211001 (Primzahl)
            crc = (0x1021101110211001 ^ (crc << 1));
        }
        else
        {
            crc <<= 1;
        }
        hash = (int64(AddThis) ^ crc);
    }
    return (hash);
}

int64 crc64 (String s, int64 hash)
{
    int L = s.length();
    if (L > 0)
    {
        for (int i = 0; i < L; i++)
        {
            hash = crc64(hash, s[i] & 0xFF);
        }
    }
    return (hash);
}

uint64 xorshift128plus (uint64 seed0, uint64 seed1)
{
    uint64 state0 = seed0;
    uint64 state1 = seed1;
    if ((state0 == 0) && (state1 == 0))
    {
        state0 = 1; // Beide Statusvariablen dürfen nicht 0 sein
    }
```

```
    uint64 s1 = state0;
    uint64 s0 = state1;
    state0 = s0;
    s1 ^= s1 << 23;
    s1 ^= s1 >> 17;
    s1 ^= s0;
    s1 ^= s0 >> 26;
    state1 = s1;
    return ((state0 & 0x7FFFFFFFFFFFFFFF) + (state1 & 0x7FFFFFFFFFFFFFFF));
}

uint64 xorshift128plus (String s, uint64 hash)
{
    int L = s.length();
    if (L > 0)
    {
        for (int i = 0; i < L; i++)
        {
            hash = xorshift128plus(hash, s[i]);
        }
    }
    return (hash);
}

byte rndbyte () // Statistisch und absolut gesehen eine richtige Zufallszahl zwischen 0 und 255
{
    int count = 0;
    long a = 0;
    long b = 0;
    while (((a == 0) || (b == 0)) && (count < 100))
    {
        a = random(65536) & 65535; // Es werden mit random() bei jedem Programmstart die gleichen Zufallszahlen erzeugt
        b = micros() & 65535;
        count += 1;
    }
    return (hiword(a * b) & 0xFF);
}

int64 rnd (int64 n) // Zufallszahl zwischen 0 und n, inclusive 0 und n
{
    if (n != 0)
    {
        int64 zahl = 0;
        for (byte i = 0; i <= 7; i++)
        {
            if (i != 7)
            {
                zahl <<= 8;
                zahl |= rndbyte();
            }
```

```
            else
            {
                zahl <<= 7;
                zahl |= (rndbyte() & 0x7F);
            }
        }
        rnd_0 = zahl % (n + 1);
    }
    return (rnd_0);
}

byte rndbytesecure () // Statistisch und absolut gesehen eine richtige Zufallszahl, wesentlich sicherer als rndbyte
{
    uint64 urand1 = (uint64(rnd(0xFFFFFFFF)) << 32) | uint64(rnd(0xFFFFFFFF));
    uint64 urand2 = (uint64(rnd(0xFFFFFFFF)) << 32) | uint64(rnd(0xFFFFFFFF));
    uint64 rand64 = xorshift128plus(urand1, urand2);
    return byte(rand64 & 0xFF);
}

double rndreal (double n) // Zufallszahl zwischen 0.0 und n, inclusive 0.0 und n
{
    if (n != 0.0)
    {
        rndreal_0 = double(rnd(9999999999)) / 9999999999.0 * n;
    }
    return (rndreal_0);
}


// *** String-Funktionen ***

String chr (int c)
{
    String s = "";
    s += char(c);
    return s;
}

int asc (String s)
{
    if (s == "")
    {
        return 0;
    }
    else
    {
        return s.charAt(0);
    }
}
```

```
int len (String s)
{
    return s.length();
}

bool number (byte n)
{
    return ((n >= 48) && (n <= 57)); // "0" bis "9"
}

bool letter (byte n)
{
    return (((n >= 65) && (n <= 90)) || ((n >= 97) && (n <= 122))); // "A" bis "Z" und "a" bis "z"
}

String spc (int length)
{
    String s = "";
    if (length > 0)
    {
        if (length > 255)
        {
            length = 255;
        }
        for (int i = 1; i <= length; i++)
        {
            s = s + " ";
        }
    }
    return s;
}

String ucase (String s)
{
    s.toUpperCase();
    return s;
}

String lcase (String s)
{
    s.toLowerCase();
    return s;
}

String boolstr (bool b, String caseTrue, String caseFalse)
{
    if (b)
    {
        return (caseTrue);
    }
```

```
    else
    {
        return (caseFalse);
    }
}

String left (String s, int length)
{
    if (length < 1)
    {
        return "";
    }
    else if (length >= s.length())
    {
        return s;
    }
    else
    {
        return s.substring(0, length);
    }
}

String part (String s, int start)
{
    if (start <= 1)
    {
        return s;
    }
    else if (start > s.length())
    {
        return "";
    }
    else
    {
        return (s.substring(start - 1));
    }
}

String right (String s, int length)
{
    int L = s.length();
    if (L <= length)
    {
        return s;
    }
    else if (length < 1)
    {
        return "";
    }
    else
```

```
    {
        return s.substring(L - length, L);
    }
}

String mid (String s, int start, int length)
{
    int L = s.length();
    if (start <= 1)
    {
        return left(s, length);
    }
    else if ((start > L) || (length < 1))
    {
        return "";
    }
    else
    {
        if (length >= (L - start + 1))
        {
            return s.substring(start - 1);
        }
        else
        {
            return s.substring(start - 1, start + length - 1);
        }
    }
}

String replace (String s, String oldstr, String newstr)
{
    s.replace(oldstr, newstr);
    return s;
}

int position (String s, String t, int start)
{
    if (start <= 1)
    {
        return s.indexOf(t) + 1;
    }
    else
    {
        return s.indexOf(t, start - 1) + 1;
    }
}

String indchar (String s, int i)
{
    if ((i >= 1) && (i <= s.length()))
```

```
    {
        return chr(s[i - 1]);
    }
    else
    {
        return ("");
    }
}

int indasc (String s, int i)
{
    if ((i >= 1) && (i <= s.length()))
    {
        return (s[i - 1]);
    }
    else
    {
        return (0);
    }
}

String deleteasc (String s, int p)
{
    return (left(s, p - 1) + part(s, p + 1));
}

String repeatstr (String s, byte n)
{
    String r = "";
    if (n > 0)
    {
        for (int i = 1; i <= n; i++)
        {
            r = r + s;
        }
    }
    return (r);
}

String repeatasc (int a, int n)
{
    String r = "";
    String zchn = chr(a);
    if (n > 0)
    {
        for (int i = 1; i <= n; i++)
        {
            r = r + zchn;
        }
    }
```

```
    return (r);
}

String strform (int64 n, int asc, byte MinVorkomma, bool TausenderPunkt)
{
    String vorzeichen = "";
    if (n < 0)
    {
        vorzeichen = "-";
        //n = abs(n);
    }
    String vorkomma = "";
    if (n == 0)
    {
        vorkomma = "0";
    }
    else
    {
        while (n !=  0)
        {
            vorkomma = chr(abs(n % 10) + 48) + vorkomma;
            n = n / 10;
        }
    }
    if (TausenderPunkt)
    {
        String s = "";
        byte l = vorkomma.length();
        for (int i = 0; i < l; i++)
        {
            if ((((l - i) % 3) == 0) && (s != ""))
            {
                s = s + chr(46); // "."
            }
            s = s + indchar(vorkomma, i + 1);
        }
        vorkomma = s;
    }
    vorkomma = vorzeichen + vorkomma;
    byte VL = vorkomma.length();
    if (MinVorkomma > VL)
    {
        vorkomma = repeatasc(asc, MinVorkomma - VL) + vorkomma;
    }
    return (vorkomma);
}

String str (int64 n)
{
    return strform(n, 32, 1, true);
```

```
}

String strrealform (double n, byte asc, byte MinVorkomma, byte MaxNachkomma, bool TausenderPunkt, bool CutZero)
{
    String vorzeichen = "";
    if (n < 0) // Negativ
    {
        vorzeichen = "-";
        n = abs(n);
    }
    if (MaxNachkomma > 9) // float ist auf 7 Stellen genau, Double auf 9 Stellen
    {
        MaxNachkomma = 9;
    }
    n = n + (1.0 / exp10(MaxNachkomma) / 2.0); // Runden
    String result = "";
    long mantissa = n;
    String vorkomma = String(mantissa);
    if (TausenderPunkt)
    {
        String s = "";
        byte l = vorkomma.length();
        for (int i = 0; i < l; i++)
        {
            if ((((l - i) % 3) == 0) && (s != ""))
            {
                s = s + chr(46); // "."
            }
            s = s + indchar(vorkomma, i + 1);
        }
        vorkomma = s;
    }
    vorkomma = vorzeichen + vorkomma;
    byte VL = vorkomma.length();
    if (MinVorkomma > VL)
    {
        vorkomma = repeatasc(asc, MinVorkomma - VL) + vorkomma;
    }
    double nachkomma = frac(n);
    if ((nachkomma != 0) || not(CutZero))
    {
        vorkomma = vorkomma + ",";
        for (byte i = 1; i <= MaxNachkomma; ++i)
        {
            nachkomma = nachkomma * 10;
            vorkomma = vorkomma + String((long)nachkomma);
            nachkomma = nachkomma - trunc(nachkomma);
        }
        if (CutZero)
        {
```

```
            while (right(vorkomma, 1) == "0")
            {
                vorkomma = left(vorkomma, vorkomma.length() - 1);
            }
        }
        if (right(vorkomma, 1) == ",")
        {
            vorkomma = left(vorkomma, vorkomma.length() - 1);
        }
    }
    return vorkomma;
}

String strrealform (double n, byte asc, byte MinVorkomma, byte MaxNachkomma, bool TausenderPunkt, bool CutZero, bool ShowPlus)
{
    String s = strrealform(n, asc, MinVorkomma, MaxNachkomma, TausenderPunkt, CutZero);
    if ((n > 0) && ShowPlus)
    {
        s = "+" + s;
    }
    return s;
}

String strreal (double n, int Nachkomma)
{
    bool CutZero = (Nachkomma == 0);
    if ((Nachkomma < 1) || (Nachkomma > 9))
    {
        Nachkomma = 9;
    }
    return strrealform(n, 32, 1, Nachkomma, true, CutZero);
}

String strIP (IPAddress ip, bool leadingZero)
{
    if (leadingZero)
    {
        return (strform(ip[0], 48, 3, false) + "." + strform(ip[1], 48, 3, false) + "." +
                strform(ip[2], 48, 3, false) + "." + strform(ip[3], 48, 3, false));
    }
    else
    {
        return (str(ip[0]) + "." + str(ip[1]) + "." + str(ip[2]) + "." + str(ip[3]));
    }
}

int64 val (String s)
{
    bool vorzeichen = false;
    if (s == "")
```

```
    {
        return 0;
    }
    else
    {
        byte p = 0;
        byte L = s.length();
        int64 result = 0;
        if (s[0] == 45) // "-"
        {
            vorzeichen = true;
            p = 1;
        }
        while (p < L)
        {
            byte ziffer = s[p];
            ++p;
            if ((ziffer >= 48) && (ziffer <= 97)) // "0" bis "9"
            {
                result = (result * 10) + ziffer - 48;
            }
            else
            {
                if (ziffer != 46) // Tausenderpunkt ignorieren
                {
                    p = L; // Abbruch bei illegalen Zeichen
                }
            }
        }
        if (vorzeichen)
        {
            return -result;
        }
        else
        {
            return result;
        }
    }
}

double realval (String s)
{
    s = replace(s, ",", ".");
    return s.toFloat();
}

IPAddress IPval (String s)
{
    int v1, v2, v3, v4;
    int p = 1;
```

```
    p = position(s, ".", 1);
    if (p > 1)
    {
        v1 = val(left(s, p - 1));
        s = part(s, p + 1);
    }
    else
    {
        return(IPAddress(0, 0, 0, 0));                          // Fehlerhafte IP-Adresse
    }
    p = position(s, ".", 1);
    if (p > 1)
    {
        v2 = val(left(s, p - 1));
        s = part(s, p + 1);
    }
    else
    {
        return(IPAddress(0, 0, 0, 0));                          // Fehlerhafte IP-Adresse
    }
    p = position(s, ".", 1);
    if (p > 1)
    {
        v3 = val(left(s, p - 1));
        s = part(s, p + 1);
    }
    else
    {
        return(IPAddress(0, 0, 0, 0));                          // Fehlerhafte IP-Adresse
    }
    v4 = val(s);
    if ((v1 >= 0) && (v1 <= 255) && (v2 >= 0) && (v2 <= 255) &&
        (v3 >= 0) && (v3 <= 255) && (v4 >= 0) && (v4 <= 255))
    {
        return(IPAddress(v1, v2, v3, v4));                      // Gültige IP-Adresse
    }
    else
    {
        return(IPAddress(0, 0, 0, 0));                          // Fehlerhafte IP-Adresse
    }
}

String fillcenter (String s, String fill, int width)
{
    int L = s.length();
    if (L < width)
    {
        s = repeatstr(fill, (width - L) / 2) + s;
        s += repeatstr(fill, width - L - (width - L) / 2);
    }
```

```
    return (s);
}


// *** Hexadezimal-Funktionen ***

bool hexnumber (byte n)
{   // "0" bis "9" und "A" bis "F" und "a" bis "f"
    return (((n >= 48) && (n <= 57)) || (((n >= 65) && (n <= 70)) || ((n >= 97) && (n <= 102))));
}

String hex (uint64 v)
{
    String hexStr = "";
    if (v == 0)
    {
        hexStr = "0";
    }
    else
    {
        while (v > 0)
        {
            byte digit = v & 0xF;
            v = v / 16;
            if (digit < 10)
            {
                hexStr = chr(digit + 48) + hexStr;
            }
            else
            {
                hexStr = chr(digit + 55) + hexStr;
            }
        }
    }
    return hexStr;
}

String hexbyte (byte v)
{
    return right("0" + hex(v), 2);
}

String hexword (uint16 v)
{
    return right("000" + hex(v), 4);
}

String hexlong (ulong v)
{
    return right("0000000" + hex(v), 8);
```

```
}

String hexint64 (uint64 v)
{
    return right("00000000000000" + hex(v), 16);
}

uint64 dec (String s)
{
    uint64 v = 0;
    byte l = s.length();
    if (l > 0)
    {
        for (int i = 0; i < l; i++)
        {
            byte b = s[i];
            if ((b >= 48) && (b <= 57))
            {
                v = (v * 16) + b - 48;
            }
            else if ((b >= 65) && (b <= 70)) // Großbuchstaben
            {
                v = (v * 16) + b - 55;
            }
            else if ((b >= 97) && (b <=102)) // Kleinbuchstaben
            {
                v = (v * 16) + b - 87;
            }
            else
            {
                i = l;
            }
        }
    }
    return v;
}

String newuniqueid ()
{
    String ID = "";
    for (byte i = 0; i <= 15; i++)
    {
        ID += hexbyte(rndbyte());
    }
    return (ID);
}


// *** Konvertierungs-Funktionen ***
```

```
String cleanasc (String s)
{
    String t = "";
    int L = s.length();
    if (L > 0)
    {
        for (int i = 0; i < L; i++)
        {
            int c = s.charAt(i);                                    // Character Code an der Position
            switch (c)
            {
                case 228: // "ä"
                    t += "ae";
                    break;
                case 246: // "ö"
                    t += "oe";
                    break;
                case 252: // "ü"
                    t += "ue";
                    break;
                case 223: // "ß"
                    t += "ss";
                    break;
                case 196: // "Ä"
                    t += "Aae";
                    break;
                case 214: // "Ö"
                    t += "Oe";
                    break;
                case 220: // "Ü"
                    t += "Ue";
                    break;
                case 128: // "€"
                    t += "EUR";
                    break;
                case 149: // "•"
                    t += ".";
                    break;
                case 160: // "␣"
                    t += " ";
                    break;
                case 162: // "¢"
                    t += "Cent";
                    break;
                case 163: // "£"
                    t += "Pfund";
                    break;
                case 165: // "¥"
                    t += "Yen";
                    break;
```

```
case 166: // "¦"
    t += "|";
    break;
case 167: // "§"
    t += "Paragraph";
    break;
case 169: // "©"
    t += "(c)";
    break;
case 171: // "«"
    t += "<<";
    break;
case 174: // "®"
    t += "(R)";
    break;
case 176: // "°"
    t += "Grad";
    break;
case 180: // "´"
    t += "'";
    break;
case 181: // "µ"
    t += "u";
    break;
case 183: // "·"
    t += ".";
    break;
case 187: // "»"
    t += ">>";
    break;
case 188: // "¼"
    t += "1/4";
    break;
case 189: // "½"
    t += "1/2";
    break;
case 190: // "¾"
    t += "3/4";
    break;
case 198: // "Æ"
    t += "AE";
    break;
case 215: // "×"
    t += "x";
    break;
case 230: // "æ"
    t += "ae";
    break;
case 8211: // "–"
    t += "-";
```

```
                    break;
                case 8212: // "—"
                    t += "—";
                    break;
                default:
                    if (c < 32)
                    {
                        t += "Ctrl-" + chr(c + 64);
                    }
                    else if (c < 128)
                    {
                        t += chr(c);
                    }
                    else
                    {
                        t += "{" + str(c) + "}";
                    }
                    break;
            }
        }
    }
    return (t);
}

String cleanhtml (String s)
{
    s.replace(" ", " ");
    s.replace("'", "&apos;");
    s.replace("ä", "&auml;");
    s.replace("ö", "&ouml;");
    s.replace("ü", "&uuml;");
    s.replace("ß", "&szlig;");
    s.replace("Ä", "&Auml;");
    s.replace("Ö", "&Ouml;");
    s.replace("Ü", "&Uuml;");
    s.replace("€", "&euro;");
    s.replace("•", "&bull;");
    s.replace(" ", " ");
    s.replace("¢", "&cent;");
    s.replace("£", "&pound;");
    s.replace("¥", "&yen;");
    s.replace("¦", "&sect;");
    s.replace("§", "&xxx;");
    s.replace("©", "&copy;");
    s.replace("«", "&laquo;");
    s.replace("®", "&reg;");
    s.replace("°", "&deg;");
    s.replace("´", "&acute;");
    s.replace("µ", "&micro;");
    s.replace("·", "&middot;");
```

```
    s.replace("»", "&raquo;");
    s.replace("¼", "&frac14;");
    s.replace("½", "&frac12;");
    s.replace("¾", "&frac34;");
    s.replace("Æ", "&AElig;");
    s.replace("×", "&times;");
    s.replace("æ", "&aelig;");
    s.replace("–", "&ndash;");
    s.replace("—", "&mdash;");
    return (s);
}

String convertescape (String s)
{
    String result = s;
    int p = 1;
    while (p > 0)
    {
        p = position(result, chr(92), p);
        if (p > 0)
        {
            if (indasc(result, p + 1) == 117)   // "u"
            {
                String h = mid(result, p + 2, 4); // 4 Zeichen Hex-Code
                int asc = dec(h);                 // Entsprechender Caracter-Code
                result = left(result, p - 1) + chr(asc) + part(result, p + 6);
            }
            else
            {
                p += 1;                           // Einzelnen Backslash ignorieren
            }
        }
    }
    return (result);
}

String quotate (String s)
{
    return chr(34) + s + chr(34);
}


// *** Datum & Uhrzeit Funktionen ***


byte monthdays (byte month, int year) // Tage im angegebenen Monat
{
    byte result = 0;
    if ((month >= 1) && (month <= 12) && (year >= 1))
    {
```

```
        result = 31 - boolval(month == 9 || month == 4 || month == 6 || month == 11) - 3 * boolval(month == 2);
        result = result + boolval(((year % 4) == 0) && (((year % 100) != 0) || ((year % 400) == 0)) && (month == 2));
    }
    return result;
}

long timetoseconds (int days, byte hour, byte minute, byte second)
{
    return (long)86400 * days + (long)3600 * hour + (long)60 * minute + (long)second;
}

long datetodays2000 (int year, byte month, byte day) // Tage seit dem 01.01.2000
{
    year = year - 2000;
    long days = day;
    for (byte i = 1; i < month; ++i)
    {
        days = days + monthdays(i, year);
    }
    if (month > 2 && year % 4 == 0)
    {
        ++days;
    }
    return days + 365 * year + (year + 3) / 4 - 1;
}

byte dayofweek (int year, byte month, byte day) // Wochentag: 0 = Sonntag, 1 = Montag usw.
{
    int K = int(0.6 + (1.0 / double(month)));
    int L = year - K;
    int Z = int(13 * (12 * K + (int)month + 1) / 5) + int(double(L) * 1.25) - int(L / 100) + int(L / 400) + day - 1;
    return (Z % 7);
}

String dayname (int day)
{
    switch (day)
    {
        case 0:
            return ("Sonntag");
            break;
        case 1:
            return ("Montag");
            break;
        case 2:
            return ("Dienstag");
            break;
        case 3:
            return ("Mittwoch");
            break;
```

```
        case 4:
            return ("Donnerstag");
            break;
        case 5:
            return ("Freitag");
            break;
        case 6:
            return ("Samstag");
            break;
        case 7:
            return ("Sonntag");
            break;
        default:
            return ("");
            break;
    }
}

String monthname (int month)
{
    switch (month)
    {
        case 1:
            return ("Januar");
            break;
        case 2:
            return ("Februar");
            break;
        case 3:
            return ("März");
            break;
        case 4:
            return ("April");
            break;
        case 5:
            return ("Mai");
            break;
        case 6:
            return ("Juni");
            break;
        case 7:
            return ("Juli");
            break;
        case 8:
            return ("August");
            break;
        case 9:
            return ("September");
            break;
        case 10:
```

```
            return ("Oktober");
            break;
        case 11:
            return ("November");
            break;
        case 12:
            return ("Dezember");
            break;
        default:
            return ("");
            break;
    }
}

byte monthnumber (String month)
{
    byte result = 0;
    month = ucase(month) + "   ";
    byte c0 = month[0];
    byte c1 = month[1];
    byte c2 = month[2];
    if (c0 == 74) // "J"
    {
        if (c1 == 65) // "A"
        {
            result = 1; // Jan
        }
        else if (c2 == 78) // "N"
        {
            result = 6; // Jun
        }
        else if (c2 == 76) // "L"
        {
            result = 7; // Jul
        }
    }
    else if (c0 == 70) // "F"
    {
        result = 2; // Feb
    }
    else if (c0 == 77) // "M"
    {
        if ((c2 == 73) || (c2 == 89)) // "I" oder "Y"
        {
            result = 5; // Mai, May
        }
        else
        {
            result = 3; // Mär, Mar
        }
```

```
    }
    else if (c0 == 65) // "A"
    {
        if (c1 == 80) // "P"
        {
            result = 4; // Apr
        }
        else
        {
            result = 8; // Aug
        }
    }
    else if (c0 == 83) // "S"
    {
        result = 9; // Sep
    }
    else if (c0 == 79) // "O"
    {
        result = 10; // Okt, Oct
    }
    else if (c0 == 78) // "N"
    {
        result = 11; // Nov
    }
    else if (c0 == 68) // "D"
    {
        result = 12; // Dez, Dec
    }
    return result;
}

String datestr (t_DateTime DateTime)
{
    byte day = DateTime.day;
    byte month = DateTime.month;
    int year = DateTime.year;
    if ((day >= 1) && (day <= monthdays(month, year))) // Day is valid
    {
        return right("0" + String(day), 2) + "."  + right("0" + String(month), 2) + "." + String(year);
    }
    else
    {
        return "";
    }
}

String timestr (t_DateTime DateTime)
{
    byte hour = DateTime.hour;
    byte minute = DateTime.minute;
```

```
    byte second = DateTime.second;
    if ((hour >= 0) && (hour <= 23) && (minute >= 0) && (minute <= 59) && (second >= 0) && (second <= 59)) // Time is valid
    {
        return right("0" + String(hour), 2) + ":"  + right("0" + String(minute), 2) + ":" + right("0" + String(second), 2);
    }
    else
    {
        return "";
    }
}
```

String Library Dokumentation

Die String-Library stellt alle möglichen String-Befehle zur Verfügung, die das Arbeiten mit Strings erheblich leichter und übersichtlicher macht.

Mathematische Funktionen

```
double minimum (double x, double y)
```

Ergibt die kleinere Zahl der beiden angegebene Zahlen.

```
double maximum (double x, double y)
```

Ergibt die größere Zahl der beiden angegebene Zahlen.

```
bool odd (long n)
```

Ergibt `true`, wenn es sich um eine ungerade Zahl handelt.

```
bool even (long n)
```

Ergibt `true`, wenn es sich um eine gerade Zahl handelt.

```
byte sgn (double n)
```

Ergibt -1 für alle negativen Zahlen, 0 für eine Null sowie 1 für alle positiven Zahlen.

```
byte boolval (bool b)
```

Ergibt 1 für `true` und 0 für `false`.

```
int64 exponent2 (byte n)
```

Ergibt den Exponentialwert zur Basis 2 (2^n) des angegebenen Exponents n. Dabei sind alle Zahlen zwischen 0 und 63 für n erlaubt. Bitte beachten Sie, dass 2^{63} als -9223372036854775808 zurückgegeben wird, da das 63. Bit das Vorzeichen repräsentiert.

```
int64 exponent10 (byte n)
```

Ergibt den Exponentialwert zur Basis 10 (10^n) des angegebenen Exponents n. Dabei sind alle Zahlen zwischen 0 und 18 für n erlaubt.

```
double frac (double n)
```

Ergibt den Nachkommateil des angegebenen Fließkomma-Wertes n.

Mathematische Funktionen für hexadezimale und BCD-Verarbeitung von Zahlen

```
byte lobyte (uint64 n)
```

Ergibt die unteren 8 Bit angegebenen Wertes n.

```
byte hibyte (uint64 n)
```

Ergibt die oberen 8 Bit von den unteren 16 Bit des angegebenen Wertes n.

```
uint16 loword (uint64 n)
```

Ergibt die unteren 16 Bit angegebenen Wertes n.

```
uint16 hiword (uint64 n)
```

Ergibt die oberen 16 Bit angegebenen Wertes n.

```
byte bcd2bin (byte val)
```

Wandelt die bcd-Zahl n in eine Binärzahl um. Wird bei der Verarbeitung von i2c-Daten für einige Sensoren benötigt.

```
byte bin2bcd (byte val)
```

Wandelt die Binärzahl n in eine bcd-Zahl um. Wird bei der Verarbeitung von i2c-Daten für einige Sensoren benötigt.

Kryptographie Funktionen

```
int64 crc64 (int64 Old64BitCRC, byte AddThis)
```

Berechnet den (inkrementellen) crc64-Hash aus dem vorhergehenden Hash (`Old64BitCRC`) und dem angegebenen Byte. Die crc64-Prüfsumme (hash-Wert) ist so lange im Klartext lesbar, wie der eingegebene Text nicht länger als 8 Zeichen ist.

```
int64 crc64 (String s, int64 hash)
```

Berechnet den crc64-Hash aus den Zeichen (ohne Längenbyte), die sich in dem angegebenen String befinden. Wenn man den Hash lesbar gestalten möchte, so ist dies möglich, wenn der Text nicht mehr als 8 Zeichen umfasst und eine 0 als Start-`hash` angegeben wird. Wenn man den Hash stärker verschlüsseln möchte, gibt man als Start-`hash` eine Primzahl, z.B. 0x1021101110211001 an.

```
uint64 xorshift128plus (uint64 seed0, uint64 seed1)
```

Der ursprüngliche Algorithmus hatte einmal eine Periode von 2^128 - 1 und bestand alle Tests der BigCrush Test Suite. Es ist dabei zu beachten, dass dieser ursprüngliche Algorithmus zwar ein Pseudeo-Zufallszahlen-Generator mit einer langen Periode war, aber keine kryptographisch relevanten Zufallszahlen erzeugt hat. Die `xorshift128plus()` Funktion kann mit Zufallszahlen gefüttert werden, um sicherheitstechnisch relevante Ergebnisse zu erzielen. Die hier verwendete, vereinfachte, Version wird in dieser Library mit jeweils zwei 64 Bit Zufallszahlen aus der `rnd()` Funktion gefüttert, um ordentlich zu funktionieren, und liefert auch immer nur den 1. Grad des Polynoms, was für diese Art der Anwendung richtige Zufallszahlen ergibt, die qualitativ besser sind als die beiden angegebenen Zufallszahlen (`seed0, seed1`).

```
uint64 xorshift128plus (String s, uint64 hash)
```

Berechnet den xorshift128plus-Hash aus den Zeichen (ohne Längenbyte), die sich in dem angegebenen String befinden. Obwohl der Hash bereits ab dem 1. Zeichen nicht mehr klar lesbar ist, sollte man als Start-`hash` eine Primzahl, z.B. 0x1021101110211001 angeben, wenn man eine wesentlich stärkere Verschlüsselung erreichen möchte.

Zufallszahlen Funktionen

```
int64 rnd_0
```

Dieser Variable enthält die zuletzt mit `rnd()` berechnete, ganzzahlige Zufallszahl.

```
double rndreal_0
```

Dieser Variable enthält die zuletzt mit `rndreal()` berechnete, Fliekomma-Zufallszahl.

```
byte rndbyte ()
```

Dieser Funktion liefert eine ganzzahlige Zufallszahl zwischen 0 und 255. Dabei wird nicht nur die eingebaute `random()` Funktion benutzt, sondern eine Kombination zusammen mit dem Microsekunden-Timer. Daurch entstehen bei jedem Programmstart neue Zufallszahlen, die trotzdem statistisch gesehen gleichmäßig über den Werteraum verteilt sind, wenn man eine große Anzahl von Zufallszahlen berechnet. Alle anderen Zufallszahlen-Funktionen aus dieser Bibliothek benutzen diese Funktion zur Berechnung von Zufallszahlen.

```
byte rndbytesecure ()
```

Dieser Funktion liefert eine ganzzahlige Zufallszahl zwischen 0 und 255. Diese wird mittels 32 Zufallszahlen, die mit `rndbyte()` erzeugt und über eine 126 Bit Polynomfunktion verbessert wurden, berechnet. Dadurch ergibt sich eine kryptographisch deutlich bessere Zufallszahl als eine mit `rndbyte()` berechnete Zufallszahl.

```
int64 rnd (int64 n)
```

Dieser Funktion liefert eine ganzzahlige Zufallszahl zwischen 0 und n (einschließlich 0 und n). Diese wird mittels 8 Zufallszahlen, die mit `rndbyte()` erzeugt werden, berechnet. Eine 0 als Parameter, `rnd(0)`, ergibt die zuletzt berechnete Zufallszahl.

```
double rndreal (double n)
```

Dieser Funktion liefert eine Fließkomma-Zufallszahl zwischen 0.0 und n.0 (einschließlich 0.0 und n.0). Diese wird mittels 8 Zufallszahlen, die mit `rndbyte()` erzeugt werden, berechnet. Eine 0.0 als Parameter, `rndreal(0.0)`, ergibt die zuletzt berechnete Zufallszahl.

String Funktionen

```
String chr (int c)
```

Ergibt einen String der genau ein Zeichen mit dem angegebenen Character Code enthält. Der Unterschied zu der standardmäßigen Funktion `char(c)` ist der Typ des Funktionsergebnisses. Die Funktion `chr()` gibt einen echten dynamischen String zurück. Wenn man z.B. mit dem Befehl „`Serial.print(" = " + chr(34));`“ versucht, ein Anführungszeichen hinter einem Text auszugeben, dann sieht man im günstigsten Fall irgendwelche Zeichen, nicht aber das Gleichheitszeichen oder das Anführungszeichen. Das kann im Extremfall sogar zum Reset des IoT-Bricks führen. Auf anderen Arduinos sind die Ergebnisse nicht weniger problematisch. Das liegt daran, dass bei der Umwandlung von `char` in `String` keine saubere Konvertierung vorgenommen wird. Strings dagegen sind in der Arduino-Welt dynamisch und Unicode-kompatibel, d.h. sie belegen nur den Speicherplatz, der tatsächlich benötigt wird. Ein leerer String belegt nur 2 Bytes. Die Speicherverwaltung von Strings wird von der Arduino IDE dynamisch vorgenommen. Es können dabei erfreulicherweise keine illegalen Pointer oder Adressverletzungen entstehen, die zum Reset führen. Daher sollte man auf Deklarationen wie „`char var[]`“ oder „`char *var`“ wann immer das möglich ist komplett verzichten. Wird von einer Funktion „`char *`“ zurückgegeben, so kann man stattdessen auch „`(String)funktion()`“ schreiben. Wird für einen Bibliotheksaufruf als Parameter „`char *var`“ verlangt, so kann man stattdessen auch „`var.c_str()`“ schreiben und „`var`“ kann ein String sein. In jedem Fall sollte man für eigene Programme immer `chr()` und nicht `char()` verwenden. Diese Funktion kann für Unicode-Zeichen auch Werte > 255 akzeptieren.

```
int asc (String s)
```

Ergibt den Character Code des ersten Zeichens im String. Diese Funktion kann bei Unicode-Zeichen auch Werte > 255 ergeben.

```
int len (String s)
```

Ergibt die Anzahl von Zeichen im String. Diese Funktion ergibt z.B. für ein einzelnes Unicode-Zeichen mit einem Character Code > 255 eine 2 und repräsentiert damit zwar den Speicherplatzbedarf des Inhalts eines Strings, nicht aber die tatsächliche Anzahl von Zeichen, wenn es sich um Unicode-Zeichen mit einem Character Code > 255 handelt. Dabei hanedlt es sich um eine Beschränkung der Arduino IDE, die keine Funktion zur Verfügung stellt, die die Anzahl von Unicode-Zeichen in einem String auszählt.

```
bool number (byte n)
```

Ergibt `true`, wenn es sich bei dem angegeben Character Code um eine Zahl zwischen 48 und 57 handelt („0“-„9“).

```
bool letter (byte n)
```

Ergibt `true`, wenn es sich bei dem angegeben Character Code um eine Zahl zwischen 65 und 90 oder 97 und 122 handelt („A“-„Z“ und „a“-„z“, keine diakritischen Buchstaben oder Ligaturen).

```
String spc (int length)
```

Ergibt einen String, der aus der angegebenen Anzahl von Leerzeichen besteht.

```
String ucase (String s)
```

Wandelt den angegebenen String in Großbuchstaben um.

```
String lcase (String s)
```

Wandelt den angegebenen String in Kleinbuchstaben um.

```
String boolstr (bool b, String caseTrue, String caseFalse)
```

Ergibt wahlweise einen der angegebenen Strings, je nachdem, ob man als b `true` oder `false` übergibt.

```
String left (String s, int length)
```

Ergibt den linken Teil des Strings mit der angegebenen Länge bzw. der angegebenen Anzahl von Buchstaben. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion.

```
String part (String s, int start)
```

Ergibt den hinteren Teil des Strings ab der angegebenen Start-Position. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion. Der erste Buchstabe im String hat dabei die Start-Position 1.

```
String right (String s, int length)
```

Ergibt den rechten Teil des Strings mit der angegebenen Länge bzw. der angegebenen Anzahl von Buchstaben. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion.

```
String mid (String s, int start, int length)
```

Ergibt den hinteren Teil des Strings ab der angegebenen Start-Position mit der angegebenen Länge bzw. der angegebenen Anzahl von Buchstaben. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion. Der erste Buchstabe im String hat dabei die Start-Position 1.

```
String replace (String s, String oldstr, String newstr)
```

Ersetzt in dem angegebenen String alle Vorkommnisse von „`oldstr`“ mit dem neuen „`newstr`“.

```
int position (String s, String t, int start)
```

Ergibt die Position des Strings „t“ im String „s“ und sucht dabei ab der angegebenen Position nach „t“. Wenn Start <= 1 ist, dann wird der gesamte String „s“ durchsucht. Der erste Buchstabe im String hat dabei die Position 1.

```
String indchar (String s, int i)
```

Ergibt das Zeichen an der angegebenen Position als String. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion. Der erste Buchstabe im String hat dabei die Position 1.

```
int indasc (String s, byte i)
```

Ergibt das Zeichen an der angegebenen Position als Character Code. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion. Der erste Buchstabe im String hat dabei die Position 1.

```
String deleteasc (String s, int p)
```

Löscht genau ein Zeichen an der angegebenen Position aus dem String heraus. Es gilt dabei die gleiche Unicode-Problematik wie bei der `len()` Funktion. Der erste Buchstabe im String hat dabei die Position 1.

```
String repeatstr (String s, int n)
```

Ergibt einen String, der aus der angegebenen Anzahl des Strings „s“ besteht.

```
String repeatasc (int a, int n)
```

Ergibt einen String, der aus der angegebenen Anzahl von Zeichen mit dem Character Code a besteht.

```
String strform (int64 n, int asc, byte MinVorkomma, bool TausenderPunkt)
```

Ergibt eine Zahl als String. Dabei kann die Mindestanzahl von Ziffern angegeben werden. Hat die Zahl weniger Ziffern, so werden links von der Zahl Zeichen mit dem angegebenen Character Code eingefügt. Auf Wunsch können 3-er Gruppen jeweils mit einem Tausenderpunkt getrennt werden.

```
String str (int64 n)
```

Ergibt eine Zahl als String. Dabei werden 3-er Gruppen jeweils mit einem Tausenderpunkt getrennt.

```
String strIP (IPAddress ip, bool leadingZero)
```

Wandelt eine IP-Adresse in einen String um. Dabei werden die vier 3-er Gruppen (von 0-255) jeweils mit einem Punkt voneinander getrennt. Wenn führende Nullen gewünscht werden, so muss für `leadingZero` ein `true` übergeben werden.

```
String strrealform (double n, int asc, byte MinVorkomma, byte MaxNachkomma,
                    bool TausenderPunkt, bool CutZero)
```

Ergibt eine Fließkommazahl als String. Dabei kann die Mindestanzahl von Ziffern angegeben werden. Hat die Zahl weniger Ziffern, so werden links von der Zahl Zeichen mit dem angegebenen Character Code eingefügt. Weiterhin kann die maximale Anzahl von Nachkommastellen begrenzt werden. Dabei werden nachfolgende Nullen im Nachkommateil der Zahl wahlweise abgeschnitten. Auf Wunsch können 3-er Gruppen jeweils mit einem Tausenderpunkt getrennt werden.

```
String strrealform (double n, int asc, byte MinVorkomma, byte MaxNachkomma,
                    bool TausenderPunkt, bool CutZero, bool ShowPlus)
```

Ergibt eine Fließkommazahl als String. Dabei kann die Mindestanzahl von Ziffern angegeben werden. Hat die Zahl weniger Ziffern, so werden links von der Zahl Zeichen mit dem angegebenen Character Code eingefügt. Weiterhin kann die maximale Anzahl von Nachkommastellen begrenzt werden. Dabei werden nachfolgende Nullen im Nachkommateil der Zahl wahlweise abgeschnitten. Auf Wunsch

können 3-er Gruppen jeweils mit einem Tausenderpunkt getrennt werden. Weiterhin können positive Zahlen (außer der Null) mit einem "+" vor der Zahl versehen werden.

```
String strreal (double n, int Nachkomma)
```

Ergibt eine Fließkommazahl als String. Dabei werden 3-er Gruppen jeweils mit einem Tausenderpunkt getrennt. Weiterhin kann die maximale Anzahl von Nachkommastellen begrenzt werden.

```
int64 val (String s)
```

Wandelt einen String in eine Integer-Zahl um.

```
double realval (String s)
```

Wandelt einen String in eine Fließkommazahl um.

```
IPAddress IPval (String s)
```

Wandelt einen String in eine IP-Adresse um. Der String muss vom Format "000.000.000.000" sein. Führende Nullen brauchen nicht mit angegeben zu werden. Falls der String einen fehler anthält, so wird "`0.0.0.0`" als `IPAddress` übergeben.

```
String fillcenter (String s, String fill, int width)
```

Ergibt einen String, der mittig den angegebenen String `s` enthält und rechts sowie links gleichmäßig mit dem angegebenen String `fill` auf die vorgegebene Länge `width` aufgefüllt wird. Damit können zentrierte Text-Überschriften für die Terminal-Ausgabe sowie für das Internet erzeugt werden. Damit diese Funktion einwandfrei funktioniert, darf in `fill` lediglich ein Zeichen übergeben werden. Bei dem Zeichen darf es sich um ein zusammengesetztes Unicode-Zeichen handeln, das auf dem Bildschirm als einzelnes Zeichen dargestellt wird. Auch sind Kombinationen aus Sonderzeichen und Steuerzeichen (Ctrl-Codes) erlaubt, solange diese optisch nur ein Zeichen auf dem Bildschirm darstellen. Daher wird an dieser Stelle auch ein `String`-Parameter verwendet und kein `int` Character Code. Je nach gewähltem `String` `s` und Länge `width` kann es vorkommen, dass die Trennzeichen auf der rechten Seite ein Zeichen länger sind als auf der linken Seite. Das ist genau dann der Fall, wenn die aufzufüllende Zeichenanzahl (`width` minus Länge(`s`)) ungerade ist. Wenn der angegebene String `s` länger als die angegebene Länge `width` ist, so wird der String `s` unverändert übergeben, d.h. es werden keine Zeichen rechts abgeschnitten.

Konvertierungs-Funktionen

```
String cleanasc (String s)
```

Ersetzt alle Zeichen mit Character Code < 32 mit den Text „Ctrl-X“, wobei X das zugehörige Ctrl-Steuerzeichen darstellt. Ersetzt weiterhin alle Zeichen mit Character Code > 127 in ein ähnlich aussehendes Zeichen oder eine Zeichenfolge, z.B. „ä“ und „ae“ oder „€“ in „EUR“. Können die Zeichen nicht in eine sinnvolle Zeichenfolge umgewandelt werden, so wird der Character Code in geschweiften Klammern angegeben, z.B. „{254}“. Wird für die Ausgabe von Informationen über ein Terminalprogramm benötigt, damit Zeichen korrekt dargestellt werden können und keine unerwarteten Artefakte auf dem Terminal-Bildschirm erscheinen.

```
String cleanhtml (String s)
```

Ersetzt viele häufig benutzte Zeichen mit Character Code > 127 in ein identisch aussehendes html-Sonderzeichen. Wird für die Ausgabe von Informationen auf einer Web-Seite (html-Seite) benötigt, damit Sonderzeichen korrekt dargestellt werden und keine unerwarteten Artefakte im Browser-Fenster erscheinen. Da nicht alle existierenden Sonderzeichen ersetzt werden, muss diese Funktion in der Library bei Bedarf um weitere Sonderzeichen ergänzt werden.

```
String convertescape (String s)
```

Wandelt Unicode-Escape-Sequenzen vom Typ "`\u1234`" innerhalb eines Strings in die entsprechenden Unicode-Zeichen um. Der Character-Code des Unicode-Zeichens nach dem "`u`" muss dabei vierstellig hexadezimal mit führenden Nullen angegeben werden. Ein "ß" hat z.B. die Escape-Sequenz "`\u00df`".

```
String quotate (String s)
```

Übergibt den String in Anführungszeichen (Character-Code 34).

Hexadezimale Funktionen

```
bool hexnumber (byte n)
```

Ergibt `true`, wenn es sich bei dem angegeben Character Code um eine Zahl zwischen 48 und 57 oder 65 und 70 oder 97 und 102 handelt handelt („0“-„9“, „A“-„F“, „a“-„f“).

```
String hex (uint64 v)
```

Ergibt eine Zahl als Hexadezimal-String. Hexadezimale Zahlen haben kein Vorzeichen.

```
String hexbyte (byte v)
```

Ergibt eine Zahl als zweistelligen Hexadezimal-String. Hexadezimale Zahlen haben kein Vorzeichen.

```
String hexword (uint v)
```

Ergibt eine Zahl als vierstelligen Hexadezimal-String. Hexadezimale Zahlen haben kein Vorzeichen.

```
String hexlong (ulong v)
```

Ergibt eine Zahl als achtstelligen Hexadezimal-String. Hexadezimale Zahlen haben kein Vorzeichen.

```
String hexint64 (uint64 v)
```

Ergibt eine Zahl als 16-stelligen Hexadezimal-String. Hexadezimale Zahlen haben kein Vorzeichen.

```
uint64 dec (String s)
```

Wandelt einen Hexadezimal-String in eine Integer-Zahl um. Hexadezimale Zahlen haben kein Vorzeichen.

```
String newuniqueid ()
```

Ergibt einen zufälligen, 32-stelligen, hexadezimalen String.

Datum & Uhrzeit Funktionen

```
byte monthdays (byte month, int year)
```

Ergibt die Anzahl der Tage in dem angegebenen Monat im angegebenen Jahr (berücksichtigt Schaltjahre).

```
long timetoseconds (int days, byte hour, byte minute, byte second)
```

Ergibt die Anzahl von Sekunden für die angegebene Zeitspanne in Tagen, Stunden, Minuten und Sekunden.

```
long datetodays2000 (int year, byte month, byte day)
```

Ergibt die Anzahl der Tage seit dem 01.01.2000 für das angegebene Jahr im angegebenen Monat an dem angegebenen Tag.

```
byte dayofweek (int year, byte month, byte day)
```

Ergibt den Wochentag für das angegebene Jahr im angegebenen Monat an dem angegebenen Tag. Dabei ist 0 = Sonntag, 1 = Montag, 2 = Dienstag, 3 = Mittwoch, 4 = Donnerstag, 5 = Freitag und 6 = Samstag.

```
String dayname (int day)
```

Ergibt den Namen des angegebenen Tages. Dabei ist 0 = Sonntag, 1 = Montag, 2 = Dienstag, 3 = Mittwoch, 4 = Donnerstag, 5 = Freitag und 6 = Samstag.

```
String monthname (int month)
```

Ergibt den Namen des angegebenen Monats. Dabei ist 1 = Januar, 2 = Februar, 3 = März, 4 = April, 5 = Mai, 6 = Juni, 7 = Juli, 8 = August, 9 = September, 10 = Oktober, 11 = November, 12 = Dezember.

```
byte monthnumber (String month)
```

Ergibt den Monat aus dem angegebenen Namen. Dabei ist 1 = Januar, 2 = Februar, 3 = März, 4 = April, 5 = Mai, 6 = Juni, 7 = Juli, 8 = August, 9 = September, 10 = Oktober, 11 = November, 12 = Dezember.

```
String datestr (t_DateTime DateTime)
```

Wandelt das angegebene Datum vom Typ `t_DateTime` in einen lesbaren String vom Format DD.MM.YYYY um.

```
String timestr (t_DateTime DateTime)
```

Wandelt die angegebene Uhrzeit vom Typ `t_DateTime` in einen lesbaren String vom Format hh:mm:ss um.

Terminal Library Listing

Die Terminal-Library stellt Befehle zur Verfügung, um mit einem Terminal-Programm zu kommunizieren. Um alle Befehle incl. der Reset-Funktion durch das Terminal-Programm komfortabel nutzen zu können, muss seitens des Terminal-Programms mit dem UniTerminal-Protokoll gearbeitet werden. Trotzdem sind die meisten Funktionen, mit etwas umständlicherer Bedienung, auch mit dem in der Arduino IDE eingebauten seriellen Monitor nutzbar. Der serielle Monitor unterstützt allerdings außer Return und Line Feed keine Steuerzeichen, z.B. um den Bidschirm löschen zu können.

```
// Terminal Library
// 1.00 - 2017-05-23
// (c) Copyright 2017
//
// Zur Verwendung mit allen Allnet Libraries


// *** Konstanten für indizierte 8 Farben ***

const byte t_RGBColor8ValueRed    [ ] = {0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF};
const byte t_RGBColor8ValueGreen  [ ] = {0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF};
const byte t_RGBColor8ValueBlue   [ ] = {0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF};

// *** Konstanten für indizierte 16 Farben ***

const byte t_RGBColor16ValueRed   [ ] = {0x00, 0xFF, 0x00, 0xFF, 0x00, 0x54, 0x00, 0x00,
                                         0x80, 0xFF, 0xA8, 0xFF, 0x00, 0xFF, 0x00, 0xFF};
const byte t_RGBColor16ValueGreen [ ] = {0x00, 0x00, 0x00, 0x00, 0x80, 0x54, 0x00, 0x80,
                                         0x40, 0x80, 0xA8, 0x80, 0xFF, 0xFF, 0xFF, 0xFF};
const byte t_RGBColor16ValueBlue  [ ] = {0x00, 0x00, 0x80, 0xFF, 0x00, 0x54, 0xFF, 0xFF,
                                         0x00, 0x00, 0xA8, 0x80, 0x00, 0x00, 0xFF, 0xFF};

// *** Konstanten ***

const byte t_ArduinoUnknown           =  0;     // Microcontroller nicht identifiziert
const byte t_ArduinoMega168           =  1;     // AT Mega 168 Microcontroller
const byte t_ArduinoMega1280          =  2;     // Arduino Mega 1280
const byte t_ATmega1284P              =  3;     // AT Mega 1284P Microcontroller
const byte t_ArduinoMega2560          =  4;     // Arduino Mega 2560
const byte t_ArduinoNano              =  5;     // Arduino Nano
const byte t_ArduinoMicro             =  6;     // Arduino Micro
const byte t_ArduinoDue               =  7;     // Arduino Due
const byte t_Teensy30                 =  8;     // Teensy 3.0
const byte t_Teensy31                 =  9;     // Teensy 3.1
const byte t_ESP8266                  = 10;     // ESP 8266 (IoT-Brick)
```

```
const byte t_Input_String            = 0;      // Modus für t_TerminalInput: String-Eingabe
const byte t_Input_Integer           = 1;      // Modus für t_TerminalInput: Integer-Eingabe
const byte t_Input_Real              = 2;      // Modus für t_TerminalInput: Real-Eingabe


// *** Variablen ***

byte       t_ResetSequence           = 0;
String     t_TerminalType            = "";      // Terminaltyp als String ("TTY", "UniTerminal" usw.)
String     t_TerminalRespond         = "";      // Die Sequenz, die beim Initialisieren vom Terminal kommt
byte       t_TerminalWidth           = 0;       // Anzahl der Spalten, normalerweise 80
byte       t_TerminalHeight          = 0;       // Anzahl der Zeilen, normalerweise 25
bool       t_TerminalInputOK         = false;   // true = Eingabe wurde mit [Return] abgeschlossen

// *** Forward-Deklarationen ***

String t_TerminalGetKey ();

// *** Utilities ***

byte t_CPUtype ()
{
#if defined(__AVR_ATmega1280__)              // Arduino Mega 1280 (8 Bit, 128 kB Flash, 8 kB SRAM)
    return t_ArduinoMega1280;
#elif defined(__AVR_ATmega2560__)            // Arduino Mega 2560 (8 Bit, 256 kB Flash, 8 kB SRAM)
    return t_ArduinoMega2560;
#elif defined(__AVR_ATmega328P__)            // Arduino Nano, Mini und Uno (8 Bit, 32 kB Flash, 2 kB SRAM)
    return t_ArduinoNano;
#elif defined(__AVR_ATmega32U4__)            // Arduino Leonardo, Micro (8 Bit, 32 kB Flash, 2,5 kB SRAM)
    return t_ArduinoMicro;
#elif defined(__AVR_ATmega168__)             // Alte Aruionos (8 Bit, 16 kB Flash, 1 kB SRAM)
    return t_ArduinoMega168;
#elif defined(__AVR_ATmega1284P__)
    return t_ATmega1284P;
#elif defined(__SAM3X8E__)                   // Arduino Due (32 Bit, 512 kB Flash, 96 kB SRAM)
    return t_ArduinoDue;
#elif defined(__MK20DX128__)                 // Teensy 3.0  (32 Bit, 128 kB Flash, 16 kB SRAM)
    return t_Teensy30;
#elif defined(__MK20DX256__)                 // Teensy 3.1  (32 Bit, 256 kB Flash, 64 kB SRAM)
    return t_Teensy31;
#elif defined(ESP8266)                       // ESP 8266    (32 Bit, 4096 kB Flash, 96 kB SRAM)
    return t_ESP8266;
#else
    return t_ArduinoUnknown;
#endif
}

String t_CPUname ()
{
#if defined(__AVR_ATmega1280__)              // Arduino Mega 1280 (8 Bit, 128 kB Flash, 8 kB SRAM)
```

```
    return "Arduino Mega 1280";
#elif defined(__AVR_ATmega2560__)              // Arduino Mega 2560 (8 Bit, 256 kB Flash, 8 kB SRAM)
    return "Arduino Mega 2560";
#elif defined(__AVR_ATmega328P__)              // Arduino Nano, Mini und Uno (8 Bit, 32 kB Flash, 2 kB SRAM)
    return "Arduino Nano";
#elif defined(__AVR_ATmega32U4__)              // Arduino Leonardo, Micro (8 Bit, 32 kB Flash, 2,5 kB SRAM)
    return "Arduino Micro";
#elif defined(__AVR_ATmega168__)               // Alte Aruionos (8 Bit, 16 kB Flash, 1 kB SRAM)
    return "Arduino Mega 168";
#elif defined(__AVR_ATmega1284P__)
    return "Arduino Mega 1284";
#elif defined(__SAM3X8E__)                     // Arduino Due (32 Bit, 512 kB Flash, 96 kB SRAM)
    return "Arduino Due";
#elif defined(__MK20DX128__)                   // Teensy 3.0  (32 Bit, 128 kB Flash, 16 kB SRAM)
    return "Teensy 3.0";
#elif defined(__MK20DX256__)                   // Teensy 3.1  (32 Bit, 256 kB Flash, 64 kB SRAM)
    return "Teensy 3.1";
#elif defined(ESP8266)                         // ESP 8266    (32 Bit, 4096 kB Flash, 96 kB SRAM)
    return "ESP 8266";
#else
    return "(Unknown)";
#endif
}

byte t_CPUbits ()
{
#if defined(__AVR_ATmega1280__)                // Arduino Mega 1280 (8 Bit, 128 kB Flash, 8 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega2560__)              // Arduino Mega 2560 (8 Bit, 256 kB Flash, 8 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega328P__)              // Arduino Nano, Mini und Uno (8 Bit, 32 kB Flash, 2 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega32U4__)              // Arduino Leonardo, Micro (8 Bit, 32 kB Flash, 2,5 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega168__)               // Alte Aruionos (8 Bit, 16 kB Flash, 1 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega1284P__)
    return 8;
#elif defined(__SAM3X8E__)                     // Arduino Due (32 Bit, 512 kB Flash, 96 kB SRAM)
    return 32;
#elif defined(__MK20DX128__)                   // Teensy 3.0  (32 Bit, 128 kB Flash, 16 kB SRAM)
    return 32;
#elif defined(__MK20DX256__)                   // Teensy 3.1  (32 Bit, 256 kB Flash, 64 kB SRAM)
    return 32;
#elif defined(ESP8266)                         // ESP 8266    (32 Bit, 4096 kB Flash, 96 kB SRAM)
    return 32;
#else
    return 0;
#endif
}
```

```
int t_CPUflashRAM ()
{
#if defined(__AVR_ATmega1280__)               // Arduino Mega 1280 (8 Bit, 128 kB Flash, 8 kB SRAM)
    return 128;
#elif defined(__AVR_ATmega2560__)             // Arduino Mega 2560 (8 Bit, 256 kB Flash, 8 kB SRAM)
    return 256;
#elif defined(__AVR_ATmega328P__)             // Arduino Nano, Mini und Uno (8 Bit, 32 kB Flash, 2 kB SRAM)
    return 32;
#elif defined(__AVR_ATmega32U4__)             // Arduino Leonardo, Micro (8 Bit, 32 kB Flash, 2,5 kB SRAM)
    return 32;
#elif defined(__AVR_ATmega168__)              // Alte Aruionos (8 Bit, 16 kB Flash, 1 kB SRAM)
    return 16;
#elif defined(__AVR_ATmega1284P__)
    return 16;
#elif defined(__SAM3X8E__)                    // Arduino Due (32 Bit, 512 kB Flash, 96 kB SRAM)
    return 512;
#elif defined(__MK20DX128__)                  // Teensy 3.0  (32 Bit, 128 kB Flash, 16 kB SRAM)
    return 128;
#elif defined(__MK20DX256__)                  // Teensy 3.1  (32 Bit, 256 kB Flash, 64 kB SRAM)
    return 256;
#elif defined(ESP8266)                        // ESP 8266    (32 Bit, 4096 kB Flash, 96 kB SRAM)
    return 4096;
#else
    return 0;
#endif
}

int t_CPUstaticRAM ()
{
#if defined(__AVR_ATmega1280__)               // Arduino Mega 1280 (8 Bit, 128 kB Flash, 8 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega2560__)             // Arduino Mega 2560 (8 Bit, 256 kB Flash, 8 kB SRAM)
    return 8;
#elif defined(__AVR_ATmega328P__)             // Arduino Nano, Mini und Uno (8 Bit, 32 kB Flash, 2 kB SRAM)
    return 2;
#elif defined(__AVR_ATmega32U4__)             // Arduino Leonardo, Micro (8 Bit, 32 kB Flash, 2,5 kB SRAM)
    return 3;
#elif defined(__AVR_ATmega168__)              // Alte Aruionos (8 Bit, 16 kB Flash, 1 kB SRAM)
    return 1;
#elif defined(__AVR_ATmega1284P__)
    return 1;
#elif defined(__SAM3X8E__)                    // Arduino Due (32 Bit, 512 kB Flash, 96 kB SRAM)
    return 96;
#elif defined(__MK20DX128__)                  // Teensy 3.0  (32 Bit, 128 kB Flash, 16 kB SRAM)
    return 16;
#elif defined(__MK20DX256__)                  // Teensy 3.1  (32 Bit, 256 kB Flash, 64 kB SRAM)
    return 64;
#elif defined(ESP8266)                        // ESP 8266    (32 Bit, 4096 kB Flash, 96 kB SRAM)
    return 96;
```

```
#else
    return 0;
#endif
}

void t_Restart ()
{
#if defined(ESP8266)                            // ESP 8266
    ESP.reset();
#elif defined(__MK20DX128__) || defined(__MK20DX256__) || defined(__SAM3X8E__) // 32 Bit CPUs Teensy/Due
#define RESTART_ADDR       0xE000ED0C
#define READ_RESTART()     (*(volatile uint32_t *)RESTART_ADDR)
#define WRITE_RESTART(val) ((*(volatile uint32_t *)RESTART_ADDR) = (val))
    WRITE_RESTART(0x5FA0004);
#else                                           // 8 Bit CPU
    asm volatile ("  jmp 0");
#endif
    delay(1000);                                    // 1 Sekunde warten
}

void t_ShutDown ()
{
    while (true)
    {
        t_TerminalGetKey();                          // Reset-Anforderung prüfen
        delay(100); // 100 ms. warten
#if defined(ESP8266)                           // ESP 8266
        ESP.wdtFeed();                              // Watchdog Timer zurücksetzen
        ESP.wdtDisable();                           // Watchdog Timer ausschalten
#endif
    }
}

t_DateTime t_CompileDateTime ()
{
    const char* compile_date = __DATE__;
    const char* compile_time = __TIME__;
    String c_date = "";
    String c_time = "";
    for (byte i = 0; i <= 10; i++)
    {
        c_date = c_date + chr(compile_date[i]);
    }
    for (byte i = 0; i <= 7; i++)
    {
        c_time = c_time + chr(compile_time[i]);
    }
    t_DateTime DateTime;
    DateTime.year = val(part(c_date, 8));
    DateTime.month = monthnumber(c_date);
```

```
    DateTime.day = val(mid(c_date, 5, 2));
    DateTime.hour = val(left(c_time, 2));
    DateTime.minute = val(mid(c_time, 4, 2));
    DateTime.second = val(part(c_time, 7));
    return DateTime;
}


// *** Terminal USB Serial Steuerzeichen ***

String t_TerminalCursorHome ()
{
    return (chr(2)); // Cursor Home
}

String t_TerminalClearScreen ()
{
    return (chr(12)); // Clear Screen
}

String t_TerminalClearEndOfLine ()
{
    return (chr(27) + "E"); // Clear End Of Line
}

String t_TerminalClearEndOfScreen ()
{
    return (chr(27) + "F"); // Clear End Of Screen
}

String t_TerminalTab ()
{
    return (chr(9)); // Tabulator
}

String t_TerminalGotoYX (int Y, int X)
{
    if (X < 1)
    {
        X = 1;
    }
    if (Y < 1)
    {
        Y = 1;
    }
    if (X > 96)
    {
        X = 96;
    }
    if (Y > 96)
```

```
    {
        Y = 96;
    }
    X = X + 31;
    Y = Y + 31;
    String result = chr(27) + "=" + chr(Y) + chr(X);
    return result;
}

String t_TerminalHtab (int X)
{
    if (X < 1)
    {
        X = 1;
    }
    if (X > 96)
    {
        X = 96;
    }
    String result = chr(1) + repeatasc(21, X - 1);
    return result;
}


// *** Terminal USB Serial ***

int t_TerminalPrint (String s)
{
    int bytesSent = 0;
    if (s != "")
    {
        bytesSent = Serial.print(s);
    }
    return (bytesSent);
}

int t_TerminalPrintLn (String s)
{
    return t_TerminalPrint(s + chr(13));
}

String t_TerminalGetKey ()
{
    String s = "";
    if (Serial.available() > 0)
    {
        char c = Serial.read(); // Read a character
        s += c;
        if (c == 31)
        {
```

```
            t_ResetSequence++;
            if (t_ResetSequence == 10)
            {
                t_Restart();
            }
        }
        else
        {
            t_ResetSequence = 0;
        }
    }
    return (s);
}

String t_TerminalRead ()
{
    String s = "";
    while (Serial.available() > 0)
    {
        s += t_TerminalGetKey();
    }
    return (s);
}

String t_TerminalGetChar ()
{
    String s = "";
    while (s == "")
    {
        #if defined(ESP8266)                        // ESP 8266
            ESP.wdtFeed();                          // Watchdog Timer zurücksetzen
        #endif
        s += t_TerminalGetKey();
    }
    return (s);
}

int t_TerminalGetAsc ()
{
    int a = asc(t_TerminalGetChar());
    return (a);
}

String t_TerminalInput (String s, byte maxChars, byte t_param_input_mode)
{
    t_TerminalInputOK = false;
    int    HPos = len(s) + 1;
    String InputStr = "";
    int    CursorPos = 0;
    int    InputLen = 0;
```

```
int    zchn = 0;
int    a = 0;
int    t = 0;

Serial.print(s);
while ((zchn != 13) && (zchn != 27))
{
    #if defined(ESP8266)                        // ESP 8266
        ESP.wdtFeed();                          // Watchdog Timer zurücksetzen
    #endif
    InputLen = len(InputStr);
    zchn = asc(t_TerminalGetChar());
    if ((zchn < 32) || (zchn == 127))
    {
        switch (zchn)
        {
            case 1:  // Ctrl-A
                CursorPos = 0;
                Serial.print(t_TerminalHtab(HPos));
                break;
            case 2:  // Ctrl-B
                CursorPos = 0;
                Serial.print(t_TerminalHtab(HPos));
                break;
            case 3:  // Ctrl-C
                CursorPos = len(InputStr);
                Serial.print(t_TerminalHtab(HPos + CursorPos));
                break;
            case 4:  // Ctrl-D
                if (CursorPos < len(InputStr))
                {
                    CursorPos = CursorPos + 1;
                    while ((CursorPos < len(InputStr)) && (indasc(InputStr, CursorPos) != 32))
                    {
                        CursorPos = CursorPos + 1;
                    }
                    Serial.print(t_TerminalHtab(HPos + CursorPos));
                }
                break;
            case 5:  // Ctrl-E
                CursorPos = len(InputStr);
                Serial.print(t_TerminalHtab(HPos + CursorPos));
                break;
            case 6:  // Ctrl-F
                CursorPos = len(InputStr);
                Serial.print(t_TerminalHtab(HPos + CursorPos));
                break;
            case 7:  // Ctrl-G
                InputStr = deleteasc(InputStr, CursorPos + 1);
                Serial.print(t_TerminalHtab(HPos + CursorPos));
```

```
    Serial.print(InputStr);
    Serial.print(" ");
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 8:  // Ctrl-H
    CursorPos = CursorPos - 1;
    if (CursorPos < 0)
    {
        CursorPos = 0;
    }
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 9:  // Ctrl-I
    break;
case 10: // Ctrl-J
    CursorPos = len(InputStr);
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 11: // Ctrl-K
    CursorPos = 0;
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 12: // Ctrl-L
    break;
case 13: // Ctrl-M
    t_TerminalInputOK = true;
    break;
case 14: // Ctrl-N
    break;
case 15: // Ctrl-O
    break;
case 16: // Ctrl-P
    break;
case 17: // Ctrl-Q
    break;
case 18: // Ctrl-R
    CursorPos = 0;
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 19: // Ctrl-S
    if (CursorPos > 0)
    {
        CursorPos = CursorPos - 1;
        while ((CursorPos > 0) && (indasc(InputStr, CursorPos) != asc(" ")))
        {
            CursorPos = CursorPos - 1;
        }
        Serial.print(t_TerminalHtab(HPos + CursorPos));
    }
    break;
```

```
case 20: // Ctrl-T
    break;
case 21: // Ctrl-U
    CursorPos = CursorPos + 1;
    if (CursorPos > len(InputStr))
    {
        CursorPos = len(InputStr);
    }
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 22: // Ctrl-V
    break;
case 23: // Ctrl-W
    break;
case 24: // Ctrl-X
    InputStr = part(InputStr, CursorPos + 1);
    CursorPos = 0;
    Serial.print(t_TerminalHtab(HPos));
    Serial.print(left(InputStr + spc(InputLen), InputLen));
    Serial.print(t_TerminalHtab(HPos + CursorPos));
    break;
case 25: // Ctrl-Y
    break;
case 26: // Ctrl-Z
    InputStr = "";
    CursorPos = 0;
    Serial.print(t_TerminalHtab(HPos));
    Serial.print(spc(InputLen));
    Serial.print(t_TerminalHtab(HPos));
    break;
case 27: // Ctrl-[
    InputStr = "";
    t_TerminalInputOK = false;
    Serial.print(t_TerminalHtab(HPos));
    Serial.print(spc(InputLen));
    Serial.print(t_TerminalHtab(HPos));
    break;
case 28: // Ctrl-Backslash
    break;
case 29: // Ctrl-]
    break;
case 30: // Ctrl-^
    t = t_TerminalGetAsc();
    if (number(t))
    {
        a = t - 48;
        t = t_TerminalGetAsc();
        if (number(t))
        {
            a = a * 10 + t - 48;
```

```
                    t = t_TerminalGetAsc();
                    if (number(t))
                    {
                        a = a * 10 + t - 48;
                    }
                }
                if ((a >= 32) && (a <= 126))
                {
                    InputStr = left(InputStr, CursorPos) + chr(a) + part(InputStr, CursorPos + 1);
                    CursorPos = CursorPos + 1;
                    Serial.print(InputStr);
                }
            }
            Serial.print(t_TerminalHtab(HPos + CursorPos));
            break;
        case 31: // Ctrl-_
            break;
        case 127:  // Del
            InputStr = deleteasc(InputStr, CursorPos);
            CursorPos = CursorPos - 1;
            if (CursorPos < 0)
            {
                CursorPos = 0;
            }
            Serial.print(t_TerminalHtab(HPos));
            Serial.print(InputStr + " ");
            Serial.print(t_TerminalHtab(HPos + CursorPos));
            break;
    } // switch
}
else // Reguläre Zeichen eingeben
{
    if ((HPos + InputLen < 80) && (InputLen < maxChars))
    {
        int zeichen = zchn;
        switch (t_param_input_mode)
        {
            case 0:  // String
                break;
            case 1:  // Integer
                if (not(number(zeichen)) && (zeichen != 45)) // "-" und Zahlen sind erlaubt
                {
                    zeichen = 0;
                }
                if ((zeichen == 45) && (CursorPos != 0)) // "-" nur am Anfang erlaubt
                {
                    zeichen = 0;
                }
                break;
            case 2:  // Fließkomma
```

```
                        if (not(number(zeichen)) && ((zeichen < 44) || (zeichen > 46))) // ",-." und Zahlen sind erlaubt
                        {
                            zeichen = 0;
                        }
                        if ((zeichen == 45) && (CursorPos != 0)) // "-" nur am Anfang erlaubt
                        {
                            zeichen = 0;
                        }
                        if (zeichen == 46)  // "."
                        {
                            zeichen = 44;    // ","
                        }
                        if ((zeichen == 44) && (position(InputStr, ",", 1) > 0)) // Nur ein Komma in der Zahl erlaubt
                        {
                            zeichen = 0;
                        }
                    }
                    if (zeichen != 0)
                    {
                        InputStr = left(InputStr, CursorPos) + chr(zeichen) + part(InputStr, CursorPos + 1);
                        CursorPos = CursorPos + 1;
                        Serial.print(t_TerminalHtab(HPos));
                        Serial.print(InputStr);
                        Serial.print(t_TerminalHtab(HPos + CursorPos));
                    }
                }
            } // Reguläres Zeichen
        } // while Loop
        return (InputStr);
}

t_DateTime t_TerminalGetDateTime (bool sync)
{
    t_DateTime DateTime;
    String initSequence = chr(27) + chr(26); // Esc-Ctrl-Z
    Serial.print(initSequence);
    delay(50); // 50 ms warten
    String s = t_TerminalRead ();
    long count = 0;
    if ((left (s, 1) == chr(9)) && (indchar(s, 12) == chr(9)) && (right(s, 1) == chr(9)))
    {
        DateTime.second = val(mid(s, 19, 2));
        do
        {
            count++;
            Serial.print(initSequence);
            delay(50); // 50 ms warten
            s = t_TerminalRead ();
        }
        while (DateTime.second == val(mid(s, 19, 2)));
```

```
        DateTime.day = val(mid(s, 2, 2));
        DateTime.month = val(mid(s, 5, 2));
        DateTime.year = val(mid(s, 8, 4));
        DateTime.hour = val(mid(s, 13, 2));
        DateTime.minute = val(mid(s, 16, 2));
        DateTime.second = val(mid(s, 19, 2));
    }
    return DateTime;
}

void t_TerminalInit ()
{
    String initSequence = "   " + chr(13) + chr(27) + "*" + chr(27) + "Z" + chr(27) + "z";
    Serial.print(initSequence);
    delay(500); // 500 ms warten, bis Init-Sequenz verarbeitet ist und Rückmeldung vorliegt
    String s = t_TerminalRead();
    if ((left (s, 1) == chr(9)) && (right(s, 1) == chr(9)))
    {
        s = mid(s, 2, len(s) - 2);
        int p = position(s, chr(9), 1);
        if (p > 0)
        {
            t_TerminalType = left(s, p - 1);
            s = part(s, p + 2); // Beide Tabulatoren in der Mitte entfernen
            p = position(s,"x", 1);
            if (p > 0)
            {
                t_TerminalWidth = val(left(s, p - 1));
                t_TerminalHeight = val(part(s, p + 1));
                t_TerminalRespond = "";
            }
        }
    }
    else
    {
        t_TerminalRespond = right(t_TerminalRespond, 32) + s;
        if (position(t_TerminalRespond, "ttttt", 1) > 0)
        {
            t_TerminalType = "TTY";
            t_TerminalWidth = 80;
            t_TerminalHeight = 24;
            t_TerminalRespond = "";
        }
        else if (position(t_TerminalRespond, "vvvvv", 1) > 0)
        {
            t_TerminalType = "DEC VT52";
            t_TerminalWidth = 80;
            t_TerminalHeight = 24;
            t_TerminalRespond = "";
        }
```

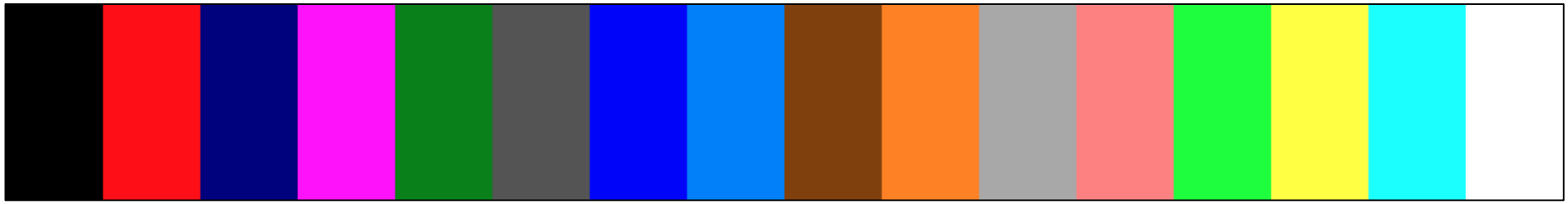
```
    }
}

void t_TerminalWaitInit ()
{
    t_TerminalRespond = "";
    while (t_TerminalWidth == 0)
    {
        t_TerminalInit();
        if (t_TerminalWidth == 0)
        {
            delay(500); // 500 ms. warten
        }
    }
    t_TerminalRespond = "";
}
```

Terminal Library Dokumentation

Die Terminal-Library stellt Befehle zur Verfügung, um mit einem Terminal-Programm zu kommunizieren. Um alle Befehle incl. der Reset-Funktion durch das Terminal-Programm komfortabel nutzen zu können, muss seitens des Terminal-Programms mit dem UniTerminal-Protokoll gearbeitet werden. Trotzdem sind die meisten Funktionen, mit etwas umständlicherer Bedienung, auch mit dem in der Arduino IDE eingebauten seriellen Monitor nutzbar. Der serielle Monitor unterstützt allerdings außer Return und Line Feed keine Steuerzeichen, z.B. um den Bidschirm löschen zu können.

Konstanten für indizierte 16 Farben

Die 16 indizierten Farben sehen wie folgt aus:



```
const byte t_RGBColorValueRed   [ ]   = {0x00, 0xFF, 0x00, 0xFF, 0x00, 0x54, 0x00, 0x00,
                                         0x80, 0xFF, 0xA8, 0xFF, 0x00, 0xFF, 0x00, 0xFF};
const byte t_RGBColorValueGreen [ ]   = {0x00, 0x00, 0x00, 0x00, 0x80, 0x54, 0x00, 0x80,
                                         0x40, 0x80, 0xA8, 0x80, 0xFF, 0xFF, 0xFF, 0xFF};
const byte t_RGBColorValueBlue  [ ]   = {0x00, 0x00, 0x80, 0xFF, 0x00, 0x54, 0xFF, 0xFF,
                                         0x00, 0x00, 0xA8, 0x80, 0x00, 0x00, 0xFF, 0xFF};
```

Allgemeine globale Konstanten

```
const byte t_ArduinoUnknown           =  0;     // Microcontroller nicht identifiziert
const byte t_ArduinoMega168           =  1;     // AT Mega 168 Microcontroller
const byte t_ArduinoMega1280          =  2;     // Arduino Mega 1280
const byte t_ATmega1284P              =  3;     // AT Mega 1284P Microcontroller
const byte t_ArduinoMega2560          =  4;     // Arduino Mega 2560
```

```
const byte t_ArduinoNano             =  5;     // Arduino Nano
const byte t_ArduinoMicro            =  6;     // Arduino Micro
const byte t_ArduinoDue              =  7;     // Arduino Due
const byte t_Teensy30                =  8;     // Teensy 3.0
const byte t_Teensy31                =  9;     // Teensy 3.1
const byte t_ESP8266                 = 10;     // ESP 8266 (IoT-Brick)

const byte t_Input_String            = 0;      // Modus für t_TerminalInput: String-Eingabe
const byte t_Input_Integer           = 1;      // Modus für t_TerminalInput: Integer-Eingabe
const byte t_Input_Real              = 2;      // Modus für t_TerminalInput: Real-Eingabe
```

Globale Variablen

```
byte       t_ResetSequence           = 0;
String     t_TerminalType            = "";      // Terminaltyp als String ("TTY", "UniTerminal" usw.)
String     t_TerminalRespond         = "";      // Die Sequenz, die beim Initialisieren vom Terminal kommt
byte       t_TerminalWidth           = 0;       // Anzahl der Spalten, normalerweise 80
byte       t_TerminalHeight          = 0;       // Anzahl der Zeilen, normalerweise 25
bool       t_TerminalInputOK         = false;   // true = Eingabe wurde mit [Return] abgeschlossen
```

Utilities

Die Utilities dienen dazu, hardwareunabhängigen Code zu schreiben und bei Bedarf zu Prüfen, auf welcher Hardware das Programm aktuell läuft. Daraus ergeben sich zum teil gravierende Unterscheide im Bereich der verwendbaren Digital- und Analog-Leitungen.

```
byte t_CPUtype ()
```

Ergibt den Typ (ein numerischer Index, 0 = unbekannter Typ) des verwendeten Microcontrollers. Der IoT-Brick hat z.B. den Typ 10. Alle identifizierbaren Typen sieht man in der Liste der globalen Konstanten der Terminal-Library.

```
String t_CPUname ()
```

Ergibt den Namen des verwendeten Microcontrollers. Der IoT-Brick hat z.B. den Namen „ESP 8266“.

```
byte t_CPUbits ()
```

Ergibt die Anzahl der Bits eines internen Registers (Registerbreite) des verwendeten Microcontrollers. Die älteren Arduinos haben 8 Bit, die neueren haben 32 Bit. Der IoT-Brick hat ebenfalls 32 Bit.

```
int t_CPUflashRAM ()
```

Ergibt die Gesamtgröße des im verwendeten Microcontroller installierten Flash-RAMs, der für Programme verwendet wird.

```
int t_CPUstaticRAM ()
```

Ergibt die Gesamtgröße des im verwendeten Microcontroller installierten statischen RAMs, der für Variablen verwendet wird.

```
void t_Restart ()
```

Führt einen Reset aus. Dabei wird geprüft, auf welcher Hardware das Programm aktuell läuft, da es keinen einheitlichen Reset-Befehl für alle Arduino-kompatiblen Microcontroller gibt, sondern für jede Gruppe von Microcontrollern ein individueller Reset mit sehr unterschiedlichen Mitteln durchgeführt werden muss. Es handelt sich dabei nicht einfach um ein erneutes Aufrufen der `setup()` Funktion mit nachfolgendem Aufrufen der `loop()` Funktion sondern um einen richtigen Reset, bei dem auch alle globalen Variablen verloren gehen.

```
void t_ShutDown ()
```

Führt eine Endlosschleife aus, ohne dass es zu einem Watch-Dog-bedingten Reset des IoT-Bricks kommt. Der Brick kann jederzeit durch Drücken der Reset-Taste oder über das Terminalprogramm neu gestartet werden.

```
t_DateTime t_CompileDateTime ()
```

Ergibt das Datum und die Uhrzeit, an dem das Programm kompiliert wurde. Falls keine andere Möglichkeit besteht, das aktuelle Datum oder die aktuelle Uhrzeit vom Terminal oder aus dem Internet oder von einem Uhrbaustein abzurufen, so erhält man wenigstens einen Näherungswert. Dieser Näherungswert kann z.B. verwendet werden, bis man einen genaueren Wert bekommt. Das verhindert, dass als Datum auf einmal der 01.01.1970 oder ein ähnlich falsches Datum irgendwo angezeigt wird.

Terminal USB Serial Steuerzeichen

Alle Steuerzeichen-Funktionen ergeben einen String, der auf dem Terminal wahlweise mit `t_TerminalPrint`, `t_TerminalPrintLn`, `Serial.print` oder `Serial.println` ausgegeben werden muss, um das gewünschte Steuerzeichen auszuführen.

```
String t_TerminalCursorHome ()
```

Setzt den Cursor auf die obere linke Ecke des Terminal-Bildschirms.

```
String t_TerminalClearScreen ()
```

Löscht den Terminal-Bildschirm und setzt den Cursor auf die obere linke Ecke.

```
String t_TerminalClearEndOfLine ()
```

Löscht alle Zeichen in der aktuellen Zeile beginnend mit dem Zeichen unter dem Cursor.

```
String t_TerminalClearEndOfScreen ()
```

Löscht alle Zeichen in der aktuellen Zeile beginnend mit dem Zeichen unter dem Cursor und alle Zeilen danach bis zum Ende des Bildschirmens.

```
String t_TerminalTab ()
```

Gibt einen Tabulator (`Ctrl-I`) auf dem Terminal aus.

```
String t_TerminalGotoYX (int Y, int X)
```

Setzt die vertikale Cursorposition (1-25) und die horizontale Cursorposition (1-80) im Terminal auf die angegebenen Werte.

```
String t_TerminalHtab (int X)
```

Setzt die horizontale Cursorposition (1-80) im Terminal auf den angegebenen Wert.

Terminal USB Serial

```
int t_TerminalPrint (String s)
```

Gibt eine Textzeile auf dem Terminal aus.

```
int t_TerminalPrintLn (String s)
```

Gibt eine Textzeile auf dem Terminal aus. Die Textzeile wird gefolgt von einem Return-Steuerzeichen abgeschlossen.

```
String t_TerminalGetKey ()
```

Liest ein einzelnes Zeichen im Terminal ein, das sich aktuell im Buffer befinden und übergibt das einzelne Zeichen als String. Wenn sich keine Zeichen im Buffer befinden, wird ein leerer String übergeben. Es wird nicht auf die Eingabe eines Zeichens gewartet. Dabei wird geprüft, ob seitens des Terminal-Programms eine Reset-Anforderung erfolgt ist und führt in diesem Fall die Reset-Anforderung aus.

```
String t_TerminalRead ()
```

Liest alle Zeichen im Terminal ein, die sich aktuell im Buffer befinden und übergibt die Zeichenfolge als String. Wenn sich keine Zeichen im Buffer befinden, wird ein leerer String übergeben. Es wird nicht auf die Eingabe eines Zeichens gewartet. Dabei wird geprüft, ob seitens des Terminal-Programms eine Reset-Anforderung erfolgt ist und führt in diesem Fall die Reset-Anforderung aus.

```
String t_TerminalGetChar ()
```

Wartet auf ein einzelnes Zeichen im Terminal, liest dieses ein und übergibt das Zeichen als String.

```
int t_TerminalGetAsc ()
```

Wartet auf ein einzelnes Zeichen im Terminal, liest dieses ein und übergibt den Asc II Code des Zeichens.

```
String t_TerminalInput (String s, byte maxChars, byte t_param_input_mode)
```

Wartet auf die EIngabe einer Zeile, die der Benutzer im Terminal eingibt und mit Return abschließen muss. Dabei wird zuerst der String s auf dem Terminal ausgegeben und danach ein String mit bis zu maxChars Zeichen eingelesen. Ein Bearbeiten und Korrigieren des Strings vor dem Drücken von Return ist dabei möglich. Mit `t_param_input_mode` wird angegeben, welche Art von Eingabe erlaubt ist. Dabei sind folgende vordefinierte Konstanten zu verwednen: `t_Input_String`, `t_Input_Integer` und `t_Input_Real`. Punkte werden bei Fließkommazahlen in Kommas umgewandelt. Zum Editieren der Eingabe sind folgende Steuerzeichen erlaubt:

`Ctrl-A`	Cursor auf den Anfang des Textes
`Ctrl-B`	Cursor auf den Anfang des Textes
`Ctrl-C`	Cursor auf das Ende des Textes
`Ctrl-D`	Cursor auf den Anfang des nächsten Wortes
`Ctrl-E`	Cursor auf das Ende des Textes
`Ctrl-F`	Cursor auf das Ende des Textes
`Ctrl-G`	Das Zeichen unter dem Cursor löschen, die Cursorposition bleibt dabei erhalten.
`Ctrl-H`	Cursor ein Zeichen nach links bewegen (Pfeil nach links Taste)
`Ctrl-J`	Cursor auf das Ende des Textes
`Ctrl-K`	Cursor auf den Anfang des Textes
`Ctrl-M`	Eingabe abschließen und String mit der Eingabe als Funktionsergebnis übergeben
`Ctrl-R`	Cursor auf den Anfang des Textes
`Ctrl-S`	Cursor auf den Anfang des Wortes, steht der Cursor bereits am Anfang dann der Anfang des vorherigen Wortes
`Ctrl-U`	Cursor ein Zeichen nach rechts bewegen
`Ctrl-X`	Gesamte Eingabe links vom Cursor löschen
`Ctrl-Z`	Gesamte Eingabe löschen

`Ctrl-[` Eingabe abbrechen und leeren String als Funktionsergebnis übergeben (Esc-Taste)
`Ctrl-^` Es folgen drei Ziffern, die dann ein Zeichen mit dem Asc II Code (032 bis 126) der drei Ziffern erzeugen
`Delete` Das Zeichen links vom Cursor löschen (Rückschritt-Taste)

```
t_DateTime t_TerminalGetDateTime ()
```

Empfängt das aktuelle Datum und die aktuelle Uhrzeit vom Terminal-Programm (nur möglich bei Verwendung des UniTerminal Protokolls).

```
void t_TerminalInit ()
```

Initialisiert die Terminal-Library und findet heraus, mit welchem Terminal-Protokoll eine Verbindung hergestellt wurde, soweit das möglich ist. Es wird nicht auf eine Terminal-Verbindung gewartet. Dazu muss der Befehl `IoT_TerminalWaitInit(bool showMessage)` verwendet werden.

```
void t_TerminalWaitInit ()
```

Diese Funktion wartet so lange, bis mit einem Terminalprogramm eine Verbindung über USB mit dem Microcontroller hergestellt wurde.

Die Übertragung des tatsächlich verwendeten Protokolls sowie der tatsächlichen Bildschirmgröße erfolgt nur bei einer Verbindung mit dem Programm UniTerminal, welches aktuell nur für macOS verfügbar ist.

Bei anderen Terminalprogrammen wie dem Arduino IDE Seriellen Monitor oder Zterm wird bei Eingabe von "`ttttt`" das TTY-Protokoll angenommen sowie bei Eingabe von "`vvvvv`" das DEC VT52 Protokoll und die Bildschirmgröße auf 80 x 24 Zeichen gesetzt.

BRK-Clock Library Listing

Die BRK-Clock Library stellt Befehle zur Verfügung, um eine Wort-Uhr aus 8 x 8 programmierbaren RGBW-Bricks zu programmieren. Aktuell wird die deutsche Version der BRK-Clock unterstützt. Die dazu benötigten, landesspezifischen Befehle sind in deutscher Nomenklatur gehalten. Eine Erweiterung mit anderen Sprachen ist dadurch problemlos möglich. Hardwarespezifische Basis dieser Library ist die NeoPixel Library, die NeoMatrix Library und die GFX Library von Adafruit, die ebenfalls geladen sein müssen. Auf Grund der Größe und Komplexität dieser Libraries macht es wenig Sinn, diese komplett als Quelltext in die BRK-Clock Library zu integrieren. Im Programm muß vor dem `#include` der Library die Datenleitung, z.B. GPIO13, mit dem Befehl `#define BRKclock_PIN 13` definiert werden.

```
// BRK Clock Library
// 1.00 - 2017-06-04
// 1.01 - 2017-06-16
// 1.02 - 2017-06-23
// 1.03 - 2017-06-30 - English Language
// (c) Copyright 2017
//
// Zur Verwendung mit dem Allnet IoT-Brick

#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>

#include <all_Sound.h>

#define LED 64  // Anzahl LEDS

//  Aufbau der 8 x 8 LED-Brick-Matrix:

//  F(63) Ü(62) N(61) F(60) Z(59) E(58) H(57) N(56)

//  V(48) O(49) R(50) p(51) N(52) A(53) C(54) H(55)

//  H(47) A(46) L(45) B(44) V(43) I(42) E(41) R(40)

//  E(32) I(33) N(34) S(35) E(36) C(37) H(38) S(39)

//  S(31) I(30) E(29) Z(28) W(27) Ö(26) L(25) F(24)

//  B(16) E(17) N(18) D(19) R(20) E(21) I(22) Ü(23)

//  R(15) Z(14) E(13) H(12) N(11) E(10) U(09) N(08)

//  K(00) A(01) C(02) H(03) T(04) E(05) L(06) F(07)
```

```
// *** Globale Typen ***

struct BRKclock_Parameter
{
    //    32 Bytes Parameter-Header:
    int64 parameterID      = 0;                    // Kennung des Parameter-Blocks "BRKClock" im Klartext
    int64 parameterHash    = 0;                    // Kennung des Parameter-Blocks "BRKClock" als verschlüsselter Hash
    int32 parameterSize    = 0;                    // Anzahl Bytes im Parameter Block
    int32 parameterReserve = 0;                    // Platzhalter für weiteren Header-Eintrag
    int32 parameterVersion = 0;                    // Version des Parameter-Blocks
    int32 parameterCount   = 0;                    // Anzahl folgenden Byte-Parameter im Block

    //    Start des Parameter-Blocks:
    byte  clockColorRed    = 0;                    // RGBW-Farbe der Uhrzeitanzeige
    byte  clockColorGreen  = 0;
    byte  clockColorBlue   = 0;
    byte  clockColorWhite  = 0;
    byte  tempColorRed     = 0;                    // RGBW-Farbe der Temperaturanzeige
    byte  tempColorGreen   = 0;
    byte  tempColorBlue    = 0;
    byte  tempColorWhite   = 0;
    byte  IP_address0      = 0;                    // 4 Bytes IP-Adresse
    byte  IP_address1      = 0;
    byte  IP_address2      = 0;
    byte  IP_address3      = 0;
    byte  IP_gateway0      = 0;                    // 4 Bytes Gateway
    byte  IP_gateway1      = 0;
    byte  IP_gateway2      = 0;
    byte  IP_gateway3      = 0;
    byte  IP_subnet0       = 0;                    // 4 Bytes Subnetzmaske
    byte  IP_subnet1       = 0;
    byte  IP_subnet2       = 0;
    byte  IP_subnet3       = 0;
    byte  staticIP         = 0;                    // 0 = Dynamische IP-Adresse, 1 = Statische IP-Adresse
    byte  alarmMode        = 0;                    // 0 = Wecker aus,    1 = Wecker ein
    byte  temperatureMode  = 0;                    // 0 = Temperaturanzeige aus,   1 = Temperaturanzeige jede Minute
    byte  colorMode        = 0;                    // 0 = Random Schrift, 1 = Weiße Schrift, 2 = Rote Schrift, usw.
    byte  buttonMode       = 0;                    // 0 = IP-Adresse anzeigen, 1 = Temperatur anzeigen, usw.
    byte  alarmHour        = 0;                    // Weckzeit: Stunden
    byte  alarmMinute      = 0;                    // Weckzeit: Minuten
};


// *** Globale Konstanten ***

#if !defined(BRKclock_PIN)                         // Wenn BRKclock_PIN nicht definiert wurde
#define BRKclock_PIN 14                           // Standard LED Pin GPIO14 benutzen
#endif
```

```
// *** Globale Variablen ***

BRKclock_Parameter BRKclock_ParameterData;

Adafruit_NeoPixel  BRKclock_Pixel = Adafruit_NeoPixel(LED, BRKclock_PIN, NEO_RGBW + NEO_KHZ800);

Adafruit_NeoMatrix BRKclock_Matrix = Adafruit_NeoMatrix(8, 8, BRKclock_PIN,
                                                        NEO_MATRIX_BOTTOM + NEO_MATRIX_LEFT +
                                                        NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,
                                                        NEO_GRBW + NEO_KHZ800);
// *** Farbe der Uhrzeit-Anzeige ***

uint32_t BRKclock_clockColor = BRKclock_Pixel.Color(0x00, 0xFF, 0x00, 0); // GRBW (grün, rot, blau, weiß), 32 Bit
byte     BRKclock_clockColorRed       = 0xFF;                             // Farbanteil für Rot-Kanal
byte     BRKclock_clockColorGreen     = 0x00;                             // Farbanteil für Grün-Kanal
byte     BRKclock_clockColorBlue      = 0x00;                             // Farbanteil für Blau-Kanal
byte     BRKclock_clockColorWhite     = 0x00;                             // Farbanteil für Weiß-Kanal


// *** Farbe der Temperatur-Anzeige ***

uint32_t BRKclock_tempColor  = BRKclock_Pixel.Color(0xFF, 0x00, 0x00, 0); // RGBW (rot, grün, blau, weiß), 32 Bit
byte     BRKclock_tempColorRed        = 0xFF;                             // Farbanteil für Rot-Kanal
byte     BRKclock_tempColorGreen      = 0x00;                             // Farbanteil für Grün-Kanal
byte     BRKclock_tempColorBlue       = 0x00;                             // Farbanteil für Blau-Kanal
byte     BRKclock_tempColorWhite      = 0x00;                             // Farbanteil für Weiß-Kanal


// *** Benutzerdefinierte Einstellungen ***

byte     BRKclock_staticIP            = 0;                                // 0 = Dynamische IP-Adresse, 1 = Statische IP-Adresse
int      BRKclock_alarmMode           = 0;                                // 0 = Wecker aus, 1 = Wecker ein
int      BRKclock_temperatureMode     = 1;                                // 0 = Temperaturanzeige aus, 1 = Temperaturanzeige jede Minute
int      BRKclock_colorMode           = 2;                                // 0 = Random Schrift, 1 = Weiße Schrift, 2 = Rote Schrift, usw.
int      BRKclock_buttonMode          = 0;                                // 0 = IP-Adresse anzeigen, 1 = Temperatur anzeigen, usw.
byte     BRKclock_alarmHour           = 0;                                // Weckzeit: Stunden
byte     BRKclock_alarmMinute         = 0;                                // Weckzeit: Minuten


// *** Globale Konstanten für spezielle 16 Bit Farbtabellen im RGB-Modus ***

const uint16_t BRKclock_Matrix16Color [ ] = {BRKclock_Matrix.Color(0x00, 0x00, 0x00),  //  0 = Schwarz
                                             BRKclock_Matrix.Color(0xFF, 0x00, 0x00),  //  1 = Rot
                                             BRKclock_Matrix.Color(0x00, 0x00, 0x80),  //  2 = Dunkelblau
                                             BRKclock_Matrix.Color(0xFF, 0x00, 0xFF),  //  3 = Purpur (Magenta)
                                             BRKclock_Matrix.Color(0x00, 0x80, 0x00),  //  4 = Dunkelgrün
                                             BRKclock_Matrix.Color(0x54, 0x54, 0x54),  //  5 = Dunkelgrau
                                             BRKclock_Matrix.Color(0x00, 0x00, 0xFF),  //  6 = Blau
```

```
                                        BRKclock_Matrix.Color(0x00, 0x80, 0xFF),  //  7 = Hellblau
                                        BRKclock_Matrix.Color(0x80, 0x40, 0x00),  //  8 = Braun
                                        BRKclock_Matrix.Color(0xFF, 0x80, 0x00),  //  9 = Orange
                                        BRKclock_Matrix.Color(0xA8, 0xA8, 0xA8),  // 10 = Hellgrau
                                        BRKclock_Matrix.Color(0xFF, 0x80, 0x80),  // 11 = Pink
                                        BRKclock_Matrix.Color(0x00, 0xFF, 0x00),  // 12 = Grün
                                        BRKclock_Matrix.Color(0xFF, 0xFF, 0x00),  // 13 = Gelb
                                        BRKclock_Matrix.Color(0x00, 0xFF, 0xFF),  // 14 = Türkis (Cyan)
                                        BRKclock_Matrix.Color(0xFF, 0xFF, 0xFF)}; // 15 = Weiß

const uint16_t BRKclock_Matrix8Color  [ ] = {BRKclock_Matrix.Color(0x00, 0x00, 0x00),  //  0 = Schwarz
                                        BRKclock_Matrix.Color(0xFF, 0x00, 0x00),  //  1 = Rot
                                        BRKclock_Matrix.Color(0xFF, 0x00, 0xFF),  //  2 = Purpur (Magenta)
                                        BRKclock_Matrix.Color(0x00, 0x00, 0xFF),  //  3 = Blau
                                        BRKclock_Matrix.Color(0x00, 0xFF, 0x00),  //  4 = Grün
                                        BRKclock_Matrix.Color(0xFF, 0xFF, 0x00),  //  5 = Gelb
                                        BRKclock_Matrix.Color(0x00, 0xFF, 0xFF),  //  6 = Türkis (Cyan)
                                        BRKclock_Matrix.Color(0xFF, 0xFF, 0xFF)}; //  7 = Weiß


// *** Globale Konstante für Gamma-Wert (Helligkeitskorrektur) ***

const byte BRKclock_Gamma             [ ] = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                                               0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
                                               1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,
                                               2,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,
                                               5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  9,  9,  9, 10,
                                              10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
                                              17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
                                              25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
                                              37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
                                              51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
                                              69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
                                              90, 92, 93, 95, 96, 98, 99,101,102,104,105,107,109,110,112,114,
                                             115,117,119,120,122,124,126,127,129,131,133,135,137,138,140,142,
                                             144,146,148,150,152,154,156,158,160,162,164,167,169,171,173,175,
                                             177,180,182,184,186,189,191,193,196,198,200,203,205,208,210,213,
                                             215,218,220,223,225,228,231,233,236,239,241,244,247,249,252,255  };


// *** BRK-Clock Grundlagen ***

void BRKclock_SetTemperatureRGBWColor (byte red, byte green, byte blue, byte white)
{
    Serial.println("Set Temp. Color: R = " + hexbyte(red) + ", G = " + hexbyte(green) + ", B = " + hexbyte(blue) + ", W = " + hexbyte(white));
    BRKclock_tempColorRed   = red;
    BRKclock_tempColorGreen = green;
    BRKclock_tempColorBlue  = blue;
    BRKclock_tempColorWhite = white;
    if (red != 0)
```

```
    {
        const byte min = 24;
        red   = byte(((double)red   / 255.0) * (255.0 - (double)min)) + min;    // Rote LED zündet erst bei 24
    }
    if (green != 0)
    {
        const byte min = 12;
        green = byte(((double)green / 255.0) * (255.0 - (double)min)) + min;    // Grüne LED zündet erst bei 12
    }
    if (blue != 0)
    {
        const byte min = 24;
        blue  = byte(((double)blue  / 255.0) * (255.0 - (double)min)) + min;    // Blaue LED zündet erst bei 24
    }
    if (white != 0)
    {
        const byte min = 3;
        white = byte(((double)white / 255.0) * (255.0 - (double)min)) + min;    // Weiße LED zündet erst bei 3
    }
    BRKclock_tempColor = BRKclock_Pixel.Color(red, green, blue, white);         // (rot, grün, blau, weiß)
    BRKclock_Matrix.setPassThruColor(BRKclock_Pixel.Color(red, green, blue, white)); // In den RGBW-Modus wechseln
}

void BRKclock_SetTemperatureRGBColor (byte red, byte green, byte blue)
{
    Serial.println("Set Temp. Color: R = " + hexbyte(red) + ", G = " + hexbyte(green) + ", B = " + hexbyte(blue));
    BRKclock_tempColorRed   = red;
    BRKclock_tempColorGreen = green;
    BRKclock_tempColorBlue  = blue;
    if (red != 0)
    {
        red   = byte(((double)red   / 255.0) * 231.0) + 24;         // Rote LED zündet erst bei 24
    }
    if (green != 0)
    {
        green = byte(((double)green / 255.0) * 243.0) + 12;         // Grüne LED zündet erst bei 12
    }
    if (blue != 0)
    {
        blue  = byte(((double)blue  / 255.0) * 231.0) + 24;         // Blaue LED zündet erst bei 24
    }
    BRKclock_tempColor = BRKclock_Matrix.Color(red, green, blue);   // (rot, grün, blau)
    BRKclock_Matrix.setPassThruColor();                             // In den RGB-Modus wechseln
}

void BRKclock_SetClockRGBWColor (byte red, byte green, byte blue, byte white)
{
    Serial.println("Set Clock Color: R = " + hexbyte(red) + ", G = " + hexbyte(green) + ", B = " + hexbyte(blue) + ", W = " + hexbyte(white));
    BRKclock_clockColorRed   = red;
    BRKclock_clockColorGreen = green;
```

```
    BRKclock_clockColorBlue  = blue;
    BRKclock_clockColorWhite = white;
    BRKclock_clockColor = BRKclock_Pixel.Color(green, red, blue, white); // (grün, rot, blau, weiß)
}

void BRKclock_Clear (bool RGBW)                                     // Wahlweise RGB- oder RGBW-Modus
{
    BRKclock_Pixel.clear();                                         // Pixel löschen
    BRKclock_Pixel.show();                                          // Anzeige aktualisieren
    if (RGBW)
    {
        BRKclock_Matrix.setPassThruColor();                         // In den RGB-Modus wechseln
    }
    BRKclock_Matrix.fillScreen(0);                                  // Alle Pixel ausschalten (schwarz)
    if (RGBW)
    {
        BRKclock_SetTemperatureRGBWColor(BRKclock_tempColorRed,     // Alte RGBW-Farbe/Modus wiederherstellen
                                         BRKclock_tempColorGreen,
                                         BRKclock_tempColorBlue,
                                         BRKclock_tempColorWhite);
    }
    else
    {
        BRKclock_SetTemperatureRGBColor(BRKclock_tempColorRed,      // Alte RGB-Farbe/Modus wiederherstellen
                                        BRKclock_tempColorGreen,
                                        BRKclock_tempColorBlue);
    }
}

void BRKclock_Init ()
{
    BRKclock_Pixel.begin();
    BRKclock_Pixel.setBrightness(255);                              // Helligkeit (0..255)
    BRKclock_Clear(true);                                           // Pixel löschen & Anzeige aktualisieren, RGBW-Modus
    BRKclock_Matrix.begin();                                        // Ausgabe auf der Matrix vorbereiten (für die Demos)
    BRKclock_Matrix.setTextWrap(false);                             // Kein Zeilenumbruch
    BRKclock_Matrix.setTextSize(1);                                 // Normale Textgröße, keine Vergrößerung
    BRKclock_Matrix.setBrightness(100);                             // Helligkeit 100%
}

void BRKclock_ShowBRK ()
{
    BRKclock_Pixel.clear();                                         // Pixel löschen und "BRK" anzeigen
    BRKclock_Pixel.setPixelColor(0, 250, 0, 0);                     // "B" in Blau
    BRKclock_Pixel.setPixelColor(15, 0, 250, 0);                    // "R" in Rot
    BRKclock_Pixel.setPixelColor(16, 0, 0, 250);                    // "K" in Grün
    BRKclock_Pixel.show();                                          // Anzeige aktualisieren
}

void BRKclock_WLANautoConnect (bool useSavedSettings)
```

```
{
    BRKclock_ShowBRK();
    IoT_WLANautoConnect(useSavedSettings);                          // Verbindung, ggf. gespeicherte, über WLAN herstellen
}

void BRKclock_FullColor (uint32_t c)
{
    for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
    {
        BRKclock_Pixel.setPixelColor(i, c);
    }
    BRKclock_Pixel.show();
}

uint32_t BRKclock_clockColorWheel (byte angle)   // Einen Winkel von 0-255 aus dem Farbkreis auswählen
{                                               // Die Farben sind eine Abbildung von r-g-b-r
    angle = 255 - (angle & 255);                  // Es wird der zugehörige Farbwert als uint32_t ermittelt
    if (angle < 85)
    {
        return BRKclock_Pixel.Color(255 - angle * 3, 0, angle * 3);
    }
    if(angle < 170)
    {
        angle -= 85;
        return BRKclock_Pixel.Color(0, angle * 3, 255 - angle * 3);
    }
    angle -= 170;
    return BRKclock_Pixel.Color(angle * 3, 255 - angle * 3, 0);
}

void BRKclock_SetClockRandomColor ()
{
    byte c = rnd(14) + 1;                                           // Indizierte Farbe zwischen 1 und 15 (ohne schwarz)
    Serial.print("Random Color #" + str(c) + " - ");
    BRKclock_SetClockRGBWColor(t_RGBColor16ValueRed[c], t_RGBColor16ValueGreen[c], t_RGBColor16ValueBlue[c], 0);
}

void BRKclock_TestMatrix (int waitTime)
{
    float factor = 256.0 / float(BRKclock_Pixel.numPixels());      // Multiplikationsfaktor, um alle Farben zu zeigen
    IoT_DisplayClear(16);                                          // OLED Display löschen (16 Punkt Schrift)
    IoT_DisplayAlignText(TEXT_ALIGN_CENTER);                       // Zentrierte Darstellung des Textes
    IoT_DisplayDrawText(64, 25, "BRK Clock");
    IoT_DisplayUpdate();
    for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)           // LED Test -> Grün - Rot - Blau - Grün
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColorWheel((int(float(i) * factor)) & 255));
        BRKclock_Pixel.show();
        delay(waitTime);
    }
```

```
}

void BRKclock_Plot8color (int x, int y, int color)
{
    BRKclock_Matrix.setPassThruColor();                             // In den RGB-Modus wechseln
    BRKclock_Matrix.drawRect(x, y, 1, 1, BRKclock_Matrix8Color[color & 7]);
}

void BRKclock_Plot16color (int x, int y, int color)
{
    BRKclock_Matrix.setPassThruColor();                             // In den RGB-Modus wechseln
    BRKclock_Matrix.drawRect(x, y, 1, 1, BRKclock_Matrix16Color[color & 15]);
}

void BRKclock_Plot (int x, int y)
{
    BRKclock_Matrix.drawRect(x, y, 1, 1, BRKclock_tempColor);
}

void BRKclock_DrawNumber (int x, int y, byte num)
{
    switch (num)
    {
        case 0:
            BRKclock_Plot(x,     y);
            BRKclock_Plot(x + 1, y);
            BRKclock_Plot(x + 2, y);
            BRKclock_Plot(x,     y + 1);
            BRKclock_Plot(x + 2, y + 1);
            BRKclock_Plot(x,     y + 2);
            BRKclock_Plot(x + 2, y + 2);
            BRKclock_Plot(x,     y + 3);
            BRKclock_Plot(x + 2, y + 3);
            BRKclock_Plot(x,     y + 4);
            BRKclock_Plot(x + 1, y + 4);
            BRKclock_Plot(x + 2, y + 4);
            break;
        case 1:
            BRKclock_Plot(x + 1, y);
            BRKclock_Plot(x + 1, y + 1);
            BRKclock_Plot(x + 1, y + 2);
            BRKclock_Plot(x + 1, y + 3);
            BRKclock_Plot(x + 1, y + 4);
            break;
        case 2:
            BRKclock_Plot(x,     y);
            BRKclock_Plot(x + 1, y);
            BRKclock_Plot(x + 2, y);
            BRKclock_Plot(x + 2, y + 1);
            BRKclock_Plot(x,     y + 2);
```

```
    BRKclock_Plot(x + 1, y + 2);
    BRKclock_Plot(x + 2, y + 2);
    BRKclock_Plot(x,     y + 3);
    BRKclock_Plot(x,     y + 4);
    BRKclock_Plot(x + 1, y + 4);
    BRKclock_Plot(x + 2, y + 4);
    break;
case 3:
    BRKclock_Plot(x,     y);
    BRKclock_Plot(x + 1, y);
    BRKclock_Plot(x + 2, y);
    BRKclock_Plot(x + 2, y + 1);
    BRKclock_Plot(x + 1, y + 2);
    BRKclock_Plot(x + 2, y + 2);
    BRKclock_Plot(x + 2, y + 3);
    BRKclock_Plot(x,     y + 4);
    BRKclock_Plot(x + 1, y + 4);
    BRKclock_Plot(x + 2, y + 4);
    break;
case 4:
    BRKclock_Plot(x,     y);
    BRKclock_Plot(x + 2, y);
    BRKclock_Plot(x,     y + 1);
    BRKclock_Plot(x + 2, y + 1);
    BRKclock_Plot(x,     y + 2);
    BRKclock_Plot(x + 1, y + 2);
    BRKclock_Plot(x + 2, y + 2);
    BRKclock_Plot(x + 2, y + 3);
    BRKclock_Plot(x + 2, y + 4);
    break;
case 5:
    BRKclock_Plot(x,     y);
    BRKclock_Plot(x + 1, y);
    BRKclock_Plot(x + 2, y);
    BRKclock_Plot(x,     y + 1);
    BRKclock_Plot(x,     y + 2);
    BRKclock_Plot(x + 1, y + 2);
    BRKclock_Plot(x + 2, y + 2);
    BRKclock_Plot(x + 2, y + 3);
    BRKclock_Plot(x,     y + 4);
    BRKclock_Plot(x + 1, y + 4);
    BRKclock_Plot(x + 2, y + 4);
    break;
case 6:
    BRKclock_Plot(x,     y);
    BRKclock_Plot(x + 1, y);
    BRKclock_Plot(x + 2, y);
    BRKclock_Plot(x,     y + 1);
    BRKclock_Plot(x,     y + 2);
    BRKclock_Plot(x + 1, y + 2);
```

```
            BRKclock_Plot(x + 2, y + 2);
            BRKclock_Plot(x,     y + 3);
            BRKclock_Plot(x + 2, y + 3);
            BRKclock_Plot(x,     y + 4);
            BRKclock_Plot(x + 1, y + 4);
            BRKclock_Plot(x + 2, y + 4);
            break;
        case 7:
            BRKclock_Plot(x,     y);
            BRKclock_Plot(x + 1, y);
            BRKclock_Plot(x + 2, y);
            BRKclock_Plot(x + 2, y + 1);
            BRKclock_Plot(x + 2, y + 2);
            BRKclock_Plot(x + 2, y + 3);
            BRKclock_Plot(x + 2, y + 4);
            break;
        case 8:
            BRKclock_Plot(x,     y);
            BRKclock_Plot(x + 1, y);
            BRKclock_Plot(x + 2, y);
            BRKclock_Plot(x,     y + 1);
            BRKclock_Plot(x + 2, y + 1);
            BRKclock_Plot(x,     y + 2);
            BRKclock_Plot(x + 1, y + 2);
            BRKclock_Plot(x + 2, y + 2);
            BRKclock_Plot(x,     y + 3);
            BRKclock_Plot(x + 2, y + 3);
            BRKclock_Plot(x,     y + 4);
            BRKclock_Plot(x + 1, y + 4);
            BRKclock_Plot(x + 2, y + 4);
            break;
        case 9:
            BRKclock_Plot(x,     y);
            BRKclock_Plot(x + 1, y);
            BRKclock_Plot(x + 2, y);
            BRKclock_Plot(x,     y + 1);
            BRKclock_Plot(x + 2, y + 1);
            BRKclock_Plot(x,     y + 2);
            BRKclock_Plot(x + 1, y + 2);
            BRKclock_Plot(x + 2, y + 2);
            BRKclock_Plot(x + 2, y + 3);
            BRKclock_Plot(x,     y + 4);
            BRKclock_Plot(x + 1, y + 4);
            BRKclock_Plot(x + 2, y + 4);
            break;
        default:
            break;
    }
}
```

```
void BRKclock_Print (String s, int pause)                           // Laufschrift
{
    BRKclock_Clear(true);                                             // Pixel löschen & Anzeige aktualisieren
    int x = BRKclock_Matrix.width();                                  // Breite der Matrix (8)
    IoT_WaitMessage();                                                // Mitteilung anzeigen
    while (not(IoT_Keypress()))
    {
        int L = (len(s) - 1) * 8;                                       // Breite der Laufschrift insgesamt in Pixel
        BRKclock_Clear(true);                                           // Pixel löschen & Anzeige aktualisieren

        BRKclock_Matrix.setCursor(x, 0);                                // Von unten rechts reinlaufen, x verringert sich im loop()
        BRKclock_Matrix.print(s);                                       // Laufschrift "Brick'R'knowledge" loslassen...
        if (--x < -L)
        {
            break;
        }
        BRKclock_Matrix.show();
        IoT_WaitKeypress(pause, false);                                 // 100 ms. oder auf Taster warten
    }
}

void BRKclock_PrintTemperature (int temperature, int pause)
{
    if ((temperature < 0) || (temperature > 99))
    {
        temperature = 0;
    }
    BRKclock_Clear(true);                                             // Pixel löschen & Anzeige aktualisieren
    BRKclock_DrawNumber(0, 2, temperature / 10);                      // Zehner
    BRKclock_DrawNumber(4, 2, temperature % 10);                      // Einer
    BRKclock_Plot(7, 1);                                              // Grad
    BRKclock_Matrix.show();                                           // Zahl zeigen
    IoT_WaitKeypress(pause, true);                                    // 10 Sekunden. oder auf Taster warten
}


// *** BRK-Clock Parameter für das EEPROM des ESP8266 ***

bool BRKclock_ParameterValidate ()
{
    if ((BRKclock_ParameterData.parameterID == crc64("BRKClock", 0)) &&
        (BRKclock_ParameterData.parameterHash == xorshift128plus("BRKClock", 0)))
    {
        return (true);                                                                  // Parameter sind gültig
    }
    else
    {
        return (false);                                                                 // Ungültige Parameter
    }
}
```

```
bool BRKclock_ParameterApply ()
{
    if (BRKclock_ParameterValidate())
    {
        BRKclock_clockColorRed = BRKclock_ParameterData.clockColorRed;            // RGBW-Farbe der Uhrzeitanzeige
        BRKclock_clockColorGreen = BRKclock_ParameterData.clockColorGreen;
        BRKclock_clockColorBlue = BRKclock_ParameterData.clockColorBlue;
        BRKclock_clockColorWhite = BRKclock_ParameterData.clockColorWhite;
        BRKclock_SetClockRGBWColor(BRKclock_clockColorRed, BRKclock_clockColorGreen, BRKclock_clockColorBlue, BRKclock_clockColorWhite); //
Schriftfarbe
        BRKclock_tempColorRed = BRKclock_ParameterData.tempColorRed;              // RGBW-Farbe der Temperaturanzeige
        BRKclock_tempColorGreen = BRKclock_ParameterData.tempColorGreen;
        BRKclock_tempColorBlue = BRKclock_ParameterData.tempColorBlue;
        BRKclock_tempColorWhite = BRKclock_ParameterData.tempColorWhite;
        BRKclock_SetTemperatureRGBWColor(BRKclock_tempColorRed, BRKclock_tempColorGreen, BRKclock_tempColorBlue, BRKclock_tempColorWhite);
        if (BRKclock_ParameterData.staticIP == 1)
        {
            WiFi.localIP()[0] = BRKclock_ParameterData.IP_address0;                 // 4 Bytes IP-Adresse
            WiFi.localIP()[1] = BRKclock_ParameterData.IP_address1;
            WiFi.localIP()[2] = BRKclock_ParameterData.IP_address2;
            WiFi.localIP()[3] = BRKclock_ParameterData.IP_address3;
            WiFi.gatewayIP()[0] = BRKclock_ParameterData.IP_gateway0;               // 4 Bytes Gateway
            WiFi.gatewayIP()[1] = BRKclock_ParameterData.IP_gateway1;
            WiFi.gatewayIP()[2] = BRKclock_ParameterData.IP_gateway2;
            WiFi.gatewayIP()[3] = BRKclock_ParameterData.IP_gateway3;
            WiFi.subnetMask()[0] = BRKclock_ParameterData.IP_subnet0;               // 4 Bytes Subnetzmaske
            WiFi.subnetMask()[1] = BRKclock_ParameterData.IP_subnet1;
            WiFi.subnetMask()[2] = BRKclock_ParameterData.IP_subnet2;
            WiFi.subnetMask()[3] = BRKclock_ParameterData.IP_subnet3;
            IPAddress ip1 = IPAddress(BRKclock_ParameterData.IP_address0,
                                      BRKclock_ParameterData.IP_address1,
                                      BRKclock_ParameterData.IP_address2,
                                      BRKclock_ParameterData.IP_address3);
            IPAddress ip2 = IPAddress(BRKclock_ParameterData.IP_gateway0,
                                      BRKclock_ParameterData.IP_gateway1,
                                      BRKclock_ParameterData.IP_gateway2,
                                      BRKclock_ParameterData.IP_gateway3);
            IPAddress ip3 = IPAddress(BRKclock_ParameterData.IP_subnet0,
                                      BRKclock_ParameterData.IP_subnet1,
                                      BRKclock_ParameterData.IP_subnet2,
                                      BRKclock_ParameterData.IP_subnet3);
            WiFi.config(ip1, ip2, ip3);                                             // Parameter: IP, Gateway, Subnet, DNS
        }
        BRKclock_staticIP = BRKclock_ParameterData.staticIP;                      // 0 = Dynamische IP-Adresse, 1 = Statische IP-Adresse
        BRKclock_alarmMode = BRKclock_ParameterData.alarmMode;                    // 0 = Wecker aus,    1 = Wecker ein
        BRKclock_temperatureMode = BRKclock_ParameterData.temperatureMode;        // 0 = Temperaturanzeige aus,   1 = Temperaturanzeige jede Minute
        BRKclock_colorMode = BRKclock_ParameterData.colorMode;                    // 0 = Random Schrift, 1 = Weiße Schrift, 2 = Rote Schrift, usw.
        BRKclock_buttonMode = BRKclock_ParameterData.buttonMode;                  // 0 = IP-Adresse anzeigen, 1 = Temperatur anzeigen, usw.
        BRKclock_alarmHour = BRKclock_ParameterData.alarmHour;                    // Weckzeit: Stunden
```

```
        BRKclock_alarmMinute = BRKclock_ParameterData.alarmMinute;              // Weckzeit: Minuten
        BRKclock_language = BRKclock_ParameterData.language;                    // Sprache: 0 = Deutsch, 1 = Englisch
        return (true);                                                          // Parameter-Block ist gültig und wurde verwendet
    }
    else
    {
        return (false);                                                         // Parameter-Block ist ungültig und wurde nicht verwendet
    }
}

void BRKclock_ParameterCreate ()
{
    //   32 Bytes Parameter-Header:
    BRKclock_ParameterData.parameterID      = crc64("BRKClock", 0);             // Kennung des Parameter-Blocks "BRKClock" im Klartext
    BRKclock_ParameterData.parameterHash    = xorshift128plus("BRKClock", 0);   // Kennung des Parameter-Blocks "BRKClock" als verschlüsselter Hash
    BRKclock_ParameterData.parameterSize    = sizeof(BRKclock_Parameter);       // Anzahl Bytes im Parameter Block
    BRKclock_ParameterData.parameterReserve = 0;                                // Platzhalter für weiteren Header-Eintrag
    BRKclock_ParameterData.parameterVersion = 1;                                // Version des Parameter-Blocks
    BRKclock_ParameterData.parameterCount   = 26;                               // Anzahl folgenden Byte-Parameter im Block

    //   Start des Parameter-Blocks:
    BRKclock_ParameterData.clockColorRed    = BRKclock_clockColorRed;           // RGBW-Farbe der Uhrzeitanzeige
    BRKclock_ParameterData.clockColorGreen  = BRKclock_clockColorGreen;
    BRKclock_ParameterData.clockColorBlue   = BRKclock_clockColorBlue;
    BRKclock_ParameterData.clockColorWhite  = BRKclock_clockColorWhite;
    BRKclock_ParameterData.tempColorRed     = BRKclock_tempColorRed;            // RGBW-Farbe der Temperaturanzeige
    BRKclock_ParameterData.tempColorGreen   = BRKclock_tempColorGreen;
    BRKclock_ParameterData.tempColorBlue    = BRKclock_tempColorBlue;
    BRKclock_ParameterData.tempColorWhite   = BRKclock_tempColorWhite;
    BRKclock_ParameterData.IP_address0      = WiFi.localIP()[0];                // 4 Bytes IP-Adresse
    BRKclock_ParameterData.IP_address1      = WiFi.localIP()[1];
    BRKclock_ParameterData.IP_address2      = WiFi.localIP()[2];
    BRKclock_ParameterData.IP_address3      = WiFi.localIP()[3];
    BRKclock_ParameterData.IP_gateway0      = WiFi.gatewayIP()[0];              // 4 Bytes Gateway
    BRKclock_ParameterData.IP_gateway1      = WiFi.gatewayIP()[1];
    BRKclock_ParameterData.IP_gateway2      = WiFi.gatewayIP()[2];
    BRKclock_ParameterData.IP_gateway3      = WiFi.gatewayIP()[3];
    BRKclock_ParameterData.IP_subnet0       = WiFi.subnetMask()[0];             // 4 Bytes Subnetzmaske
    BRKclock_ParameterData.IP_subnet1       = WiFi.subnetMask()[1];
    BRKclock_ParameterData.IP_subnet2       = WiFi.subnetMask()[2];
    BRKclock_ParameterData.IP_subnet3       = WiFi.subnetMask()[3];
    BRKclock_ParameterData.staticIP         = BRKclock_staticIP;                // 0 = Dynamische IP-Adresse, 1 = Statische IP-Adresse
    BRKclock_ParameterData.alarmMode        = BRKclock_alarmMode;               // 0 = Wecker aus,    1 = Wecker ein
    BRKclock_ParameterData.temperatureMode  = BRKclock_temperatureMode;         // 0 = Temperaturanzeige aus,   1 = Temperaturanzeige jede Minute
    BRKclock_ParameterData.colorMode        = BRKclock_colorMode;               // 0 = Random Schrift, 1 = Weiße Schrift, 2 = Rote Schrift, usw.
    BRKclock_ParameterData.buttonMode       = BRKclock_buttonMode;              // 0 = IP-Adresse anzeigen, 1 = Temperatur anzeigen, usw.
    BRKclock_ParameterData.alarmHour        = BRKclock_alarmHour;               // Weckzeit: Stunden
    BRKclock_ParameterData.alarmMinute      = BRKclock_alarmMinute;             // Weckzeit: Minuten
    BRKclock_ParameterData.language         = BRKclock_language;                // Sprache: 0 = Deutsch, 1 = Englisch
}
```

```
void BRKclock_ParameterNew ()
{
    //   32 Bytes Parameter-Header:
    BRKclock_ParameterData.parameterID      = crc64("BRKClock", 0);             // Kennung des Parameter-Blocks "BRKClock" im Klartext
    BRKclock_ParameterData.parameterHash    = xorshift128plus("BRKClock", 0);   // Kennung des Parameter-Blocks "BRKClock" als verschlüsselter Hash
    BRKclock_ParameterData.parameterSize    = sizeof(BRKclock_Parameter);       // Anzahl Bytes im Parameter Block
    BRKclock_ParameterData.parameterReserve = 0;                                // Platzhalter für weiteren Header-Eintrag
    BRKclock_ParameterData.parameterVersion = 1;                                // Version des Parameter-Blocks
    BRKclock_ParameterData.parameterCount   = 26;                               // Anzahl folgenden Byte-Parameter im Block

    //   Start des Parameter-Blocks:
    BRKclock_ParameterData.clockColorRed    = 0xFF;                             // RGBW-Farbe der Uhrzeitanzeige
    BRKclock_ParameterData.clockColorGreen  = 0x00;
    BRKclock_ParameterData.clockColorBlue   = 0x00;
    BRKclock_ParameterData.clockColorWhite  = 0x00;
    BRKclock_ParameterData.tempColorRed     = 0xFF;                             // RGBW-Farbe der Temperaturanzeige
    BRKclock_ParameterData.tempColorGreen   = 0x00;
    BRKclock_ParameterData.tempColorBlue    = 0x00;
    BRKclock_ParameterData.tempColorWhite   = 0x00;
    BRKclock_ParameterData.IP_address0      = 0;                                // 4 Bytes IP-Adresse
    BRKclock_ParameterData.IP_address1      = 0;
    BRKclock_ParameterData.IP_address2      = 0;
    BRKclock_ParameterData.IP_address3      = 0;
    BRKclock_ParameterData.IP_gateway0      = 0;                                // 4 Bytes Gateway
    BRKclock_ParameterData.IP_gateway1      = 0;
    BRKclock_ParameterData.IP_gateway2      = 0;
    BRKclock_ParameterData.IP_gateway3      = 0;
    BRKclock_ParameterData.IP_subnet0       = 0;                                // 4 Bytes Subnetzmaske
    BRKclock_ParameterData.IP_subnet1       = 0;
    BRKclock_ParameterData.IP_subnet2       = 0;
    BRKclock_ParameterData.IP_subnet3       = 0;
    BRKclock_ParameterData.staticIP         = 0;                                // 0 = Dynamische IP-Adresse, 1 = Statische IP-Adresse
    BRKclock_ParameterData.alarmMode        = 0;                                // 0 = Wecker aus,    1 = Wecker ein
    BRKclock_ParameterData.temperatureMode  = 1;                                // 0 = Temperaturanzeige aus,   1 = Temperaturanzeige jede Minute
    BRKclock_ParameterData.colorMode        = 2;                                // 0 = Random Schrift, 1 = Weiße Schrift, 2 = Rote Schrift, usw.
    BRKclock_ParameterData.buttonMode       = 0;                                // 0 = IP-Adresse anzeigen, 1 = Temperatur anzeigen, usw.
    BRKclock_ParameterData.alarmHour        = 0;                                // Weckzeit: Stunden
    BRKclock_ParameterData.alarmMinute      = 0;                                // Weckzeit: Minuten
    BRKclock_ParameterData.language         = 0;                                // Sprache: 0 = Deutsch, 1 = Englisch
}

bool BRKclock_ParameterLoad ()
{
    EEPROM.get(0, BRKclock_ParameterData); // Parameter aus dem EEPROM laden und danach die geladenen Daten validieren
    if (BRKclock_ParameterValidate())
    {
        return (true);                                                          // Parameter sind gültig und wurden erfolgreich geladen
    }
    else
```

```
    {
        return (false);                                                 // Ungültige Parameter
    }
}

void BRKclock_ParameterSave ()
{
    if (BRKclock_ParameterValidate())
    {
        EEPROM.put(0, BRKclock_ParameterData); // Parameter in das EEPROM sichern
    }
}


// *** BRK-Clock Uhr-Funktionen in Deutsch ***

void BRKclock_fuenf ()
{
    for (int i = 60; i <= 63; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_zehn ()
{
    for (int i = 56; i <= 59; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_vor ()
{
    for (int i = 48; i <= 50; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_nach ()
{
    for (int i = 52; i <= 55; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_halb ()
{
```

```
    for (int i = 44; i <= 47; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_Stunde (int hr)
{
    hr = hr % 12;
    if (hr == 0)      // 12 oder 24 Uhr
    {
        hr = 12;
    }

    switch (hr)
    {
        case 1:
            for (int i = 32; i <= 35; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            break;
        case 2:
            for (int i = 27; i <= 28; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            for (int i = 21; i <= 22; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            break;
        case 3:
            for (int i = 19; i <= 22; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            break;
        case 4:
            for (int i = 40; i <= 43; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            break;
        case 5:
            for (int i = 7; i <= 8; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            for (int i = 23; i <= 24; i++)
```

```
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 6:
    for (int i = 35; i <= 39; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 7:
    for (int i = 16; i <= 18; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    for (int i = 29; i <= 31; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 8:
    for (int i = 1; i <= 4; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 9:
    for (int i = 8; i <= 11; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 10:
    for (int i = 11; i <= 14; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 11:
    for (int i = 5; i <= 7; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 12:
    for (int i = 24; i <= 28; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
```

```
        default:
            Serial.println("Fehler: Stunden");         // Fehler Stunden
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                BRKclock_Pixel.setPixelColor(i, 255, 0, 0); // red
            }
            break;
    }
}

void BRKclock_Zeit (int h, int m)
{
    Serial.print(IoT_NTPtime() + ": ");
    BRKclock_Pixel.clear();
    if ((m <= 2) || (m >= 58))  // h (h:58-h:02)
    {
        Serial.println("h (h:58-h:02)");
        if (m <= 2)
        {
            BRKclock_Stunde(h);
        }
        else
        {
            BRKclock_Stunde(h + 1);
        }
    }
    else if ((3 <= m) && (m <= 7))
    {    // FÜNF NACH h (h:03-h:07)
        Serial.println("FÜNF NACH h (h:03-h:07)");
        BRKclock_fuenf();
        BRKclock_nach();
        BRKclock_Stunde(h);
    }
    else if ((8 <= m) && (m <= 12))
    {   // ZEHN NACH h (h:08-h:12)
        Serial.println("ZEHN NACH h (h:08-h:12)");
        BRKclock_zehn();
        BRKclock_nach();
        BRKclock_Stunde(h);
    }
    else if ((13 <= m) && (m <= 17))
    {  // FÜNFZEHN NACH h (h:13-h:17)
        Serial.println("FÜNFZEHN NACH h (h:13-h:17)");
        BRKclock_fuenf();
        BRKclock_zehn();
        BRKclock_nach();
        BRKclock_Stunde(h);
    }
    else if ((18<= m) && (m <= 22))
    {  // ZEHN VOR HALB h+1 (h:18-h:22)
```

```
    Serial.println("ZEHN VOR HALB h+1 (h:18-h:22)");
    BRKclock_zehn();
    BRKclock_vor();
    BRKclock_halb();
    BRKclock_Stunde(h + 1);
}
else if ((23<= m) && (m <= 27))
{  // FÜNF VOR HALB h+1 (h:23-h:27)
    Serial.println("FÜNF VOR HALB h+1 (h:23-h:27)");
    BRKclock_fuenf();
    BRKclock_vor();
    BRKclock_halb();
    BRKclock_Stunde(h + 1);
}
else if ((28<= m) && (m <= 32))
{  //  HALB h+1 (h:28-h:32)
    Serial.println("HALB h+1 (h:28-h:32)");
    BRKclock_halb();
    BRKclock_Stunde(h + 1);
}
else if ((33<= m) && (m <= 37))
{  // FÜNF NACH HALB h+1 (h:33-h:37)
    Serial.println("FÜNF NACH HALB h+1 (h:33-h:37)");
    BRKclock_fuenf();
    BRKclock_nach();
    BRKclock_halb();
    BRKclock_Stunde(h + 1);
}
else if ((38<= m) && (m <= 42))
{  // ZEHN NACH HALB h+1 (h:38-h:42)
    Serial.println("ZEHN NACH HALB h+1 (h:38-h:42)");
    BRKclock_zehn();
    BRKclock_nach();
    BRKclock_halb();
    BRKclock_Stunde(h + 1);
}
else if ((43<= m) && (m <= 47))
{  // FÜNFZEHN VOR h+1 (h:43-h:47)
    Serial.println("FÜNFZEHN VOR h+1 (h:43-h:47)");
    BRKclock_fuenf();
    BRKclock_zehn();
    BRKclock_vor();
    BRKclock_Stunde(h + 1);
}
else if ((48<= m) && (m <= 52))
{  // ZEHN VOR h+1 (h:48-h:52)
    Serial.println("ZEHN VOR h+1 (h:48-h:52)");
    BRKclock_zehn();
    BRKclock_vor();
    BRKclock_Stunde(h + 1);
```

```
    }
    else if ((53<= m) && (m <= 57))
    {  // FÜNF VOR h+1 (h:53-h:57)
        Serial.println("FÜNF VOR h+1 (h:53-h:57)");
        BRKclock_fuenf();
        BRKclock_vor();
        BRKclock_Stunde(h + 1);
    }
    else
    {                                          // Fehler Minuten
        Serial.println("Fehler: Minuten");
        for(int j = 0; j < BRKclock_Pixel.numPixels(); j++)
        {
            BRKclock_Pixel.setPixelColor(j, 100, 100, 100);    // Weiß
        }
    }
    BRKclock_Pixel.show();
}


// *** BRK-Clock Uhr-Funktionen in Englisch ***

void BRKclock_five ()
{
    for (int i = 48; i <= 49; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    BRKclock_Pixel.setPixelColor(51, BRKclock_clockColor);
    BRKclock_Pixel.setPixelColor(53, BRKclock_clockColor);
}

void BRKclock_ten ()
{
    for (int i = 52; i <= 53; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    BRKclock_Pixel.setPixelColor(55, BRKclock_clockColor);
}

void BRKclock_fifteen ()
{
    for (int i = 48; i <= 50; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    for (int i=52; i<=55; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
```

```
    }
}

void BRKclock_twenty ()
{
    for (int i=56; i<=61; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_to ()
{
    for (int i=40; i<=41; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_past ()
{
    for (int i=41; i<=44; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_half ()
{
    for (int i = 62; i <= 63; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    for (int i = 46; i <= 47; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
}

void BRKclock_Hour (int hr)
{
    hr = hr % 12;
    if (hr == 0)      // 12 oder 24 Uhr
    {
        hr = 12;
    }

    switch (hr)
    {
        case 1:
```

```
    for (int i = 29; i <= 31; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 2:
    for (int i = 16; i <= 17; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    BRKclock_Pixel.setPixelColor(14, BRKclock_clockColor);
    break;
case 3:
    for (int i = 24; i <= 28; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 4:
    for (int i = 12; i <= 15; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 5:
    for (int i = 8; i <= 11; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 6:
    for (int i = 0; i <= 2; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 7:
    for (int i = 3; i <= 7; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 8:
    for (int i = 35; i <= 39; i++)
    {
        BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
    }
    break;
case 9:
    for (int i = 32; i <= 35; i++)
```

```
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            break;
        case 10:
            BRKclock_Pixel.setPixelColor(23, BRKclock_clockColor);
            BRKclock_Pixel.setPixelColor(24, BRKclock_clockColor);
            BRKclock_Pixel.setPixelColor(39, BRKclock_clockColor);
            break;
        case 11:
            for (int i = 18; i <= 23; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            break;
        case 12:
            for (int i = 16; i <= 19; i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_clockColor);
            }
            BRKclock_Pixel.setPixelColor(21, BRKclock_clockColor);
            BRKclock_Pixel.setPixelColor(22, BRKclock_clockColor);
            break;
        default:
            Serial.println("Error: Hours");         // Fehler Stunden
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                BRKclock_Pixel.setPixelColor(i, 255, 0, 0); // red
            }
            break;
    }
}

void BRKclock_Time (int h, int m)
{
    Serial.print(IoT_NTPtime() + ": ");
    BRKclock_Pixel.clear();
    if ((m <= 2) || (m >= 58))        // h (h:58-h:02)
    {
        Serial.println("h (h:58-h:02)");
        if (m <= 2)
        {
            BRKclock_Hour(h);
        }
        else
        {
            BRKclock_Hour(h+1);
        }
    }
    else if ((3 <= m) && (m <= 7))    // Five past h (h:03-h:07)
```

```
{
    Serial.println("Five past h (h:03-h:07)");
    BRKclock_five();
    BRKclock_past();
    BRKclock_Hour(h);
}
else if ((8 <= m) && (m <= 12))   // Ten past h (h:08-h:12)
{
    Serial.println("Ten past h (h:08-h:12)");
    BRKclock_ten();
    BRKclock_past();
    BRKclock_Hour(h);
}
else if ((13 <= m) && (m <= 17))  // Fifteen past h (h:13-h:17)
{
    Serial.println("Fifteen past h (h:13-h:17)");
    BRKclock_fifteen();
    BRKclock_past();
    BRKclock_Hour(h);
}
else if ((18<= m) && (m <= 22))  // Twenty past h+1 (h:18-h:22)
{
    Serial.println("Twenty past h+1 (h:18-h:22)");
    BRKclock_twenty();
    BRKclock_past();
    BRKclock_Hour(h);
}
else if ((23<= m) && (m <= 27))  // Twenty Five past h+1 (h:23-h:27)
{
    Serial.println("Twenty Five past h+1 (h:23-h:27)");
    BRKclock_five();
    BRKclock_twenty();
    BRKclock_past();
    BRKclock_Hour(h);
}
else if ((28<= m) && (m <= 32))  //  Half past h+1 (h:28-h:32)
{
    Serial.println("Half past h+1 (h:28-h:32)");
    BRKclock_half();
    BRKclock_past();
    BRKclock_Hour(h);
}
else if ((33<= m) && (m <= 37))  // Twenty Five to h+1 (h:33-h:37)
{
    Serial.println("Twenty Five to h+1 (h:33-h:37)");
    BRKclock_five();
    BRKclock_twenty();
    BRKclock_to();
    BRKclock_Hour(h+1);
}
```

```
    else if ((38<= m) && (m <= 42))  // Twenty to h+1 (h:38-h:42)
    {
        Serial.println("Twenty to h+1 (h:38-h:42)");
        BRKclock_twenty();
        BRKclock_to();
        BRKclock_Hour(h+1);
    }
    else if ((43<= m) && (m <= 47))  // Fifteen to h+1 (h:43-h:47)
    {
        Serial.println("Fifteen to h+1 (h:43-h:47)");
        BRKclock_fifteen();
        BRKclock_to();
        BRKclock_Hour(h+1);
    }
    else if ((48<= m) && (m <= 52))  // Ten to h+1 (h:48-h:52)
    {
        Serial.println("Ten to h+1 (h:48-h:52)");
        BRKclock_ten();
        BRKclock_to();
        BRKclock_Hour(h+1);
    }
    else if ((53<= m) && (m <= 57))  // Five to h+1 (h:53-h:57)
    {
        Serial.println("Five to h+1 (h:53-h:57)");
        BRKclock_five();
        BRKclock_to();
        BRKclock_Hour(h+1);
    }
    else
    {                                           // Fehler Minuten
        Serial.println("Error: Minutes");
        for (int j = 0; j < BRKclock_Pixel.numPixels(); j++)
        {
            BRKclock_Pixel.setPixelColor(j, 100, 100, 100);    // Weiß
        }
    }
    BRKclock_Pixel.show();
}


// *** BRK-Clock Uhr-Funktionen International ***

void BRKclock_International (int h, int m)
{
    switch (BRKclock_language)
    {
        case 0:
            BRKclock_Zeit (h, m);
            break;
        case 1:
```

```
            BRKclock_Time (h, m);
            break;
        default:
            break;
    }
}


// *** BRK-Clock Demos ***

void BRKclock_Kaleidoscope (int size, bool color16)
{
    BRKclock_Clear(false);                                          // Pixel löschen & Anzeige aktualisieren
    long t = millis();
    size--;
    for (int w = 0; w <= size * 10; w++)
    {
        for (int i = 0; i <= (size / 2); i++)
        {
            for (int j = 0; j <= (size / 2); j++)
            {
                int k = i + j;
                int color = j * 3 / (i + 3) + (i + 1) * w * 3;
                if (color16)
                {
                    BRKclock_Plot16color(i, k, color);
                    BRKclock_Plot16color(k, i, color);
                    BRKclock_Plot16color(size - i, size - k, color);
                    BRKclock_Plot16color(size - k, size - i, color);
                    BRKclock_Plot16color(k, size - i, color);
                    BRKclock_Plot16color(size - i, k, color);
                    BRKclock_Plot16color(i, size - k, color);
                    BRKclock_Plot16color(size - k, i, color);
                }
                else
                {
                    BRKclock_Plot8color(i, k, color);
                    BRKclock_Plot8color(k, i, color);
                    BRKclock_Plot8color(size - i, size - k, color);
                    BRKclock_Plot8color(size - k, size - i, color);
                    BRKclock_Plot8color(k, size - i, color);
                    BRKclock_Plot8color(size - i, k, color);
                    BRKclock_Plot8color(i, size - k, color);
                    BRKclock_Plot8color(size - k, i, color);
                }
                BRKclock_Matrix.show();
                IoT_Idle();
                IoT_WaitKeypress(50, false);                                // 50 ms. oder auf Taster warten
                if (IoT_Keypress())
                {
```

```
                    return;
                }
            }
        }
    }
}

void BRKclock_Rectangles ()
{
    BRKclock_Clear(false);                                          // Pixel löschen & Anzeige aktualisieren
    int i = 0;
    IoT_WaitMessage();                                              // Mitteilung anzeigen
    while (not(IoT_Keypress()))
    {
        i++;
        BRKclock_Matrix.drawRect(0, 0, 8, 8, BRKclock_Matrix16Color[i & 15]);
        BRKclock_Matrix.show();
        IoT_WaitKeypress(500, false);                                 // 500 ms. oder auf Taster warten
        i++;
        BRKclock_Matrix.drawRect(1, 1, 6, 6, BRKclock_Matrix16Color[i & 15]);
        BRKclock_Matrix.show();
        IoT_WaitKeypress(500, false);                                 // 500 ms. oder auf Taster warten
        i++;
        BRKclock_Matrix.drawRect(2, 2, 4, 4, BRKclock_Matrix16Color[i & 15]);
        BRKclock_Matrix.show();
        IoT_WaitKeypress(500, false);                                 // 500 ms. oder auf Taster warten
        i++;
        BRKclock_Matrix.drawRect(3, 3, 2, 2, BRKclock_Matrix16Color[i & 15]);
        BRKclock_Matrix.show();
        IoT_WaitKeypress(500, false);                                 // 500 ms. oder auf Taster warten
    }
}

void BRKclock_clockColorWipe (uint32_t c, byte wait)                   // Schlangenlinien
{
    for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
    {
        BRKclock_Pixel.setPixelColor(i, c);
        BRKclock_Pixel.show();
        IoT_WaitKeypress(wait, false);                                // wait ms. oder auf Taster warten
        if (IoT_Keypress())
        {
            return;
        }
    }
}

void BRKclock_WhiteOverRainbow (byte wait, byte whiteSpeed, byte whiteLength) // Weiße Welle über Regenbogen
{
    if (whiteLength >= BRKclock_Pixel.numPixels())
```

```
    {
        whiteLength = BRKclock_Pixel.numPixels() - 1;
    }
    int head = whiteLength - 1;
    int tail = 0;
    int loops = 3;
    int loopNum = 0;
    static unsigned long lastTime = 0;
    while (true)
    {
        for (int j = 0; j < 256; j++)
        {
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                if ((i >= tail && i <= head) || (tail > head && i >= tail) || (tail > head && i <= head) )
                {
                    BRKclock_Pixel.setPixelColor(i, BRKclock_Pixel.Color(0,0,0, 255));
                }
                else
                {
                    BRKclock_Pixel.setPixelColor(i, BRKclock_clockColorWheel(((i * 256 / BRKclock_Pixel.numPixels()) + j) & 255));
                }
            }
            if (millis() - lastTime > whiteSpeed)
            {
                head++;
                tail++;
                if(head == BRKclock_Pixel.numPixels())
                {
                    loopNum++;
                }
                lastTime = millis();
            }
            if (loopNum == loops)
            {
                return;
            }
            head%=BRKclock_Pixel.numPixels();
            tail%=BRKclock_Pixel.numPixels();
            BRKclock_Pixel.show();
            IoT_WaitKeypress(wait, false);                              // wait ms. oder auf Taster warten
            if (IoT_Keypress())
            {
                return;
            }
        }
    }
}

void BRKclock_PulseWhite (byte wait, byte count)                  // Weißes Pulsieren
```

```
{
    for (byte loop = 1; loop <= count; loop++)
    {
        for (int j = 0; j < 256 ; j++)
        {
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_Pixel.Color(0,0,0, BRKclock_Gamma[j] ) );
            }
            IoT_WaitKeypress(wait, false);                              // wait ms. oder auf Taster warten
            if (IoT_Keypress())
            {
                return;
            }
            BRKclock_Pixel.show();
        }
        for (int j = 255; j >= 0 ; j--)
        {
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_Pixel.Color(0,0,0, BRKclock_Gamma[j] ) );
            }
            IoT_WaitKeypress(wait, false);                              // wait ms. oder auf Taster warten
            if (IoT_Keypress())
            {
                return;
            }
            BRKclock_Pixel.show();
        }
    }
}

void BRKclock_RainbowFade2White (byte wait, int rainbowLoops, int whiteLoops) // Regenbogen wird weiß
{
    float fadeMax = 100.0;
    int fadeVal = 0;
    uint32_t wheelVal;
    int redVal, greenVal, blueVal;

    for (int k = 0 ; k < rainbowLoops ; k ++)
    {
        for (int j=0; j<256; j++) // 5 Zyklen für alle Farben des Farbrades
        {
            for (int i=0; i< BRKclock_Pixel.numPixels(); i++)
            {
                wheelVal = BRKclock_clockColorWheel(((i * 256 / BRKclock_Pixel.numPixels()) + j) & 255);
                redVal = ((wheelVal >> 16) & 0xFF) * float(fadeVal/fadeMax);
                greenVal = ((wheelVal >> 8) & 0xFF) * float(fadeVal/fadeMax);
                blueVal = (wheelVal & 0xFF) * float(fadeVal/fadeMax);
                BRKclock_Pixel.setPixelColor(i, BRKclock_Pixel.Color(redVal, greenVal, blueVal));
```

```
            }
            if (k == 0 && fadeVal < fadeMax - 1)      // Erster Durchlauf, Fade In
            {
                fadeVal++;
            }
            else if (k == rainbowLoops - 1 && j > 255 - fadeMax)        // Letzter Durchlauf, Fade Out
            {
                fadeVal--;
            }
            BRKclock_Pixel.show();
            IoT_WaitKeypress(wait, false);                              // wait ms. oder auf Taster warten
            if (IoT_Keypress())
            {
                return;
            }
        }
    }
    IoT_WaitKeypress(500, false);                                   // 500 ms. oder auf Taster warten
    if (IoT_Keypress())
    {
        return;
    }

    for (int k = 0 ; k < whiteLoops ; k ++)
    {
        for (int j = 0; j < 256 ; j++)
        {
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_Pixel.Color(0,0,0, BRKclock_Gamma[j]));
            }
            BRKclock_Pixel.show();
        }
        IoT_WaitKeypress(1000, false);                                // 1 Sekunde oder auf Taster warten
        if (IoT_Keypress())
        {
            return;
        }
        for (int j = 255; j >= 0 ; j--)
        {
            for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
            {
                BRKclock_Pixel.setPixelColor(i, BRKclock_Pixel.Color(0,0,0, BRKclock_Gamma[j]));
            }
            BRKclock_Pixel.show();
        }
    }
    IoT_WaitKeypress(500, false);                                   // 500 ms. oder auf Taster warten
}
```

```
void BRKclock_AmbienceColors (int wait)                             // Ambientebeleuchtung
{
    BRKclock_FullColor(BRKclock_Pixel.Color(0,255,0,0));              // Rot
    IoT_WaitKeypress(wait, false);                                    // wait ms. oder auf Taster warten
    if (IoT_Keypress())
    {
        return;
    }
    BRKclock_FullColor(BRKclock_Pixel.Color(255,0,0,0));              // Grün
    IoT_WaitKeypress(wait, false);                                    // wait ms. oder auf Taster warten
    if (IoT_Keypress())
    {
        return;
    }
    BRKclock_FullColor(BRKclock_Pixel.Color(0,0,255,0));              // Blau
    IoT_WaitKeypress(wait, false);                                    // wait ms. oder auf Taster warten
    if (IoT_Keypress())
    {
        return;
    }
    BRKclock_FullColor(BRKclock_Pixel.Color(0,0,0,255));              // Weiß
    IoT_WaitKeypress(wait, false);                                    // wait ms. oder auf Taster warten
    if (IoT_Keypress())
    {
        return;
    }
}

void BRKclock_RainbowCycle (byte wait)                              // Regenbogenzyklus
{
    for (int j = 0; j < 256 * 5; j++) // 5 cycles of all colors on wheel
    {
        for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
        {
            BRKclock_Pixel.setPixelColor(i, BRKclock_clockColorWheel(((i * 256 / BRKclock_Pixel.numPixels()) + j) & 255));
        }
        BRKclock_Pixel.show();
        IoT_WaitKeypress(wait, false);                                  // wait ms. oder auf Taster warten
        if (IoT_Keypress())
        {
            return;
        }
    }
}

void BRKclock_Rainbow (byte wait)
{
    for (int j = 0; j < 256; j++)
    {
        for (int i = 0; i < BRKclock_Pixel.numPixels(); i++)
```

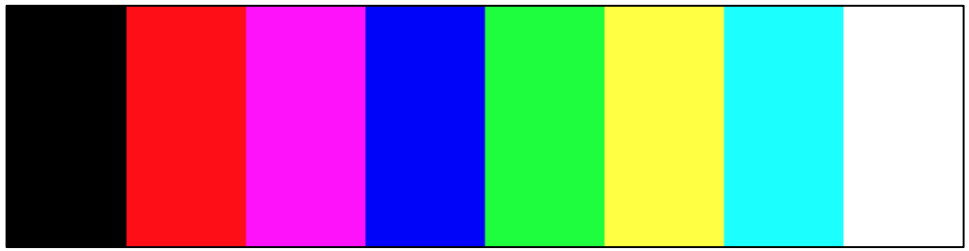
```
        {
            BRKclock_Pixel.setPixelColor(i, BRKclock_clockColorWheel((i + j) & 255));
        }
        BRKclock_Pixel.show();
        IoT_WaitKeypress(wait, false);                              // wait ms. oder auf Taster warten
        if (IoT_Keypress())
        {
            return;
        }
    }
}

void BRKclock_SnakeLines (byte wait)
{
    for (int i = 1; i <= 7; i++)
    {

        BRKclock_clockColorWipe(BRKclock_Pixel.Color(t_RGBColor8ValueGreen[i],
                                                     t_RGBColor8ValueRed[i],
                                                     t_RGBColor8ValueBlue[i]), wait);
    }
}
```

BRK-Clock Library Dokumentation

Die BRK-Clock Library stellt Befehle zur Verfügung, um eine Wort-Uhr aus 8 x 8 programmierbaren RGBW-Bricks zu programmieren. Aktuell wird die deutsche Version der BRK-Clock unterstützt. Die dazu benötigten, landesspezifischen Befehle sind in deutscher Nomenklatur gehalten. Eine Erweiterung mit anderen Sprachen ist dadurch problemlos möglich. Hardwarespezifische Basis dieser Library ist die NeoPixel Library, die NeoMatrix Library und die GFX Library von Adafruit, die ebenfalls geladen sein müssen. Auf Grund der Größe und Komplexität dieser Libraries macht es wenig Sinn, diese komplett als Quelltext in die BRK-Clock Library zu integrieren. Im Programm muß vor dem `#include` der Library die Datenleitung, z.B. GPIO13, mit dem Befehl `#define BRKclock_PIN 13` definiert werden. Wird `BRKclock_PIN` nicht definiert, so wird automatisch `BRKclock_PIN 14` verwendet.

Globale Konstanten für Farbdefinition der speziellen 16 Bit Matrix-Farben

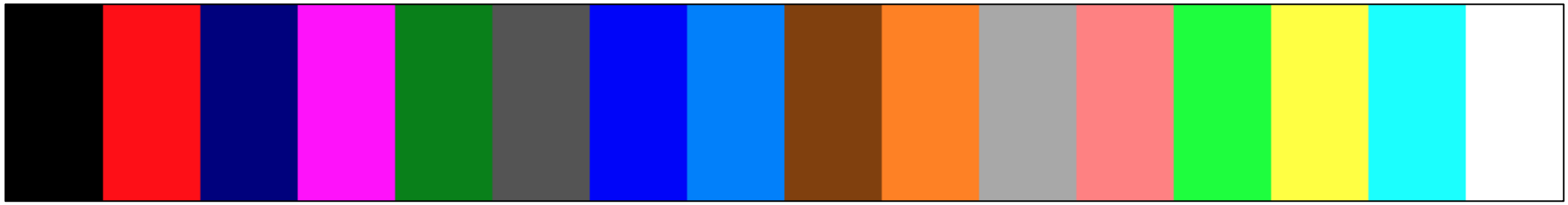
Die folgenden Farbdefinitionen, einmal für die 8 Grundfarben und einmal für erweiterte 16 Farben, beziehen sich auf die Farbdefinition des `BRKclock_Matrix` Objektes und aller Befehle, die damit in Zusammenhang stehen. Diese werden gepackt in nur 16 Bit (an Stelle der normalerweise benötigten 24 Bit) gespeichert. Das betrifft auch die `BRKclock_Plot8color` und `BRKclock_Plot16color` Befehle. Die 8 indizierten Farben sehen wie folgt aus:



```
const uint16_t BRKclock_Matrix8Color  [ ] = {BRKclock_Matrix.Color(0x00, 0x00, 0x00),  //  0 = Schwarz
                                             BRKclock_Matrix.Color(0xFF, 0x00, 0x00),  //  1 = Rot
                                             BRKclock_Matrix.Color(0xFF, 0x00, 0xFF),  //  2 = Purpur (Magenta)
                                             BRKclock_Matrix.Color(0x00, 0x00, 0xFF),  //  3 = Blau
                                             BRKclock_Matrix.Color(0x00, 0xFF, 0x00),  //  4 = Grün
                                             BRKclock_Matrix.Color(0xFF, 0xFF, 0x00),  //  5 = Gelb
                                             BRKclock_Matrix.Color(0x00, 0xFF, 0xFF),  //  6 = Türkis (Cyan)
                                             BRKclock_Matrix.Color(0xFF, 0xFF, 0xFF)}; //  7 = Weiß
```

Die indizierten Farben haben dieselben Farbdefinitionen wie in der Terminal-Library. Die in den 8 indizierten Farben enthaltenen Anteile von Rot, Grün und Blau (RGB) lassen sich über die Konstanten `t_RGBColor8ValueRed[index8]`, `t_RGBColor8ValueGreen[index8]` und `t_RGBColor8ValueBlue[index8]` bestimmen. Die in den 16 indizierten Farben enthaltenen Anteile von Rot, Grün und Blau lassen sich über die Konstanten `t_RGBColor16ValueRed[index16]`, `t_RGBColor16ValueGreen[index16]` und `t_RGBColor16ValueBlue[index16]` bestimmen. Der `index16` darf dabei einen Wert zwischen 0 und 15 annehmen.

Die 16 indizierten Farben sehen wie folgt aus:



```
const uint16_t BRKclock_Matrix16Color [ ] = {BRKclock_Matrix.Color(0x00, 0x00, 0x00),  //  0 = Schwarz
                                             BRKclock_Matrix.Color(0xFF, 0x00, 0x00),  //  1 = Rot
                                             BRKclock_Matrix.Color(0x00, 0x00, 0x80),  //  2 = Dunkelblau
                                             BRKclock_Matrix.Color(0xFF, 0x00, 0xFF),  //  3 = Purpur (Magenta)
                                             BRKclock_Matrix.Color(0x00, 0x80, 0x00),  //  4 = Dunkelgrün
                                             BRKclock_Matrix.Color(0x54, 0x54, 0x54),  //  5 = Dunkelgrau
                                             BRKclock_Matrix.Color(0x00, 0x00, 0xFF),  //  6 = Blau
                                             BRKclock_Matrix.Color(0x00, 0x80, 0xFF),  //  7 = Hellblau
                                             BRKclock_Matrix.Color(0x80, 0x40, 0x00),  //  8 = Braun
                                             BRKclock_Matrix.Color(0xFF, 0x80, 0x00),  //  9 = Orange
                                             BRKclock_Matrix.Color(0xA8, 0xA8, 0xA8),  // 10 = Hellgrau
                                             BRKclock_Matrix.Color(0xFF, 0x80, 0x80),  // 11 = Pink
                                             BRKclock_Matrix.Color(0x00, 0xFF, 0x00),  // 12 = Grün
                                             BRKclock_Matrix.Color(0xFF, 0xFF, 0x00),  // 13 = Gelb
                                             BRKclock_Matrix.Color(0x00, 0xFF, 0xFF),  // 14 = Türkis (Cyan)
                                             BRKclock_Matrix.Color(0xFF, 0xFF, 0xFF)}; // 15 = Weiß
```

Globale Konstanten für den Gamma-Wert (Helligkeitskorrektur)

Die Gamma-Tabelle wird für eine Harmonisierung der Helligkeit der LEDs benötigt.

```
const byte BRKclock_Gamma             [ ] = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                                               0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
                                               1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,
                                               2,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,
                                               5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  9,  9,  9, 10,
                                              10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
                                              17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
                                              25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
                                              37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
                                              51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
                                              69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
                                              90, 92, 93, 95, 96, 98, 99,101,102,104,105,107,109,110,112,114,
                                             115,117,119,120,122,124,126,127,129,131,133,135,137,138,140,142,
                                             144,146,148,150,152,154,156,158,160,162,164,167,169,171,173,175,
                                             177,180,182,184,186,189,191,193,196,198,200,203,205,208,210,213,
                                             215,218,220,223,225,228,231,233,236,239,241,244,247,249,252,255  };
```

Globale Variablen

```
BRKclock_Parameter BRKclock_ParameterData;

Adafruit_NeoPixel  BRKclock_Pixel  = Adafruit_NeoPixel(LED, PIN, NEO_RGBW + NEO_KHZ800);

Adafruit_NeoMatrix BRKclock_Matrix = Adafruit_NeoMatrix(8, 8, PIN,
                                                        NEO_MATRIX_BOTTOM + NEO_MATRIX_LEFT +
                                                        NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,
                                                        NEO_GRBW + NEO_KHZ800);
```

Farbe der Temperatur-Anzeige

```
uint32_t BRKclock_ClockColor = BRKclock_Pixel.Color(150, 0, 150, 0);   // (grün, rot, blau, weiß)
byte     BRKclock_ClockColorRed       = 0xFF;                          // Farbanteil für Rot-Kanal
byte     BRKclock_ClockColorGreen     = 0x00;                          // Farbanteil für Grün-Kanal
byte     BRKclock_ClockColorBlue      = 0x00;                          // Farbanteil für Blau-Kanal
byte     BRKclock_ClockColorWhite     = 0x00;                          // Farbanteil für Weiß-Kanal
```

Farbe der Uhrzeit-Anzeige

```
uint16_t BRKclock_tempColor  = BRKclock_Matrix.Color(0xFF, 0x00, 0x00);// (rot, grün, blau), 16 Bit komprimiert
byte     BRKclock_tempColorRed        = 0xFF;                          // Farbanteil für Rot-Kanal
byte     BRKclock_tempColorGreen      = 0x00;                          // Farbanteil für Grün-Kanal
byte     BRKclock_tempColorBlue       = 0x00;                          // Farbanteil für Blau-Kanal
```

Benutzerdefinierte Einstellungen

```
byte     BRKclock_staticIP            = 0;                             // 0 = Dynamische IP-Adresse, 1 = Statische
int      BRKclock_alarmMode           = 0;                             // 0 = Wecker aus, 1 = Wecker ein
int      BRKclock_temperatureMode     = 1;                             // 0 = Temperaturanzeige aus, 1 = jede Minute
int      BRKclock_colorMode           = 2;                             // 0 = Random Schrift, 1 = Weiße Schrift, usw.
int      BRKclock_buttonMode          = 0;                             // 0 = IP-Adresse anzeigen, 1 = Temperatur usw.
byte     BRKclock_alarmHour           = 0;                             // Weckzeit: Stunden
byte     BRKclock_alarmMinute         = 0;                             // Weckzeit: Minuten
```

BRK-Clock Funktionen

```
void BRKclock_Init ()
```

Initialisiert die BRK-Clock und schaltet alle RGBW-Bricks aus.

```
void BRKclock_WLANautoConnect (bool useSavedSettings)
```

Stellt eine Verbindung zum zuletzt benutzen WLAN-Hotspot her, wenn für `useSavedSettings` als Wert `true` übergeben wurde. Falls die Verbindung nicht hergestellt werden konnte oder für `useSavedSettings` als Wert `false` übergeben wurde, dann wird der Benutzer durch eine Meldung im OLED-Display (falls ein OLED-Display angeschlossen ist) dazu aufgefordert, den „IoT-Brick“ Hotspot in den Einstellungen z.B. eines Smartphones auszuwählen. Sobald diese Auswahl erfolgt ist, öffnet sich im Smartphone der Browser und danach kann der gewünschte Hotspot ausgewählt und das Password eingegeben werden. Danach wird das OLED-Display (falls ein OLED-Display angeschlossen ist) gelöscht und das Programm fortgesetzt. Während des Verbindungsaufbaus leuchten in der Matrix die drei Buchstaben "**BRK**", in der unteren linken Ecke, in den Farben blau, rot und grün auf. Bei der englischen Wort-Uhr sind es die Buchstaben "**TFS**" an derselben Position.

```
void BRKclock_Clear (bool RGBW)
```

Setzt alle RGBW-LEDs der BRK-Clock auf schwarz. Dabei werden die Pixel-Speicher sowohl für `BRKclock_Pixel` als auch für `BRKclock_Matrix` gelöscht und auf schwarz gesetzt. Nach Aufruf dieses Befehls wird wahlweise der RGBW-Modus für alle `BRKclock_Matrix`-Befehle eingeschaltet, wenn für `RGBW` ein `true` übergeben wird, sonst wird der RGB-Modus für alle `BRKclock_Matrix`-Befehle eingeschaltet.

```
void BRKclock_FullColor (uint32_t c)
```

Setzt alle RGBW-LEDs der BRK-Clock auf die angegebene Farbe.

```
void BRKclock_Plot8color (int x, int y, int color)
```

Setzt den Brick an der angegebenen Koordinate (x=0, y=0 ist der Brick oben links) auf einen der 8 vordefinierten Farben. Die Farben sind von 0 bis 7 durchnumeriert. Wird ein Farbwert größer 7 angegeben, so wird der Farbwert modulo 8 genommen, d.h. die 8 entspricht der 0, die 9 der 1 usw.

```
void BRKclock_Plot16color (int x, int y, int color)
```

Setzt den Brick an der angegebenen Koordinate (x=0, y=0 ist der Brick oben links) auf einen der 16 vordefinierten Farben. Die Farben sind von 0 bis 15 durchnumeriert. Wird ein Farbwert größer 15 angegeben, so wird der Farbwert modulo 16 genommen, d.h. die 16 entspricht der 0, die 17 der 1 usw.

```
void BRKclock_SetClockRandomColor ()
```

Setzt die Variable `BRKclock_ClockColor` auf einen zufälligen Farbwert aus den 16 indizierten Farben 1 bis 15, also ohne schwarz. Die Variablen für die Farbkanäle `BRKclock_ClockColorRed`, `BRKclock_ClockColorGreen`, `BRKclock_ClockColorBlue` und `BRKclock_ClockColorWhite` werden auf die zugehörigen Werte gesetzt.

```
void BRKclock_SetClockRGBWColor (byte red, byte green, byte blue, byte white)
```

Setzt die Variable `BRKclock_ClockColor` auf den angegebenen Farbwert sowie die Variablen für die Farbkanäle `BRKclock_ClockColorRed`, `BRKclock_ClockColorGreen`, `BRKclock_ClockColorBlue` und `BRKclock_ClockColorWhite` auf die angegebenen Werte.

```
void BRKclock_SetTemperatureRGBColor (byte red, byte green, byte blue)
```

Setzt die Variable `BRKclock_TemperatureColor` auf den angegebenen RGB-Farbwert sowie die Variablen für die Farbkanäle `BRKclock_TemperatureColorRed`, `BRKclock_TemperatureColorGreen` und `BRKclock_TemperatureColorBlue` auf die angegebenen Werte. Nach Aufruf dieses Befehls wird der RGB-Modus für alle `BRKclock_Matrix`-Befehle eingeschaltet.

```
void BRKclock_SetTemperatureRGBWColor (byte red, byte green, byte blue, byte white)
```

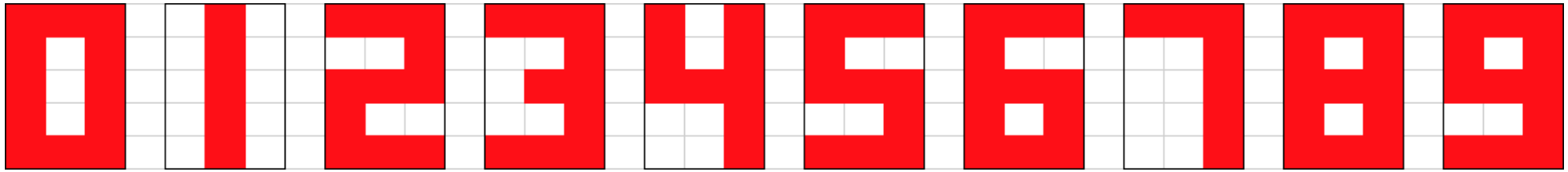
Setzt die Variable `BRKclock_TemperatureColor` auf den angegebenen RGBW-Farbwert sowie die Variablen für die Farbkanäle `BRKclock_TemperatureColorRed`, `BRKclock_TemperatureColorGreen`, `BRKclock_TemperatureColorBlue` und `BRKclock_TemperatureColorWhite` auf die angegebenen Werte. Nach Aufruf dieses Befehls wird der RGBW-Modus für alle `BRKclock_Matrix`-Befehle eingeschaltet.

```
void BRKclock_Plot (int x, int y)
```

Setzt den Brick an der angegebenen Koordinate (x=0, y=0 ist der Brick oben links) auf den durch die Variable `BRKclock_tempColor` definierten Farbwert. Die Farbe kann man durch `SetTemperatureRGBColor()` auf einen beliebigen RGB-Wert und durch `SetTemperatureRGBWColor()` auf einen beliebigen RGBW-Wert setzen.

```
void BRKclock_DrawNumber (int x, int y, byte num)
```

Zeichnet eine Zahl von 0 bis 9 an der angegebenen Koordinate. Die Zahl benötigt 3 Pixel in der Breite und 5 Pixel in der Höhe. Jede Zahl hat die gleiche Höhe und die gleiche Breite. Es wird der in der Variable `BRKclock_tempColor` definierten Farbwert verwendet. Die Farbe kann man durch `SetTemperatureRGBColor()` auf einen beliebigen RGB-Wert setzen.
Die Zahlen sehen dabei wie folgt aus:



```
void BRKclock_Print (String s, int pause)
```

Zeigt eine Laufschrift auf der 8 x 8 Matrix an, die von rechts nach links läuft, mit der angegebenen Pause in Millisekunden zwischen jedem horizontalen Scroll-Schritt. Als Farbe für die Laufschrift wird die Temperatur-Farbe benutzt.

```
void BRKclock_PrintTemperature (int temperature, int pause)
```

Zeigt eine zweistellige Temperatur an. Danach wir die angegebene Pause in Millisekunden abgewartet, bevor die Funktion beendet wird. Als Farbe für die Temperatur wird die Temperatur-Farbe benutzt.

```
void BRKclock_TestMatrix (int waitTime)
```

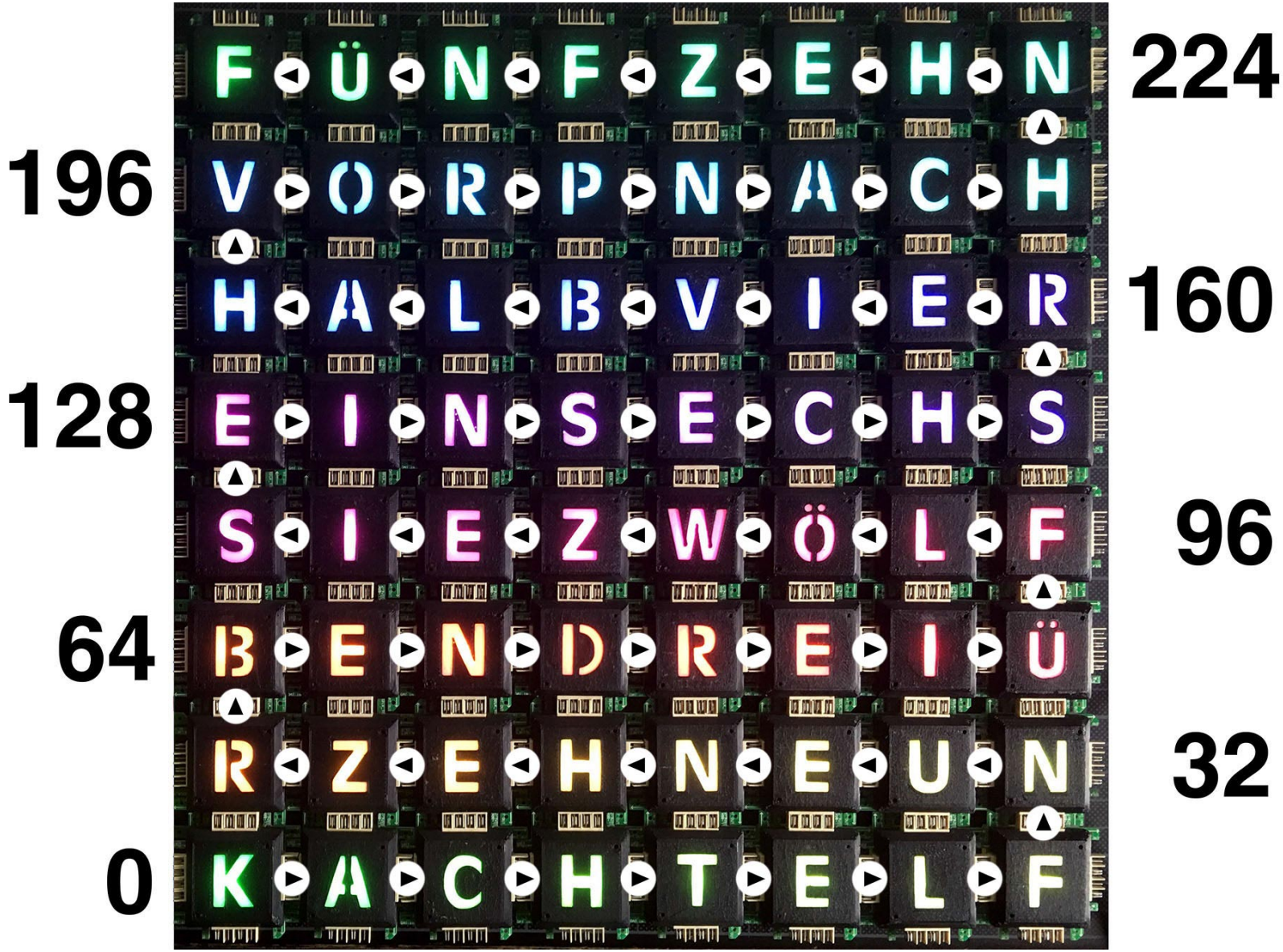
Schaltet alle RGBW-Bricks ein, beginnend mit grün (Farbwinkel 0) unten links bekommt jeder RGBW-Brick einen um 4 höheren Farbwinkel-Wert als sein Vorgänger. Die Reihenfolge entspricht der hardwaremäßig verdrahteten Reihenfolge der RGBW-Bricks. Bevor sich der nächste Brick einschaltet, wird eine Pause eingelegt. Die Länge der Pause in Millisekunden wird in `waitTime` übergeben.

```
uint32_t BRKclock_ClockColorWheel (byte angle)
```

Wählt eine Farbe an Hand des mit `angle` angegebenen Winkels (0-255) im Farbkreis aus und übergibt den zugehörigen RGB-Farbwert als Funktionsergebnis. Die Variable `BRKclock_ClockColor` kann z.B. mit diesem Funktionsergebnis auf die gewünschte Anzeige-Farbe gesetzt werden. Dabei sind die mit `angle` angegebenen Winkel den folgenden Farben zugeordnet:

0	Grün	32	Hellgrün	48	Gelb	64	Orange
88	Rot	96	Pink	128	Purpur (Magenta)	160	Violett
172	Blau	196	Hellblau	224	Türkis (Cyan)	256	Grün (identisch mit 0)

Winkel-Farbwerte der BRK-Clock während des Selbsttests. Die Pfeile zeigen die Reihenfolge der RGBW-Brick-Verknüpfung an:



BRK-Clock Parameter für das EEPROM des ESP8266

`bool BRKclock_ParameterValidate ()`

Diese Funktion überprüft, ob der BRK-Clock Parameter-Block eine gültige Signatur enthält, in dem Fall wird ein `true` übergeben. Wenn die Signatur ungültig ist, wird ein `false` übergeben.

`bool BRKclock_ParameterApply ()`

Übernimmt den BRK-Clock Parameter-Block in die aktuelle Konfiguration. Dadurch können sich alle Parameter ändern. Die IP-Adresse aus dem Parameter-Block wird nur dann übernommen, wenn es sich um eine statische IP-Adresse handelt. Für eine statische IP-Adresse muss der Parameter-Block-Eintrag `staticIP` auf 1 gesetzt sein. Wenn der Parameter-Block-Eintrag `staticIP` auf 0 gesetzt ist, dann handelt es sich um eine dynamische IP-Adresse, die nicht für die aktuelle Konfiguration übernommen wird. Es wird überprüft, ob der Parameter-Block eine gültige Signatur enthält, in dem Fall wird ein `true` übergeben. Wenn die Signatur ungültig ist, werden keine Daten aus dem Parameter-Block übernommen und ein `false` übergeben.

`void BRKclock_ParameterCreate ()`

Erzeugt den BRK-Clock Parameter-Block aus der aktuelle Konfiguration. Der Parameter-Block wird mit einer Signatur versehen. Durch die Signatur wird sichergestellt, dass die Daten in dem Parameter-Block durch den Befehl `BRKclock_ParameterCreate` oder den Befehl `BRKclock_ParameterNew` erzeugt wurden und die Verwendung der Daten problemlos möglich ist.

`void BRKclock_ParameterNew ()`

Erzeugt den BRK-Clock Parameter-Block aus der werksmäßigen Standard-Konfiguration. Der Parameter-Block wird mit einer Signatur versehen. Durch die Signatur wird sichergestellt, dass die Daten in dem Parameter-Block durch den Befehl `BRKclock_ParameterCreate` oder den Befehl `BRKclock_ParameterNew` erzeugt wurden und die Verwendung der Daten problemlos möglich ist.

`bool BRKclock_ParameterLoad ()`

Liest den BRK-Clock Parameter-Block aus dem EEPROM des ESP8266. Es wird überprüft, ob der Parameter-Block eine gültige Signatur enthält, in dem Fall wird ein `true` übergeben. Wenn die Signatur ungültig ist, werden keine Daten aus dem Parameter-Block übernommen und ein `false` übergeben.

```
void BRKclock_ParameterSave ()
```

Schreibt den BRK-Clock Parameter-Block in das EEPROM des ESP8266, wenn dieser eine gültige Signatur enthält. Wenn die Signatur ungültig ist, werden keine Daten in das EEPROM des ESP8266 geschrieben.

BRK-Clock Uhrzeit-Funktionen in deutsch

```
void BRKclock_fuenf ()
```

Blendet das Wort „FÜNF“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_zehn ()
```

Blendet das Wort „ZEHN“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_vor ()
```

Blendet das Wort „VOR“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_nach ()
```

Blendet das Wort „NACH“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_halb ()
```

Blendet das Wort „HALB“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_Stunde (int hr)
```

Blendet das deutsche Wort für die Stunde (EINS bis ZWÖLF) der Uhr-Matrix ein. Die numerisch gewünschte Stunde (1-12) wird in der Variable `hr` übergeben. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_Zeit (int h, int m)
```

Setzt die angegebene Uhrzeit auf der BRK-Clock in die entsprechenden deutschen Wort-Kombinationen um.

BRK-Clock Uhrzeit-Funktionen in englisch

```
void BRKclock_five ()
```

Blendet das Wort „`FIVE`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_ten ()
```

Blendet das Wort „`TEN`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_fifteen ()
```

Blendet das Wort „`FIFTEEN`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_twenty ()
```

Blendet das Wort „`TWENTY`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_to ()
```

Blendet das Wort „`TO`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_past ()
```

Blendet das Wort „`PAST`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_half ()
```

Blendet das Wort „`HALF`“ in der Uhr-Matrix ein. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_Hour (int hr)
```

Blendet das englische Wort für die Stunde (`ONE` bis `TWELVE`) der Uhr-Matrix ein. Die numerisch gewünschte Stunde (1-12) wird in der Variable `hr` übergeben. Die Farbe für die Anzeige wird durch die Variable `BRKclock_ClockColor` definiert.

```
void BRKclock_Time (int h, int m)
```

Setzt die angegebene Uhrzeit auf der BRK-Clock in die entsprechenden englischen Wort-Kombinationen um.

BRK-Clock Uhrzeit-Funktionen international

```
void BRKclock_International (int h, int m)
```

Setzt die angegebene Uhrzeit auf der BRK-Clock in die entsprechenden Wort-Kombinationen um und zwar in Abhängigkeit von der mit `BRKclock_language` gewählten Sprache entweder in deutsch oder in englisch.

BRK-Clock Demos

```
void BRKclock_Kaleidoscope (int size, bool color16)
```

Zeigt ein Kaleidoskop in der angegebenen Pixel-Größe (8 für die BRK Clock) in wahlweise 8 Farben (`color16` = `false`) oder 16 Farben (`color16` = `true`). Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_Rectangles ()
```

Zeigt eine Rechtecks-Demo in 16 Farben. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_ClockColorWipe (uint32_t c, byte wait)
```

Zeigt eine Schlangenlinien-Demo in Millionen Farben. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_WhiteOverRainbow (byte wait, byte whiteSpeed, byte whiteLength)
```

Zeigt eine weiße Welle über einer Regenbogen-Demo in Millionen Farben. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_PulseWhite (byte wait, byte count)
```

Zeigt eine pulsierende weiße Fläche über die gesamte Matrix. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_RainbowFade2White (byte wait, int rainbowLoops, int whiteLoops)
```

Zeigt eine Regenbogen-Demo in Millionen Farben, die zum Ende hin weiß wird. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_AmbienceColors (int wait)
```

Zeigt eine Ambientebeleuchtung über die gesamte Matrix in den vier Grundfarben rot, grün, blau und weiß. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_RainbowCycle (byte wait)                    // Regenbogenzyklus
```

Zeigt eine zyklische Regenbogen-Demo in Millionen Farben. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_Rainbow (byte wait)
```

Zeigt eine Regenbogen-Demo in Millionen Farben. Die Demo kann mit dem Taster unterbrochen werden.

```
void BRKclock_SnakeLines (byte wait)
```

Zeigt eine Schlangenlinien-Demo in den 8 indizierten Farben ohne schwarz. Die Demo kann mit dem Taster unterbrochen werden.

Sound Library Listing

Die Sound-Library stellt Funktionen zur einkanaligen Soundausgabe und Musikerzeugung zur Verfügung. Eine Erweiterung auf mehrere Kanäle ist problemlos möglich. Auf Grund der wenigen GPIOs des IoT-Bricks jedoch aktuell nicht notwendig.

```
// Sound Library
// 1.00 - 2017-06-08
// (c) Copyright 2017
//
// Zur Verwendung mit allen Allnet Libraries

byte snd_GPIO = 3;

// *** Konstanten für Musik-Töne in 8 Oktaven ***

#define snd_Note_B0  31
#define snd_Note_C1  33
#define snd_Note_CS1 35
#define snd_Note_D1  37
#define snd_Note_DS1 39
#define snd_Note_E1  41
#define snd_Note_F1  44
#define snd_Note_FS1 46
#define snd_Note_G1  49
#define snd_Note_GS1 52
#define snd_Note_A1  55
#define snd_Note_AS1 58
#define snd_Note_B1  62
#define snd_Note_C2  65
#define snd_Note_CS2 69
#define snd_Note_D2  73
#define snd_Note_DS2 78
#define snd_Note_E2  82
#define snd_Note_F2  87
#define snd_Note_FS2 93
#define snd_Note_G2  98
#define snd_Note_GS2 104
#define snd_Note_A2  110
#define snd_Note_AS2 117
#define snd_Note_B2  123
#define snd_Note_C3  131
#define snd_Note_CS3 139
#define snd_Note_D3  147
#define snd_Note_DS3 156
#define snd_Note_E3  165
#define snd_Note_F3  175
#define snd_Note_FS3 185
```

```
#define snd_Note_G3  196
#define snd_Note_GS3 208
#define snd_Note_A3  220
#define snd_Note_AS3 233
#define snd_Note_B3  247
#define snd_Note_C4  262
#define snd_Note_CS4 277
#define snd_Note_D4  294
#define snd_Note_DS4 311
#define snd_Note_E4  330
#define snd_Note_F4  349
#define snd_Note_FS4 370
#define snd_Note_G4  392
#define snd_Note_GS4 415
#define snd_Note_A4  440
#define snd_Note_AS4 466
#define snd_Note_B4  494
#define snd_Note_C5  523
#define snd_Note_CS5 554
#define snd_Note_D5  587
#define snd_Note_DS5 622
#define snd_Note_E5  659
#define snd_Note_F5  698
#define snd_Note_FS5 740
#define snd_Note_G5  784
#define snd_Note_GS5 831
#define snd_Note_A5  880
#define snd_Note_AS5 932
#define snd_Note_B5  988
#define snd_Note_C6  1047
#define snd_Note_CS6 1109
#define snd_Note_D6  1175
#define snd_Note_DS6 1245
#define snd_Note_E6  1319
#define snd_Note_F6  1397
#define snd_Note_FS6 1480
#define snd_Note_G6  1568
#define snd_Note_GS6 1661
#define snd_Note_A6  1760
#define snd_Note_AS6 1865
#define snd_Note_B6  1976
#define snd_Note_C7  2093
#define snd_Note_CS7 2217
#define snd_Note_D7  2349
#define snd_Note_DS7 2489
#define snd_Note_E7  2637
#define snd_Note_F7  2794
#define snd_Note_FS7 2960
#define snd_Note_G7  3136
#define snd_Note_GS7 3322
```

```
#define snd_Note_A7  3520
#define snd_Note_AS7 3729
#define snd_Note_B7  3951
#define snd_Note_C8  4186
#define snd_Note_CS8 4435
#define snd_Note_D8  4699
#define snd_Note_DS8 4978


// *** Sound-Funktionen ***

void snd_Init (byte soundGPIO)
{
    snd_GPIO = soundGPIO;
    noTone(snd_GPIO);                           // Ggf. bestehende Tonausgabe abschalten
    pinMode(snd_GPIO, OUTPUT);                  // GPIO auf Ausgabe schalten
    digitalWrite(snd_GPIO, LOW);                // GPIO auf 0V schalten
}

void snd_click ()
{
    tone(snd_GPIO, 1000, 5);                    // Klick erzeugen: 1 kHz Ton für 5 ms.
}

void snd_beep ()
{
    tone(snd_GPIO, 1000, 500);                  // Beep erzeugen: 1 kHz Ton für 500 ms.
}

void snd_music (long frequency, long duration)
{
    tone(snd_GPIO, frequency, duration);        // Ton ausgeben (Frequenz in Hz, Dauer in ms.)
}
```

Sound Library Dokumentation

Die Sound-Library stellt Funktionen zur einkanaligen Soundausgabe und Musikerzeugung zur Verfügung. Eine Erweiterung auf mehrere Kanäle ist problemlos möglich. Auf Grund der wenigen GPIOs des IoT-Bricks jedoch aktuell nicht einsetzbar. Eine Verstärkerschaltung wie in Beispiel 13 wird benötigt, um eine Lautstärke zu erreichen, die man deutlich hören kann. Von einem direkten Anschluß eines Lautsprechers wird abgeraten, da die hohe Spannung von 5 Volt den Lautsprecher beschädigen kann.

Variable für den zur Tonausgabe verwenden GPIO

```
byte snd_GPIO = 13;
```

Konstanten für Musik-Töne in 8 Oktaven

```
#define snd_Note_B0  31
#define snd_Note_C1  33
#define snd_Note_CS1 35
#define snd_Note_D1  37
#define snd_Note_DS1 39
#define snd_Note_E1  41
#define snd_Note_F1  44
#define snd_Note_FS1 46
#define snd_Note_G1  49
#define snd_Note_GS1 52
#define snd_Note_A1  55
#define snd_Note_AS1 58
#define snd_Note_B1  62
#define snd_Note_C2  65
#define snd_Note_CS2 69
#define snd_Note_D2  73
#define snd_Note_DS2 78
#define snd_Note_E2  82
#define snd_Note_F2  87
#define snd_Note_FS2 93
#define snd_Note_G2  98
#define snd_Note_GS2 104
#define snd_Note_A2  110
```

```
#define snd_Note_AS2 117
#define snd_Note_B2  123
#define snd_Note_C3  131
#define snd_Note_CS3 139
#define snd_Note_D3  147
#define snd_Note_DS3 156
#define snd_Note_E3  165
#define snd_Note_F3  175
#define snd_Note_FS3 185
#define snd_Note_G3  196
#define snd_Note_GS3 208
#define snd_Note_A3  220
#define snd_Note_AS3 233
#define snd_Note_B3  247
#define snd_Note_C4  262
#define snd_Note_CS4 277
#define snd_Note_D4  294
#define snd_Note_DS4 311
#define snd_Note_E4  330
#define snd_Note_F4  349
#define snd_Note_FS4 370
#define snd_Note_G4  392
#define snd_Note_GS4 415
#define snd_Note_A4  440
#define snd_Note_AS4 466
#define snd_Note_B4  494
#define snd_Note_C5  523
#define snd_Note_CS5 554
#define snd_Note_D5  587
#define snd_Note_DS5 622
#define snd_Note_E5  659
#define snd_Note_F5  698
#define snd_Note_FS5 740
#define snd_Note_G5  784
#define snd_Note_GS5 831
#define snd_Note_A5  880
#define snd_Note_AS5 932
#define snd_Note_B5  988
#define snd_Note_C6  1047
#define snd_Note_CS6 1109
```

```
#define snd_Note_D6  1175
#define snd_Note_DS6 1245
#define snd_Note_E6  1319
#define snd_Note_F6  1397
#define snd_Note_FS6 1480
#define snd_Note_G6  1568
#define snd_Note_GS6 1661
#define snd_Note_A6  1760
#define snd_Note_AS6 1865
#define snd_Note_B6  1976
#define snd_Note_C7  2093
#define snd_Note_CS7 2217
#define snd_Note_D7  2349
#define snd_Note_DS7 2489
#define snd_Note_E7  2637
#define snd_Note_F7  2794
#define snd_Note_FS7 2960
#define snd_Note_G7  3136
#define snd_Note_GS7 3322
#define snd_Note_A7  3520
#define snd_Note_AS7 3729
#define snd_Note_B7  3951
#define snd_Note_C8  4186
#define snd_Note_CS8 4435
#define snd_Note_D8  4699
#define snd_Note_DS8 4978
```

Sound-Funktionen

```
void snd_Init (byte soundGPIO)
```

Initialisiert die Sound-Ausgabe auf den GPIO mit der angegebenen Nummer.

```
void snd_click ()
```

Erzeugt einen Click im Lautsprecher (5 ms. Tonsignal mit 1 kHz Frequenz). Es ist dabei zu beachten, dass die Tonausgabe asynchron erfolgt, d.h. das Programm wird parallel zum Abspielen des Clicks fortgesetzt.

```
void snd_beep ()
```

Erzeugt einen Beep im Lautsprecher (500 ms. Tonsignal mit 1 kHz Frequenz). Es ist dabei zu beachten, dass die Tonausgabe asynchron erfolgt, d.h. das Programm wird parallel zum Abspielen des Clicks fortgesetzt.

```
void snd_music (long frequency, long duration, bool synchronous)
```

Erzeugt einen Musikton im Lautsprecher mit der angegebenen Frequenz in Herz und der angegebenen Dauer in Millisekunden. Für die Frequenz kann eine der vordefinierten Musik-Ton-Konstanten verwendet werden. Diese decken einen Umfang von knapp 8 Oktaven ab. Die Tonausgabe kann dabei wahlweise synchron oder asynchron erfolgen. Bei asynchroner Tonausgabe wird das Programm parallel zum Abspielen des Tons fortgesetzt. Um eine andauernde Tonausgabe zu stoppen, kann man `snd_Init(soundGPIO)` aufrufen.

DHT11 Library Listing

Die DHT11-Library stellt Funktionen zur Benutzung des kombinierten Luftfeuchtigkeit- und Temperatur-Sensors DHT11 zur Verfügung. Im Programm muß vor dem `#include` der Library die Datenleitung, z.B. GPIO12, mit dem Befehl `#define DHT11_PIN 12` definiert werden.

```
// DHT11 Library (Temperatur- und Luftfeuchtigkeits-Sensor)
// 1.00 - 2017-06-16
// (c) Copyright 2017
//
// Zur Verwendung mit dem Allnet IoT-Brick


#include <DHT.h>


// *** Globale Variablen ***

bool                DHT11_Inited                    = false;     // Variable für die Initialisierung der DHT11 Benutzung
DHT                 DHT11_Sensor(DHT11_PIN, DHT11);              // Variable vom Typ DHT definieren für Sensortyp DHT11


// *** Temperatur-Sensor Funktionen ***

void DHT11_Init ()
{
    DHT11_Inited = true;
    DHT11_Sensor.begin();                                          // Sensor initialisieren
}

double DHT11_Temperature ()
{
    if (DHT11_Inited)
    {
        return (DHT11_Sensor.readTemperature());
    }
    else
    {
        return (-127);
    }
}

double DHT11_Humidity ()
{
    if (DHT11_Inited)
```

```
    {
        return (DHT11_Sensor.readHumidity());
    }
    else
    {
        return (-127);
    }
}
```

DHT11 Library Dokumentation

Die DHT11-Library stellt Funktionen zur Benutzung des kombinierten Luftfeuchtigkeit- und Temperatur-Sensors DHT11 zur Verfügung. Im Programm muß vor dem `#include` der Library die Datenleitung, z.B. GPIO12, mit dem Befehl `#define DHT11_PIN 12` definiert werden.

Variablen für die Verwendung des DHT11-Sensors

```
bool                DHT11_Inited         = false;     // Variable für die Initialisierung der DHT11 Benutzung
DHT                 DHT11_Sensor(DHT11_PIN, DHT11);   // Variable vom Typ DHT definieren für Sensortyp DHT11
```

Sensor-Funktionen

`void DHT11_Init ()`

Initialisiert die Benutzung des DHT11-Sensors.

`double DHT11_Temperature ()`

Ergibt die aktuelle Temperatur am DHT11-Sensor oder -127, wenn der Sensor nicht initialisiert wurde.

`double DHT11_Humidity ()`

Ergibt die aktuelle Luftfeuchtigkeit am DHT11-Sensor oder -127, wenn der Sensor nicht initialisiert wurde.

DS18B20 Library Listing

Die DS18B20-Library stellt Funktionen zur Benutzung des Temperatur-Sensors DS18B20 zur Verfügung. Im Programm muß vor dem `#include` der Library die Datenleitung, z.B. GPIO14, mit dem Befehl `#define DS18B20_PIN 14` definiert werden.

```
// DS18B20 Library (Temperatur-Sensor)
// 1.00 - 2017-06-16
// (c) Copyright 2017
//
// Zur Verwendung mit dem Allnet IoT-Brick


#include <OneWire.h>
#include <DallasTemperature.h>


// *** Globale Variablen ***

bool                DS18B20_Inited                  = false;     // Variable für die Initialisierung der DS18B20 Benutzung
OneWire             DS18B20_oneWire(DS18B20_PIN);
DallasTemperature   DS18B20_Sensor(&DS18B20_oneWire);
DeviceAddress       DS18B20_SensorAddr;


// *** Temperatur-Sensor Funktionen ***

bool DS18B20_Init (bool messages)
{
    DS18B20_Inited = false;
    DS18B20_Sensor.begin();
    int numberOfDevices = DS18B20_Sensor.getDeviceCount();
    if (numberOfDevices > 0)
    {
        if (DS18B20_Sensor.getAddress(DS18B20_SensorAddr, 0))
        {
            if (DS18B20_Sensor.validFamily(DS18B20_SensorAddr))
            {
                DS18B20_Sensor.setResolution(DS18B20_SensorAddr, 9);
                if (messages)
                {
                    Serial.print("Valid Temperature Sensor found, resolution currently set to: ");
                    Serial.println(str(DS18B20_Sensor.getResolution(DS18B20_SensorAddr)));
                    Serial.print("Temperature is");
                }
                DS18B20_Sensor.requestTemperatures();                           // Temperatur anfordern. Ist notwendig, sonst erscheint immer 85°
                double t = DS18B20_Sensor.getTempC(DS18B20_SensorAddr);
```

```
            if (not(isnan(t)))
            {
                int TemperatureVal = int(t);
                if (TemperatureVal >= -100)
                {
                    if (messages)
                    { // 2 Nachkommastellen, mit Tausenderpunkt, mit ggf. Nullen nach dem Komma
                        String Temperature = strrealform (t, 32, 1, 0, true, false) + "°C";
                        Serial.println(": " + Temperature);
                    }
                    DS18B20_Inited = true;
                }
                else
                {
                    if (messages)
                    {
                        Serial.println(" not valid."); // -127 = Disconnected
                    }
                }
            }
            else
            {
                if (messages)
                {
                    Serial.println(" invalid.");
                }
            }
        }
        else
        {
            if (messages)
            {
                Serial.println("Invalid Temperature Sensor found.");
            }
        }
    }
    else
    {
        if (messages)
        {
            Serial.println("Temperature Sensor does not respond.");
        }
    }
}
else
{
    if (messages)
    {
        Serial.println("Temperature Sensor not found.");
    }
```

```
    }
    return (DS18B20_Inited);
}

double DS18B20_Temperature ()
{
    if (DS18B20_Inited)
    {
        return (DS18B20_Sensor.getTempC(DS18B20_SensorAddr));
    }
    else
    {
        return (-127);
    }
}
```

DS18B20 Library Dokumentation

Die DS18B20-Library stellt Funktionen zur Benutzung des Temperatur-Sensors DS18B20 zur Verfügung. Im Programm muß vor dem `#include` der Library die Datenleitung, z.B. GPIO14, mit dem Befehl `#define DS18B20_PIN 14` definiert werden.

Variablen für die Verwendung des DS18B20-Sensors

```
bool                DS18B20_Inited          = false;     // Variable für die Initialisierung der DS18B20 Benutzung
OneWire             DS18B20_oneWire(DS18B20_PIN);
DallasTemperature   DS18B20_Sensor(&DS18B20_oneWire);
DeviceAddress       DS18B20_SensorAddr;
```

Sensor-Funktionen

`bool DS18B20_Init (bool messages)`

Initialisiert die Benutzung des DHT11-Sensors und ergibt `true`, wenn die Initialisierung erfolgreich war. Mit dem Parameter `messages` können wahlweise Terminal-Ausgaben zum Status der Sensor-Initialisierung ausgegeben werden.

`double DS18B20_Temperature ()`

Ergibt die aktuelle Temperatur am DHT11-Sensor oder -127, wenn der Sensor nicht initialisiert wurde.

MQTT Library Listing

Die MQTT-Library stellt Funktionen zur Benutzung des MQTT-Protokolls zur Verfügung.

```
// MQTT Library
// 1.00 - 2017-07-04
// (c) Copyright 2017
//
// Zur Verwendung mit allen Allnet Libraries

#include <PubSubClient.h>
#include <Client.h>


// *** Allgemeine Globale Variablen ***

WiFiClient   MQTT_WifiClient;                                   // Der MQTT-Wifi-Client für den Verbindungsaufbau
PubSubClient MQTT_Client (MQTT_WifiClient);                     // Der MQTT-Kommunikations-Client

String MQTT_ServerURL           = "";                           // MQTT Server URL, z.B. "iot.allnet.de"
uint16 MQTT_ServerPort          = 0;                            // MQTT Server Port, normalerweise 1883
String MQTT_ServerClientID      = "";                           // Eine zufällige Unique-ID
String MQTT_ServerUserName      = "";                           // Benutzer-Name
String MQTT_ServerUserPW        = "";                           // Benutzer-Password
String MQTT_ServerTopic         = "";                           // Das Thema (als Pfad), das benutzt werden soll

String MQTT_EventQueue          = "";                           // MQTT Event-Warteschlange
bool   MQTT_EventAvail          = false;                        // Wird true, sobald ein Event verfügbar ist
uint32 MQTT_EventCurrentTime    = 0;                            // Millisekunden-Zeit des letzten Events oder 0 = Kein Event
String MQTT_EventCurrentTopic   = "";                           // Das Thema (als Pfad) des letzten Events
String MQTT_EventCurrentMessage = "";                           // Die Mitteilung zum Thema des letzten Events
bool   MQTT_EventBusy           = false;                        // True = Der Event-Handler ist grade aktiv, MQTT_EventQueue nicht verfügbar
bool   MQTT_EventTrace          = false;                        // True = Jeden Event sobald er ankommt auf dem Terminal ausgeben
bool   MQTT_ConnectTrace        = true;                         // True = Jeden Connect-Versuch auf dem Terminal ausgeben


// *** Globale Variablen für Werte vom Allnet-MQTT-Server ***

double MQTT_AllnetTemperaturHamburg      = 0.0;
double MQTT_AllnetTemperaturBerlin       = 0.0;
double MQTT_AllnetTemperaturFrankfurt    = 0.0;
double MQTT_AllnetTemperaturStuttgart    = 0.0;
double MQTT_AllnetTemperaturHannover     = 0.0;
double MQTT_AllnetTemperaturMuenchen     = 0.0;
double MQTT_AllnetTemperaturKoeln        = 0.0;
double MQTT_AllnetTemperaturDuesseldorf  = 0.0;
```

```
double MQTT_AllnetTemperaturNuernberg    = 0.0;
double MQTT_AllnetTemperaturDresden      = 0.0;
double MQTT_AllnetTemperaturNaumburg     = 0.0;
double MQTT_AllnetTemperaturHeidelberg   = 0.0;
double MQTT_AllnetTemperaturCottbus      = 0.0;
double MQTT_AllnetTemperaturSchwerin     = 0.0;
double MQTT_AllnetTemperaturMainz        = 0.0;
double MQTT_AllnetTemperaturWiesbaden    = 0.0;
double MQTT_AllnetTemperaturBremen       = 0.0;
double MQTT_AllnetTemperaturDortmund     = 0.0;
double MQTT_AllnetTemperaturKiel         = 0.0;
double MQTT_AllnetTemperaturLeipzig      = 0.0;
double MQTT_AllnetTemperaturRegensburg   = 0.0;
double MQTT_AllnetTemperaturRostock      = 0.0;
double MQTT_AllnetTemperaturSaarbruecken = 0.0;
double MQTT_AllnetTemperaturTrier        = 0.0;
double MQTT_AllnetTemperaturUlm          = 0.0;
double MQTT_AllnetTemperaturWuerzburg    = 0.0;
double MQTT_AllnetTemperaturOldenburg    = 0.0;
double MQTT_AllnetTemperaturPotsdam      = 0.0;
double MQTT_AllnetTemperaturMagdeburg    = 0.0;

double MQTT_AllnetFinanceDaxVal          = 0.0;
double MQTT_AllnetFinanceDaxDifVal       = 0.0;
double MQTT_AllnetFinanceDaxDifPerc      = 0.0;

double MQTT_AllnetFinanceMDaxVal         = 0.0;
double MQTT_AllnetFinanceMDaxDifVal      = 0.0;
double MQTT_AllnetFinanceMDaxDifPerc     = 0.0;

double MQTT_AllnetFinanceEStx50Val       = 0.0;
double MQTT_AllnetFinanceEStx50DifVal    = 0.0;
double MQTT_AllnetFinanceEStx50DifPerc   = 0.0;

double MQTT_AllnetFinanceGoldVal         = 0.0;
double MQTT_AllnetFinanceGoldDifVal      = 0.0;
double MQTT_AllnetFinanceGoldDifPerc     = 0.0;

double MQTT_AllnetFinanceOelVal          = 0.0;
double MQTT_AllnetFinanceOelDifVal       = 0.0;
double MQTT_AllnetFinanceOelDifPerc      = 0.0;

double MQTT_AllnetFinanceEURVal          = 0.0;
double MQTT_AllnetFinanceEURDifVal       = 0.0;
double MQTT_AllnetFinanceEURDifPerc      = 0.0;

double MQTT_AllnetFinanceBTCVal          = 0.0;
double MQTT_AllnetFinanceBTCDifVal       = 0.0;
double MQTT_AllnetFinanceBTCDifPerc      = 0.0;
```

```
double MQTT_AllnetFinanceETHVal          = 0.0;
double MQTT_AllnetFinanceETHDifVal       = 0.0;
double MQTT_AllnetFinanceETHDifPerc      = 0.0;


// *** MQTT-Funktionen ***

void MQTT_ClearEvents ()
{
    while (MQTT_EventBusy)    // Neuer Event wird gerade empfangen
    {
        delay(1);               // 1 Millisekunde warten
    }
    MQTT_EventQueue = "";
    MQTT_EventAvail = false;
    MQTT_EventCurrentTime = 0;
    MQTT_EventCurrentTopic = "";
    MQTT_EventCurrentMessage = "";
}

void MQTT_CallbackHandler (char* topic, byte* payload, unsigned int length)
{
    MQTT_EventBusy = true;
    uint32 m = micros();
    String Topic = cleanasc(String(topic));
    String Message = "";
    for (int i = 0; i < length; i++)
    {
        if (payload[i] != 3) // Ctrl-C ist nicht erlaubt, Event-Record-Trennzeichen
        {
            Message += (char)payload[i];
        }
    }
    if (MQTT_EventTrace)
    {
        Serial.print("Event " + String(m) + " [" + Topic + "] ");
        Serial.println(left(Message, 50));
    }
    MQTT_EventAvail = true;
    MQTT_EventQueue += String(m);
    MQTT_EventQueue += chr(9);
    MQTT_EventQueue += Topic;
    MQTT_EventQueue += chr(13);
    MQTT_EventQueue += Message;
    MQTT_EventQueue += chr(3);
    MQTT_EventBusy = false;
}

void MQTT_Disconnect ()
{
```

```
    MQTT_Client.disconnect();
}

void MQTT_Init (String serverURL, uint16 serverPort, String clientID, String userName, String userPassword)
{
    MQTT_ServerURL      = serverURL;
    MQTT_ServerPort     = serverPort;
    MQTT_ServerClientID = clientID;
    MQTT_ServerUserName = userName;
    MQTT_ServerUserPW   = userPassword;

    MQTT_Disconnect();
    MQTT_Client.setServer  (MQTT_ServerURL.c_str(), MQTT_ServerPort);
    MQTT_Client.setCallback(MQTT_CallbackHandler);
    MQTT_ClearEvents();
}

void MQTT_Init (IPAddress ipAdr, uint16 serverPort, String clientID, String userName, String userPassword)
{
    MQTT_ServerURL      = strIP(ipAdr, false);
    MQTT_ServerPort     = serverPort;
    MQTT_ServerClientID = clientID;
    MQTT_ServerUserName = userName;
    MQTT_ServerUserPW   = userPassword;

    MQTT_Disconnect();
    MQTT_Client.setServer  (ipAdr, MQTT_ServerPort);
    MQTT_Client.setCallback(MQTT_CallbackHandler);
    MQTT_ClearEvents();
}

int MQTT_GetNextEvent () // 0 = Kein Event vorhanden, 1 = Event erfolgreich gelesen, <0 = Fehlercodes
{
    if (MQTT_EventAvail)
    {
        while (MQTT_EventBusy)    // Neuer Event wird gerade empfangen
        {
            delay(1);               // 1 Millisekunde warten
        }
        int m = position(MQTT_EventQueue, chr(9), 1);
        if (m > 0)
        {
            MQTT_EventCurrentTime = val(mid(MQTT_EventQueue, 1, m - 1));
            int t = position(MQTT_EventQueue, chr(13), m + 1);
            if (t > 0)
            {
                MQTT_EventCurrentTopic = mid(MQTT_EventQueue, m + 1, t - m - 1);
                int p = position(MQTT_EventQueue, chr(3), t + 1);
                if (p > 0)
                {
```

```
                    MQTT_EventCurrentMessage = mid(MQTT_EventQueue, t + 1, p - t - 1);
                    MQTT_EventQueue = part(MQTT_EventQueue, p + 1);
                    MQTT_EventAvail = (MQTT_EventQueue != "");
                    return (1);
                }
                else
                {
                    return (-3); // Error: Event-Message nicht gefunden
                }
            }
            else
            {
                return (-2); // Error: Event-Topic nicht gefunden
            }
        }
        else
        {
            return (-1); // Error: Event-Time nicht gefunden
        }
    }
    else
    {
        return (0);
    }
}

bool MQTT_Connect (String topic)
{
    if (MQTT_ConnectTrace)
    {
        Serial.print("Looking for MQTT server '" + MQTT_ServerURL + "'... ");
    }
    bool success = false;
    MQTT_Client.disconnect();
    if ((MQTT_ServerUserName == "") && (MQTT_ServerUserPW == ""))
    {
        success = MQTT_Client.connect(MQTT_ServerClientID.c_str());
    }
    else
    {
        success = MQTT_Client.connect(MQTT_ServerClientID.c_str(), MQTT_ServerUserName.c_str(),  MQTT_ServerUserPW.c_str());
    }
    if (success)
    {
        if (MQTT_ConnectTrace)
        {
            Serial.println("Connected.");
        }
        MQTT_ServerTopic = topic;
        MQTT_Client.subscribe(MQTT_ServerTopic.c_str());
```

```
    }
    else
    {
        if (MQTT_ConnectTrace)
        {
            Serial.println("");
            Serial.print("MQTT server not connected. RC = ");
            Serial.println(MQTT_Client.state());
        }
    }
    return (success);
}

bool MQTT_HandleClient ()
{
    if (MQTT_Client.connected())
    {
        MQTT_Client.loop();
        return (true);
    }
    else
    {
        bool success = MQTT_Connect(MQTT_ServerTopic);
        MQTT_Client.loop();
        return (success);
    }
}

bool MQTT_AllnetParseEventCurrent ()
{
    bool result = true;
    double v = realval(MQTT_EventCurrentMessage);
    if (MQTT_EventCurrentTopic      == "general/data/temperatur/hamburg")
    {
        MQTT_AllnetTemperaturHamburg = v;
    }
    else if (MQTT_EventCurrentTopic == "general/data/temperatur/berlin")
    {
        MQTT_AllnetTemperaturBerlin = v;
    }
    else if (MQTT_EventCurrentTopic == "general/data/temperatur/frankfurt")
    {
        MQTT_AllnetTemperaturFrankfurt = v;
    }
    else if (MQTT_EventCurrentTopic == "general/data/temperatur/stuttgart")
    {
        MQTT_AllnetTemperaturStuttgart = v;
    }
    else if (MQTT_EventCurrentTopic == "general/data/temperatur/hannover")
    {
```

```
    MQTT_AllnetTemperaturHannover = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/muenchen")
{
    MQTT_AllnetTemperaturMuenchen = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/koeln")
{
    MQTT_AllnetTemperaturKoeln = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/duesseldorf")
{
    MQTT_AllnetTemperaturDuesseldorf = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/nuernberg")
{
    MQTT_AllnetTemperaturNuernberg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/dresden")
{
    MQTT_AllnetTemperaturDresden = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/naumburg")
{
    MQTT_AllnetTemperaturNaumburg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/heidelberg")
{
    MQTT_AllnetTemperaturHeidelberg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/cottbus")
{
    MQTT_AllnetTemperaturCottbus = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/schwerin")
{
    MQTT_AllnetTemperaturSchwerin = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/mainz")
{
    MQTT_AllnetTemperaturMainz = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/wiesbaden")
{
    MQTT_AllnetTemperaturWiesbaden = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/bremen")
{
    MQTT_AllnetTemperaturBremen = v;
}
```

```
else if (MQTT_EventCurrentTopic == "general/data/temperatur/dortmund")
{
    MQTT_AllnetTemperaturDortmund = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/kiel")
{
    MQTT_AllnetTemperaturKiel = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/leipzig")
{
    MQTT_AllnetTemperaturLeipzig = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/regensburg")
{
    MQTT_AllnetTemperaturRegensburg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/rostock")
{
    MQTT_AllnetTemperaturRostock = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/saarbruecken")
{
    MQTT_AllnetTemperaturSaarbruecken = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/trier")
{
    MQTT_AllnetTemperaturTrier = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/ulm")
{
    MQTT_AllnetTemperaturUlm = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/wuerzburg")
{
    MQTT_AllnetTemperaturWuerzburg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/oldenburg")
{
    MQTT_AllnetTemperaturOldenburg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/potsdam")
{
    MQTT_AllnetTemperaturPotsdam = v;
}
else if (MQTT_EventCurrentTopic == "general/data/temperatur/magdeburg")
{
    MQTT_AllnetTemperaturMagdeburg = v;
}
else if (MQTT_EventCurrentTopic == "general/data/finance/dax")    // "GER","DAX","7/4/2017","4:15pm",12468.32,"-6.99 - -0.06%"
{
```

```
    int p = position(MQTT_EventCurrentMessage, ":", 1);
    if (p > 0)
    {
        p = position(MQTT_EventCurrentMessage, ",", p + 1);
        if (p > 0)
        {
            MQTT_AllnetFinanceDaxVal = realval(part(MQTT_EventCurrentMessage, p + 1));
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceDaxDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceDaxDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                }
            }
        }
    }
}
else if (MQTT_EventCurrentTopic == "general/data/finance/mdax")   // "GER","MDAX","7/4/2017","4:15pm",24652.80,"-49.91 - -0.20%"
{
    int p = position(MQTT_EventCurrentMessage, ":", 1);
    if (p > 0)
    {
        p = position(MQTT_EventCurrentMessage, ",", p + 1);
        if (p > 0)
        {
            MQTT_AllnetFinanceMDaxVal = realval(part(MQTT_EventCurrentMessage, p + 1));
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceMDaxDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceMDaxDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                }
            }
        }
    }
}
else if (MQTT_EventCurrentTopic == "general/data/finance/estx50") // "ZRH","ESTX50 EUR P","7/4/2017","4:15pm",3490.65,"-1.16 - -0.03%"
{
    int p = position(MQTT_EventCurrentMessage, ":", 1);
    if (p > 0)
    {
        p = position(MQTT_EventCurrentMessage, ",", p + 1);
        if (p > 0)
        {
```

```
                MQTT_AllnetFinanceEStx50Val = realval(part(MQTT_EventCurrentMessage, p + 1));
                p = position(MQTT_EventCurrentMessage, ",", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceEStx50DifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                    p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                    if (p > 0)
                    {
                        MQTT_AllnetFinanceEStx50DifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                    }
                }
            }
        }
    }
    else if (MQTT_EventCurrentTopic == "general/data/finance/gold")   // "CMX","Gold Aug 17","7/4/2017","10:20am",1219.20,"-23.10 - -1.86%"
    {
        int p = position(MQTT_EventCurrentMessage, ":", 1);
        if (p > 0)
        {
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceGoldVal = realval(part(MQTT_EventCurrentMessage, p + 1));
                p = position(MQTT_EventCurrentMessage, ",", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceGoldDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                    p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                    if (p > 0)
                    {
                        MQTT_AllnetFinanceGoldDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                    }
                }
            }
        }
    }
    else if (MQTT_EventCurrentTopic == "general/data/finance/oel")    // "NYM","Light Sweet Crude Oil Futures,A","7/4/2017","10:20am",47.07,"+1.03 -
+2.24%"
    {
        int p = position(MQTT_EventCurrentMessage, ":", 1);
        if (p > 0)
        {
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceOelVal = realval(part(MQTT_EventCurrentMessage, p + 1));
                p = position(MQTT_EventCurrentMessage, ",", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceOelDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
```

```
                p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceOelDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                }
            }
        }
    }
}
else if (MQTT_EventCurrentTopic == "general/data/finance/eurusd")    // "CCY","EUR/USD","7/7/2017","10:15am",1.1420,"-0.0001 - -0.0131%".2615%"
{
    int p = position(MQTT_EventCurrentMessage, ":", 1);
    if (p > 0)
    {
        p = position(MQTT_EventCurrentMessage, ",", p + 1);
        if (p > 0)
        {
            MQTT_AllnetFinanceEURVal = realval(part(MQTT_EventCurrentMessage, p + 1));
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceEURDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceEURDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                }
            }
        }
    }
}
else if (MQTT_EventCurrentTopic == "general/data/finance/btc")    // "CCY","Bitcoin USD","7/7/2017","10:00am",2570.0801,"-6.9199 - -0.2685%"
{
    int p = position(MQTT_EventCurrentMessage, ":", 1);
    if (p > 0)
    {
        p = position(MQTT_EventCurrentMessage, ",", p + 1);
        if (p > 0)
        {
            MQTT_AllnetFinanceBTCVal = realval(part(MQTT_EventCurrentMessage, p + 1));
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceBTCDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceBTCDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                }
            }
```

```
            }
        }
    }
    else if (MQTT_EventCurrentTopic == "general/data/finance/ether")    // "CCY","Ethereum","7/21/2017","8:29am",237.0000,"+2.2000 - +0.9370%"
    {
        int p = position(MQTT_EventCurrentMessage, ":", 1);
        if (p > 0)
        {
            p = position(MQTT_EventCurrentMessage, ",", p + 1);
            if (p > 0)
            {
                MQTT_AllnetFinanceETHVal = realval(part(MQTT_EventCurrentMessage, p + 1));
                p = position(MQTT_EventCurrentMessage, ",", p + 1);
                if (p > 0)
                {
                    MQTT_AllnetFinanceETHDifVal = realval(part(MQTT_EventCurrentMessage, p + 2));
                    p = position(MQTT_EventCurrentMessage, " - ", p + 1);
                    if (p > 0)
                    {
                        MQTT_AllnetFinanceETHDifPerc = realval(part(MQTT_EventCurrentMessage, p + 3));
                    }
                }
            }
        }
    }
    else
    {
        result = false;
    }
    return (result);
}
```

MQTT Library Dokumentation

Die MQTT-Library stellt Funktionen zur Benutzung des MQTT-Protokolls zur Verfügung.

Allgemeine Variablen für die MQTT-Library

```
WiFiClient   MQTT_WifiClient;                // Der MQTT-Wifi-Client für den Verbindungsaufbau
PubSubClient MQTT_Client (MQTT_WifiClient);  // Der MQTT-Kommunikations-Client

String MQTT_ServerURL           = "";        // MQTT Server URL, z.B. "iot.allnet.de"
uint16 MQTT_ServerPort          = 0;         // MQTT Server Port, normalerweise 1883
String MQTT_ServerClientID      = "";        // Eine zufällige Unique-ID
String MQTT_ServerUserName      = "";        // Benutzer-Name
String MQTT_ServerUserPW        = "";        // Benutzer-Password
String MQTT_ServerTopic         = "";        // Das Thema (als Pfad), das benutzt werden soll

String MQTT_EventQueue          = "";        // MQTT Event-Warteschlange
bool   MQTT_EventAvail          = false;     // Wird true, sobald ein Event verfügbar ist
uint32 MQTT_EventCurrentTime    = 0;         // Millisekunden-Zeit des letzten Events oder 0 = Kein Event
String MQTT_EventCurrentTopic   = "";        // Das Thema (als Pfad) des letzten Events
String MQTT_EventCurrentMessage = "";        // Die Mitteilung zum Thema des letzten Events
bool   MQTT_EventBusy           = false;     // True = Der Event-Handler ist grade aktiv, MQTT_EventQueue nicht verfügbar
bool   MQTT_EventTrace          = false;     // True = Jeden Event sobald er ankommt auf dem Terminal ausgeben
bool   MQTT_ConnectTrace        = true;      // True = Jeden Connect-Versuch auf dem Terminal ausgeben
```

Globale Variablen für Werte vom Allnet-MQTT-Server

Alle folgenden Variablen werden von der Funktion `MQTT_AllnetParseEventCurrent()` mit Werten befüllt. Die Variablen für die Temperatur enthalten den Temperaturwert in der jeweiligen Stadt in Grad Celsius. Alle Werte sind auf zwei Nachkommastellen genau.

```
double MQTT_AllnetTemperaturHamburg      = 0.0;
double MQTT_AllnetTemperaturBerlin       = 0.0;
double MQTT_AllnetTemperaturFrankfurt    = 0.0;
double MQTT_AllnetTemperaturStuttgart    = 0.0;
double MQTT_AllnetTemperaturHannover     = 0.0;
double MQTT_AllnetTemperaturMuenchen     = 0.0;
```

```
double MQTT_AllnetTemperaturKoeln        = 0.0;
double MQTT_AllnetTemperaturDuesseldorf  = 0.0;
double MQTT_AllnetTemperaturNuernberg    = 0.0;
double MQTT_AllnetTemperaturDresden      = 0.0;
double MQTT_AllnetTemperaturNaumburg     = 0.0;
double MQTT_AllnetTemperaturHeidelberg   = 0.0;
double MQTT_AllnetTemperaturCottbus      = 0.0;
double MQTT_AllnetTemperaturSchwerin     = 0.0;
double MQTT_AllnetTemperaturMainz        = 0.0;
double MQTT_AllnetTemperaturWiesbaden    = 0.0;
double MQTT_AllnetTemperaturBremen       = 0.0;
double MQTT_AllnetTemperaturDortmund     = 0.0;
double MQTT_AllnetTemperaturKiel         = 0.0;
double MQTT_AllnetTemperaturLeipzig      = 0.0;
double MQTT_AllnetTemperaturRegensburg   = 0.0;
double MQTT_AllnetTemperaturRostock      = 0.0;
double MQTT_AllnetTemperaturSaarbruecken = 0.0;
double MQTT_AllnetTemperaturTrier        = 0.0;
double MQTT_AllnetTemperaturUlm          = 0.0;
double MQTT_AllnetTemperaturWuerzburg    = 0.0;
double MQTT_AllnetTemperaturOldenburg    = 0.0;
double MQTT_AllnetTemperaturPotsdam      = 0.0;
double MQTT_AllnetTemperaturMagdeburg    = 0.0;
```

Die Variablen für die Aktienwerte und Währungen enthalten den Aktienwert in Punkten bzw. die Währung umgerechnet in US-Dollar (**`...Val`**), die Differenz zum Vortag (**`...DifVal`**) sowie die prozentuale Differenz (**`...DifValPerc`**). Alle Werte sind auf zwei Nachkommastellen genau.

```
double MQTT_AllnetFinanceDaxVal          = 0.0;
double MQTT_AllnetFinanceDaxDifVal       = 0.0;
double MQTT_AllnetFinanceDaxDifPerc      = 0.0;

double MQTT_AllnetFinanceMDaxVal         = 0.0;
double MQTT_AllnetFinanceMDaxDifVal      = 0.0;
double MQTT_AllnetFinanceMDaxDifPerc     = 0.0;

double MQTT_AllnetFinanceEStx50Val       = 0.0;
double MQTT_AllnetFinanceEStx50DifVal    = 0.0;
double MQTT_AllnetFinanceEStx50DifPerc   = 0.0;
```

```
double MQTT_AllnetFinanceGoldVal         = 0.0;
double MQTT_AllnetFinanceGoldDifVal      = 0.0;
double MQTT_AllnetFinanceGoldDifPerc     = 0.0;

double MQTT_AllnetFinanceOelVal          = 0.0;
double MQTT_AllnetFinanceOelDifVal       = 0.0;
double MQTT_AllnetFinanceOelDifPerc      = 0.0;

double MQTT_AllnetFinanceEURVal          = 0.0;
double MQTT_AllnetFinanceEURDifVal       = 0.0;
double MQTT_AllnetFinanceEURDifPerc      = 0.0;

double MQTT_AllnetFinanceBTCVal          = 0.0;
double MQTT_AllnetFinanceBTCDifVal       = 0.0;
double MQTT_AllnetFinanceBTCDifPerc      = 0.0;

double MQTT_AllnetFinanceETHVal          = 0.0;
double MQTT_AllnetFinanceETHDifVal       = 0.0;
double MQTT_AllnetFinanceETHDifPerc      = 0.0;
```

MQTT-Funktionen

```
void MQTT_ClearEvents ()
```

Löscht die Warteschlange für alle MQTT-Events und löscht die Variablen für den zuletzt ausgelesenen Event. Wenn der zuletzt ausgelesene Event gelöscht oder bei Programmstart nocht nicht gesetzt ist, hat die Variable `MQTT_EventCurrentTime` den Wert 0.

```
void MQTT_CallbackHandler (char* topic, byte* payload, unsigned int length)
```

Diese Funktion wird automatisch aufgerufen und schreibt einen auftretenden MQTT-Event in die Warteschlange. Es besteht keinen Grund, diese Funktion manuell aufzurufen.

```
void MQTT_Disconnect ()
```

Trennt die aktuelle MQTT-Verbindung, die mit der Funktion `MQTT_Connect()` hergestellt wurde. Wenn keine Verbindung hergestellt ist, macht diese Funktion nichts.

```
void MQTT_Init (String serverURL, uint16 serverPort, String clientID,
                String userName, String userPassword)
```

Stellt eine MQTT-Verbindung zu dem Server mit der angegebenen URL und dem angegebenen Port her. Der Server-Port ist üblicherweise 1883. Es werden nur unverschlüsselte Verbindungen unterstützt. Die Client ID ist üblicherweise eine Unique-ID oder beliebig. Es gibt allerdings Server (z.B. mqtt.mydevices.com), die eine vorgegebene Client ID erwarten. Für Verbindungen, die eine Anmeldung mit Name und Password benötigen, sind die entsprechenden Werte anzugeben. Wird keine Anmeldung mit Namen und Password benötigt, wird ein leerer String für beide Parameter übergeben.

```
void MQTT_Init (IPAddress ipAdr, uint16 serverPort, String clientID,
                String userName, String userPassword)
```

Stellt eine MQTT-Verbindung zu dem Server mit der angegebenen IP-Adresse und dem angegebenen Port her. Der Server-Port ist üblicherweise 1883. Es werden nur unverschlüsselte Verbindungen unterstützt. Die Client ID ist üblicherweise eine Unique-ID oder beliebig. Es gibt allerdings Server (z.B. mqtt.mydevices.com), die eine vorgegebene Client ID erwarten. Für Verbindungen, die eine Anmeldung mit Name und Password benötigen, sind die entsprechenden Werte anzugeben. Wird keine Anmeldung mit Namen und Password benötigt, wird ein leerer String für beide Parameter übergeben.

```
int MQTT_GetNextEvent ()
```

Liest den nächsten Event aus der Warteschlange und speichert die Werte in den globalen Variablen `MQTT_EventCurrentTime` (der Wert des Millisekunden-Timers zum Zeitpunkt des Auftreten des Events), `MQTT_EventCurrentTopic` (das Thema, d.h. der vollständige Pfad des Events) und `MQTT_EventCurrentMessage` (die Mitteilung für diesen Event, d.h. die eigentlichen Daten, die man verarbeiten möchte).

```
bool MQTT_Connect (String topic)
```

Stellt eine MQTT-Verbindung zu dem zuvor mit `MQTT_Init()` ausgewählten Server her und abboniert das angegebene Thema. Sollen mehrere Themen abboniert werden, so kann man ein `#` als Joker verwenden, z.B. „`general/data/#`“.

`bool MQTT_HandleClient ()`

Diese Funktion muss mindestens einmal in der `loop()` Funktion aufgerufen werden. Dabei wird geprüft, ob eine Verbindung besteht und bei Bedarf mit `MQTT_Connect()` neu aufgebaut. Außerdem wird die Event-Bearbeitung durch den `MQTT_CallbackHandler()` im Hintergrund ermöglicht.

`bool MQTT_AllnetParseEventCurrent ()`

Untersucht den aktuellen Event (`MQTT_EventCurrentTopic` und `MQTT_EventCurrentMessage`) auf Werte, die der Allnet-MQTT-Server zur Verfügung stellt. Diese werden dann in die zugehörigen globalen Variablen für die entsprechenden Werte übernommen. Wenn ein Wert erfolgreich erkannt und übernommen wurde, wird ein `true` übergeben. Falls ein unbekannter Wert im Event steht, wird ein `false` übergeben und das Programm muss diesen Event selbst untersuchen.