Die neue NanoESP-Library mit MQTT, HTTP und Blynk-Support

Im Kontext des neues IoT Adventskalenders 2016 wurde die Library zum NanoESP um einige Funktionen erweitert. Neben den klassischen Befehlen zum Steuern des Moduls, gibt es in dieser Version auch zwei Sub-Libraries, die das Übermitteln von Webseiten erleichtern oder die Verwendung des beliebten MQTT(http://mqtt.org/) IoT-Protokolls ermöglichen.

Unter den Beispielen befindet sich außerdem ein Programm, welches Blynk-Projekte (http://www.blynk.cc/) ermöglicht. Damit Sie das Programm verwenden können, benötigen Sie eine zusätzliche Library:

https://github.com/blynkkk/blynk-library/releases/tag/v0.3.4

Auf dieser Seite möchte ich die einzelnen Funktionen und Methoden etwas detaillierter erläutern sowie auf etwaige Neuerungen eingehen. Die Basis der Library bildet die SoftwareSerial-Library, die bereits in Arduino integriert ist. Aufgrund dessen können Befehle wie find() und findUntil() auch mit dieser Library genutzt werden.

Inhalt

Basics (NanoESP)

Funktion	**Erläuterung**
boolean **init**(boolean vDebug=false);	Initialisiert das Board und stellt default Werte ein (Transfermode = 0, Multiple Connections = 1) Wenn der Debug-Parameter true ist, wird bei einem fehlerhaften Befehl Zusatzinformationen über den Seriellen Monitor ausgegeben
String **sendCom**(String command);	Sendet ein AT-Kommando an das ESP-Modul
boolean **sendCom**(String command, char respond[]);	s.o.; Zusätzlich wird überprüft, ob das Modul eine Antwort sendet, die dem Respond-Parameter entspricht.
boolean **setMultipleConnections**();	Erlaubt Mehrfachverbindungen (*AT+CIPMUX=1*)
boolean **setTransferMode**() ;	Setzt den Transfermodus auf transparent (*AT+CIPMODE=0*)
boolean **reset**();	Resetet das Board und wartet, bis es wieder bereit ist

boolean **configWifiMode**(int modus);	Stellt den WLAN Modus ein (*STATION, ACCESSPOINT, DUAL*)
boolean **configWifi**(int modus, String ssid, String password);	Stellt das WLAN ein mit Modus, SSID und Password
boolean **configWifiStation**(String ssid, String password);	Stellt eine Verbindung zu einem WLAN-Router her
boolean **configWifiAP**(String ssid, String password);	Stellt einen eignen AccesPoint zur Verfügung (Password darf leer sein, sonst min sechs Zeichen)
boolean **configWifiAP**(String ssid, String password, int channel, int crypt);	S.o. Zusätzliche Parameter: WLAN-Kanal, Verschlüsselungsmodus
boolean **disconnectWifi**();	Trennt die WLAN-Verbindung
bool **wifiConnected**();	Überprüft, ob automatisch eine WLAN-Verbindung zu einer Station hergestellt wurde (Zeitersparnis gegenüber neue Verbindung bei bekannter Station)
bool **getIpMac**(String &ip, String &mac);	Liefert IP und MAC-Adresse des Boards. Parameter werden als Referenz übergeben
String **getIp**();	Ermittelt die IP/IPs des Boards (*AT+CIFSR*)
boolean **newConnection**(int id, String type, String ip , unsigned int port);	Baut eine neue Verbindung (TCP oder UDP) auf. Parameter: Verbindungs-ID (0-4), Typ (TCP/UDP), IP (Ziel IP oder Adresse), Port
boolean **closeConnection**(int id) ;	Trennt die Verbindung mit der angegebenen Verbindungs-ID
boolean **startUdpServer**(int id, String ip , unsigned int port, unsigned int recvport, int mode=0);	Öffnet eine UDP-Verbindung Der Empfangs-Port kann ein anderer sein als der Sende-Port. Modus 0: Die Ziel IP ändert sich nicht Modus 1: die Ziel IP ändert sich einmal, wenn das Board eine Nachricht von einer anderen IP erhält. Modus 2: Das Board ändert die Ziel IP immer, wenn es eine neue Nachricht von einer anderen IP erhalten hat.
boolean **endUdpServer**(int id);	Beendet die UDP-Verbindung mit der angegeben ID
boolean **startTcpServer**(unsigned int port) ;	Startet einen TCP-Server unter dem angegeben Port. Es kann nur ein TCP-Server aktiviert sein
boolean **endTcpServer**();	Beendet den TCP-Server
boolean **sendData**(int id, String msg);	Sendet Text über die angegebene Verbindung
boolean **sendDataClose**(int id, String msg);	Sendet Text und schließt danach die Verbindung
bool **sendRaw**(int id, unsigned char data[], int LenChar);	Sendet binäre Daten (kein String)

int **getId**();	Gibt die Verbindungs-ID bei Datenempfang an. Wenn keine Daten empfangen wurden ist der Wert -1
bool **recvData**(int &id,int &len);	Wird true, wenn Daten empfangen wurden. ID und Länge werden per Referenz zurückgegeben
int **ping**(String adress);	Pingt den Server unter der angegebenen Adresse an. Die Antwort-Zeit in ms wird zurückgegeben. Ist die Zeit 0 wurde keine Antwort empfangen.
void **serialDebug**();	Stellt eine direkte Verbindung zwischen Software und Hardware Serieller Schnittstelle her. Kann zum Testen von AT-Kommandos verwendet werden.

Webserver (NanoESP_HTTP)

Funktion	**Erläuterung**
bool **recvRequest**(int &id, String &method, String &ressource, String &parameter);	Gibt true zurück, wenn ein http-Request empfangen wurde. ID, Methode (POST oder GET) sowie angeforderte URL und etwaige Befehle werden per Referenz zurückgegeben.
bool **recvHTTP**(int id, int len, String &method, String &ressource, String &parameter);	s.o.
bool **sendFromFlash**(int client, const char *website, int len);	Sendet eine als Progmem-Variable gespeicherte Webseite direkt an den angeben Client (Verbindungs-ID)
bool **sendStreamHeader**(int connectionId);	Sendet die Antwort auf einen Event-Stream-Request. Server-Send-Events können zum schnellen Datenaustausch vom Server zum Client genutzt werden (http://www.html5rocks.com/en/tutorials/eventsource/basics/)
bool **sendRequest**(int id, char method[5], String address);	Sendet einen http-Request an einen Server (ID, Methode (GET/POST), URL)
bool **sendRequest**(int id, char method[5], String address, String parameter);	Sendet einen http-Request an einen Server inklusive Sub-Parameter

MQTT (NanoESP_MQTT)

Funktion	**Erläuterung**
bool **connect**(int id, String brooker, unsigned int port, String deviceId);	Baut eine Verbindung zu einem MQTT-Server auf. Minimale Parameter: Verbindungs-ID, Broker-Adresse und Geräte-ID (darf nur einmal auf dem Broker vorhanden sein) (Defaultwerte: Clean Session = true, keepAliveTime = 120s)
bool **connect**(*[s.o]*, mqtt_msg * lastWill);	Erweitere Parameter der Connect-Funktion: Parameter: lastWill-Nachricht
bool **connect**(*[s.o]* , bool cleanSession, byte keepAliveTime);	Erweitere Parameter der Connect-Funktion: Parameter: cleanSession, keepAliveTime

bool **connect**(*[s.o]* , bool cleanSession, byte keepAliveTime, mqtt_msg * lastWill);	Erweitere Parameter der Connect-Funktion: Parameter: cleanSession, keepAliveTime, lastWill-Nachricht
bool **connect**(*[s.o]*, String userName , String password);	Erweitere Parameter der Connect-Funktion: Parameter: Nutzername und Passwort für Broker mit Benutzerverwaltung
bool **connect***([s.o]*, bool cleanSession, byte keepAliveTime, mqtt_msg * lastWill , String userName , String password);	Erweitere Parameter der Connect-Funktion
bool **disconnect**(int id);	Trennt Verbindung zum MQTT-Broker
bool **subscribe**(int id, String topic) ;	Abonniert ein Topic
bool **subscribe**(int id, String topic, byte qos);	Abonniert ein Topic mit Qos
bool **subscribe**(int id, String topic, byte qos, void (*g)(String value));	Abonniert ein Topic und weißt dem Topic eine Funktion zu, die aufgerufen wird, wenn eine Nachricht unter dem Topic empfangen wird (s.u. für weitere Erklärungen)
bool **unsubscribe**(int id, String topic) ;	Deabonniert ein Topic
bool **publish**(int id, String topic, String value);	Veröffentlicht Nachricht unter dem angeben Topic
bool **publish**(int id, String topic, String value, byte qos, bool retain);	Veröffentlicht Nachricht unter dem angeben Topic Zusätzliche Parameter: Quality of Service, Retain
bool **publish**(int id, mqtt_msg *msg);	Veröffentlicht Nachricht unter dem angeben Topic (Datentyp mqtt_msg s.u.)
bool **recvMQTT**(int id, int len, String &topic, String &value);	Gibt True zurück wenn eine MQTT-Nachricht empfangen wurde. Topic und Nachricht per Referenz
bool **recvMQTT**(int &id, String &topic, String &value) ;	Gibt True zurück wenn eine MQTT-Nachricht empfangen wurde. ID, Topic und Nachricht per Referenz
void **stayConnected**(int id);	Hält die Verbindung zum Broker aktiv, indem in regelmäßigen Abständen (i.d.R. StayConnected-Zeit/2) ein Ping gesendet wird
bool **ping**(int id);	Pingt den MQTT-Broker an (damit Verbindung bestehen bleibt)

Datentyp mqtt_msg

Mqtt_msg ist eine Struktur, über die eine MQTT-Nachricht definiert werden kann.

```
typedef struct{
    String topic;
    String value;
    byte qos;
    bool retain;
} mqtt_msg;
```

Beispiel:

```
mqtt_msg msgMode = {"NanoESP/test/rgb/mode","1“, 0, true};
```

Es können auch einzelne Werte der Nachricht verändert werden:

msgMode.value = „0“;

Abonnieren mit Funktions-Verknüpfung:

In dem Befehl

bool **subscribe**(int id, String topic, byte qos, void (*g)(String value));

kann als letzter Parameter eine Funktion angeben werden, die dann aufgerufen wird, wenn zu diesem Topic eine Nachricht empfangen wurde (Wildcards erlaubt). Die Funktion muss einen Parameter vom Typ String haben, der die Nachricht des Topics enthält. Es können maximal 5 dieser Funktions-Verknüpfungen definiert werden.

Beispiel:

```
mqtt.subscribe(0, „test/song", 2, gotnewSong)


void gotnewSong(String value) {
  Serial.println("New Song: " + value);
}
```