PANDORA PRODUCTS

# ESP32 Libraries
ESP32

Jim Schimpf

Document Number: PAN-201805001
Revision Number: 0.7
2 June 2018

Pandora Products.
215 Uschak Road
Derry, PA 15627

Creative Commons Attribution 4.0 International License 2018 Pandora Products. All other product names mentioned herein are trademarks or registered trademarks of their respective owners.

Pandora Products.

215 Uschak Road

Derry, PA 15627

Phone: 724-539.1276

Email: jim.schimpf@gmail.com

Pandora Products. has carefully checked the information in this document and believes it to be accurate. However, Pandora Products assumes no responsibility for any inaccuracies that this document may contain. In no event will Pandora Products. be liable for direct, indirect, special, exemplary, incidental, or consequential damages resulting from any defect or omission in this document, even if advised of the possibility of such damages.

In the interest of product development, Pandora Products reserves the right to make improvements to the information in this document and the products that it describes at any time, without notice or obligation.

## Document Revision History

| Version | Author | Description | Date |
|---|---|---|---|
| 0.1 | js | Initial Version | 10-May-2018 |
| 0.2 | js | Add EEPROM library | 11-May-2018 |
| 0.3 | js | Finish EEPROM and put in ESP32 | 12-May-2018 |
| 0.4 | js | mDNS added | 13-May-2018 |
| 0.5 | js | HTTPClient | 14-May-2018 |
| 0.6 | js | Preferences and forward | 23-May-2018 |
| 0.7 | js | Last library (Wire) added | 2-Jun-2018 |

# Contents

# 1 Introduction

The Adafruit HUZZA32 Feather[2] is a development board that features the ESP32 an advanced version of the ESP8266[1]. It is dual core so one core can run your application and the other can run the TCP/IP WiFi process. Unlike the ESP8266 you don't have to yield, your application can run continuously. The development environment builds on the Arduino system[3].

The ESP/32 has a set of libraries for Arduino development. These are **not** documented like the standard Arduino libraries. All that is there are examples for the libraries and the source code. This document will attempt to document the API for each library.

## 1.1 Contributors

I would like to thank the writers of the libraries and examples without which I could not write this.

| **Library** | **Cxntributer** |
|---|---|
| ArduinoOTA | Ivan Grokhotkov and Hristo Gochkov |
| BlueToothSerial | Evandro Luis Copercini |
| DNSServer | Kristijan Novoselić |
| EEPROM | Ivan Grokhotkov |
| ESP32 | Hristo Gochkov, Ivan Grokhtkov |
| ESPmDNS | Hristo Gochkov, Ivan Grokhtkov |
| FS | Hristo Gochkov, Ivan Grokhtkov |
| HTTPClient | Markus Sattler |
| Preferences | Hristo Gochkov |
| SD | Arduino, SparkFun |
| SD_MMC | Hristo Gochkov, Ivan Grokhtkov |
| SimpleBLE | Hristo Gochkov |
| SPI | Hristo Gochkov |
| SPIFFS | Hristo Gochkov, Ivan Grokhtkov |
| Ticker | Bert Melis |
| Update | Hristo Gochkov |
| WiFi | Hristo Gochkov |
| WiFiClientSecure | Evandro Luis Copercini |
| Wire | Hristo Gochkov |

Table 1: Contributors

# 2 Libraries

## 2.1 ArduinoOTA

### 2.1.1 Use

This library allows over-the-air updates to Arduino applications.

```
#include "ArduinoOTA.h"
ArduinoOTA update = ArduinoOTA(); // Create update object
```

### 2.1.2 API

#### 2.1.2.1 setPort

```
ArduinoOTAClass& setPort(uint16_t port);
```

| Output | Parameter | Meaning |
|---|---|---|
| | port | OTA update will use this port Default: 3232 |
| OTAClass | | Pointer to OTAClass |

#### 2.1.2.2 setHostname

```
ArduinoOTAClass& setHostname(const char *hostname);
```

| Output | Parameter | Meaning |
|---|---|---|
| | hostname | New hostname Default: esp32-xxxxxx |
| OTAClass | | Pointer to OTAClass |

#### 2.1.2.3 getHostname

```
String getHostname();
```

| Output | Parameter | Meaning |
|---|---|---|
| hostname | | Get current hostname |

#### 2.1.2.4 setPassword

```
ArduinoOTAClass& setPassword(const char *password);
```

| Output | Parameter | Meaning |
|---|---|---|
| | password | New Update password Default: NULL |
| OTAClass | | Pointer to OTAClass |

#### 2.1.2.5 setPasswordHash

```
ArduinoOTAClass& setPasswordHash(const char *password);
```

Set password in the form of an MD5 password

| Output | Parameter | Meaning |
|---|---|---|
| | password | New Update password Default: NULL |
| OTAClass | | Pointer to OTAClass |

#### 2.1.2.6 setRebootOnSuccess

```
ArduinoOTAClass& setRebootOnSuccess(bool reboot);
```

| Output | Parameter | Meaning |
|---|---|---|
| | reboot | Reboot on download success Default: true |
| OTAClass | | Pointer to OTAClass |

#### 2.1.2.7 setMdnsEnabled

```
ArduinoOTAClass& setMdnsEnabled(bool enabled);
```

Set if device should advertise to the Arduino IDE.

| Output | Parameter | Meaning |
|---|---|---|
| | enabled | Advertise to Arduino IDE Default: true |
| OTAClass | | Pointer to OTAClass |

#### 2.1.2.8 Call back functions

```
typedef std::function<void(void)> THandlerFunction;
typedef std::function<void(ota_error_t)> THandlerFunction_Error;
typedef std::function<void(unsigned int, unsigned int)> THandlerFunction_Pro
    //This callback will be called when OTA connection has begun
    ArduinoOTAClass& onStart(THandlerFunction fn);
    //This callback will be called when OTA has finished
    ArduinoOTAClass& onEnd(THandlerFunction fn);
    //This callback will be called when OTA encountered Error
    ArduinoOTAClass& onError(THandlerFunction_Error fn);
```

## 2.2 BlueToothSerial

### 2.2.1 Use

This library allows a serial connection over BlueTooth between the ESP32 and another BlueTooth device. The library is very similar to the Serial library of the standard Arduino

```
#include "BlueToothSerial.h"
BlueToothSerial btser = BlueToothSerial();
```

### 2.2.2 API

#### 2.2.2.1 begin

```
bool begin(String localName=String());
```

| Output | Parameter | Meaning |
|---|---|---|
| | localName | BlueTooth name to pair with |
| bool | | True if accepted |

#### 2.2.2.2 available

```
int available(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | # of chars in queue or 0 if none or no client |

#### 2.2.2.3 peek

```
int peek(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | next char or 0 if none or no client |

#### 2.2.2.4 hasClient

```
bool hasClient(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | True if connected to a client |

#### 2.2.2.5 read

```
int read(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | Next char from client |

#### 2.2.2.6 write

```
size_t write(uint8_t c);
```

| Output | Parameter | Meaning |
|---|---|---|
| | c | Character to send to Client |
| size_t | | # Characters written |

#### 2.2.2.7 write

```
size_t write(const uint8_t *buffer, size_t size);
```

| Output | Parameter | Meaning |
|---|---|---|
| | buffer | buffer to send to Client |
| | size | # Characters in buffer |
| size_t | | # Characters written |

#### 2.2.2.8 flush

```
void flush();
```

| Output | Parameter | Meaning |
|---|---|---|
| none | | Flush all characters in transit |

#### 2.2.2.9 end

```
void end;
```

| Output | Parameter | Meaning |
|---|---|---|
| none | | Drop connection |

## 2.3 DNSServer

### 2.3.1 Use

This is a conventional DNS server that can be run over the WiFi connection. As such I am not going to detail the API as the example included in the library makes it quite clear.

```
#include "DNSServer.h"
DNSServer dnsServer = DNSServer();
```

## 2.4 EEPROM

### 2.4.1 Use

This library uses a section of Flash memory as a non-volatile memory store. It can act like the EEPROM library in the standard Arduino. You must generate partitions in the Flash and define their location and size before any use. Max size = 0x1000 (4K). You need the partition manager (go to https://github.com/francis94c/ESP32Partitions and install it).

Example:

```
/* Generated partition that would work perfectly with this example
  #Name,    Type, SubType, Offset,   Size,    Flags
  nvs,      data, nvs,     0x9000,   0x5000,
  otadata,  data, ota,     0xe000,   0x2000,
  app0,     app,  ota_0,   0x10000,  0x140000,
  app1,     app,  ota_1,   0x150000, 0x140000,
  eeprom0,  data, 0x99,    0x290000, 0x1000,
  eeprom1,  data, 0x9a,    0x291000, 0x500,
  eeprom2,  data, 0x9b,    0x292000, 0x100,
  spiffs,   data, spiffs,  0x293000, 0x16d000,
*/
```

Then you could instantiate various EEPROM classes as:

```
#include "EEPROM.h"
// Instantiate eeprom objects with parameter/argument
// names and size same as in the partition table
EEPROMClass  NAMES("eeprom0", 0x1000);
EEPROMClass  HEIGHT("eeprom1", 0x500);
EEPROMClass  AGE("eeprom2", 0x100);
```

### 2.4.2 API

#### 2.4.2.1 begin

```
bool begin(size_t size);
```

| Output | Parameter | Meaning |
|---|---|---|
| | size | # bytes in EEPROM section |
| bool | | Area Setup |

#### 2.4.2.2 read

```
uint8_t read(int address);
```

| Output | Parameter | Meaning |
|---|---|---|
| | address | 0 based address in EEPROM area, address < size |
| uint8_t | | Returned byte |

#### 2.4.2.3 write

```
void write(int address, uint8_t val);
```

| Output | Parameter | Meaning |
|---|---|---|
| | address | 0 based address in EEPROM area, address < size |
| | val | Byte to write |
| none | | Data queued into EERPROM area |

#### 2.4.2.4 length

```
uint16_t length();
```

| Output | Parameter | Meaning |
|---|---|---|
| uint16_t | | Size of EEPROM area <= 4K |

#### 2.4.2.5 commit

```
bool commit();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | All data written to EEPROM area, non-volatile now |

#### 2.4.2.6 end

```
void end();
```

| Output | Parameter | Meaning |
|---|---|---|
| None | | EEPROM area closed |

**2.4.2.7 Specialized functions** These are helper functions that read/write different data types from/to EEPROM

```
uint8_t readByte(int address);
int8_t readChar(int address);
uint8_t readUChar(int address);
int16_t readShort(int address);
uint16_t readUShort(int address);
int32_t readInt(int address);
uint32_t readUInt(int address);
int32_t readLong(int address);
uint32_t readULong(int address);
int64_t readLong64(int address);
uint64_t readULong64(int address);
float_t readFloat(int address);
double_t readDouble(int address);
bool readBool(int address);
size_t readString(int address, char* value, size_t maxLen);
String readString(int address);
size_t readBytes(int address, void * value, size_t maxLen);
template <class T> T readAll (int address, T &);
size_t writeByte(int address, uint8_t value);
size_t writeChar(int address, int8_t value);
size_t writeUChar(int address, uint8_t value);
size_t writeShort(int address, int16_t value);
size_t writeUShort(int address, uint16_t value);
size_t writeInt(int address, int32_t value);
size_t writeUInt(int address, uint32_t value);
size_t writeLong(int address, int32_t value);
size_t writeULong(int address, uint32_t value);
size_t writeLong64(int address, int64_t value);
size_t writeULong64(int address, uint64_t value);
size_t writeFloat(int address, float_t value);
size_t writeDouble(int address, double_t value);
size_t writeBool(int address, bool value);
size_t writeString(int address, const char* value);
size_t writeString(int address, String value);
```

```
        size_t writeBytes(int address, const void* value, size_t len);
        template <class T> T writeAll (int address, const T &);
```

**NOTE:** Remember to advance the next written address by the **size_t** returned by each function.

### 2.4.3 ESP32

**2.4.3.1 Use** This is not a library but a collection of examples showing how to use new features of the ESP32. It does not have an API but the examples access ESP32 hooks to use the features.

**2.4.3.2 AnalogOut** How to use the ledAnalogWrite() for LED control and the sigmaDelta() output for voltage outputs.

**2.4.3.3 ChipID** Read out of the ESP32 chip ID.

**2.4.3.4 DeepSleep** How to use the processor shutdown (Deep Sleep) call.

**2.4.3.5 ESPNow** Connects two ESP32's a "master" and a "slave" over TP/IP WiFi.

**2.4.3.6 Hall Sensor** Use of the Hall magnetic sensor in the ESP32.

**2.4.3.7 ResetReason** Gets startup reason for coming out of deep sleep

**2.4.3.8 Time** Uses WiFi connected ESP32 to get the real time from a master system and set the RTC.

**2.4.3.9 Timer** Use of the WatchDog and other hardware timers in the ESP32.

**2.4.3.10 Touch** Use of the touch sensors built into the ESP32.

## 2.5 ESPmDNS

### 2.5.1 Use

mDNS (Multicast DNS[4] see `https://en.wikipedia.org/wiki/Multicast_DNS`) is a system that advertises services from a server over a network segment. This is particularly useful for the ESP32 since when it attaches to a WiFi network it DHCP's an address and mDNS is one of the few ways it can make that address known to clients.

To use this service first a WiFi server must be set up and a connection made with the WiFi access point.

```
#include <ESPmDNS.h>
```

The include file supplies a MDNSResponder class MDNS

### 2.5.2 API

#### 2.5.2.1 begin

```
bool begin(const char* hostName);
```

| Output | Parameter | Meaning |
|---|---|---|
| | hostname | Hostname part of <hostname>.local mDNS string |
| bool | | mDNS started |

#### 2.5.2.2 end

```
void end();
```

| Output | Parameter | Meaning |
|---|---|---|
| none | | mDNS stopped |

#### 2.5.2.3 setInstanceName

Sets the value of **<instance name>**.<type>.<protocol>.<domain>.local

```
void setInstanceName(String name);
void setInstanceName(const char * name);=
void setInstanceName(char * name);
```

| Output | Parameter | Meaning |
|---|---|---|
| | name | Instance Name |
| none | | |

#### 2.5.2.4 addService

Sets up advertising for a new service

```
 void addService(char *service, char *proto, uint16_t port);
void addService(const char *service, const char *proto, uint16_t port);
void addService(String service, String proto, uint16_t port);
```

| Output | Parameter | Meaning |
|---|---|---|
| | service | Service name |
| | protocol | Service protocol |
| | port | Port used by service |
| none | | |

#### 2.5.2.5 addServiceTxt Sets up advertising for a service

```
 bool addServiceTxt(char *name, char *proto, char * key, char * value);
void addServiceTxt(const char *name, const char *proto, const char *key,const char * value );
void addServiceTxt(String name, String proto, String key, String value);
```

| Output | Parameter | Meaning |
|---|---|---|
| | service | Service name |
| | protocol | Service protocol |
| | port | Port used by service |
| bool | | True if service advertised |

#### 2.5.2.6 enableArduino I don't know what this does

```
void enableArduino(uint16_t port=3232, bool auth=false);
```

| Output | Parameter | Meaning |
|---|---|---|
| | port | Default = 3232 |
| | auth | Default = false |
| none | | |

#### 2.5.2.7 disableArduino This turns it off.

```
void disableArduino();
```

| Output | Parameter | Meaning |
|---|---|---|
| none | | Disable Arduino advertising |

#### 2.5.2.8 enableWorkstation I don't know what this does

```
void enableWorkstation(wifi_interface_t interface=ESP_IF_WIFI_STA);
```

| Output | Parameter | Meaning |
|---|---|---|
| | interface | Enable advertising on this interface Default: WIFI |
| none | | Enable mDNS advertising |

#### 2.5.2.9 disableWorkstation

```
void disableWorkstation();
```

| Output | Parameter | Meaning |
|---|---|---|
| none | | Disable workstation advertising |

**2.5.2.10 queryHost** This returns the IP address of the selected host

```
 IPAddress queryHost(char *host, uint32_t timeout=2000);
IPAddress queryHost(const char *host, uint32_t timeout=2000);
IPAddress queryHost(String host, uint32_t timeout=2000);
```

| Output | Parameter | Meaning |
|---|---|---|
| | host | Host name |
| | timeout | Timeout on query Default = 2 seconds |
| IPAddress | | Host IP address or NULL |

**2.5.2.11 queryService** Return the # of mDNS advertisers that have matching service and protocol.

```
int queryService(char *service, char *proto);
int queryService(const char *service, const char *proto);
int queryService(String service, String proto);
```

| Output | Parameter | Meaning |
|---|---|---|
| | service | Service searched for |
| | proto | Protocol searched for3 |
| int | | # Found |

**2.5.2.12 hostname** After successful queryService() call use value 0 <-> n returned to get the specific service information

```
String hostname(int idx);
```

| Output | Parameter | Meaning |
|---|---|---|
| | idx | 0<idx<n from successful queryService |
| String | | Hostname |

**2.5.2.13 IP** After successful queryService() call use value 0 <-> n returned to get the specific service information

```
IPAddress IP(int idx);
```

| Output | Parameter | Meaning |
|---|---|---|
| | idx | 0<idx<n from successful queryService |
| IPAddress | | Service IP address |

#### 2.5.2.14 IPv6

After successful queryService() call use value 0 <-> n returned to get the specific service information

```
IPv6Address IPv6(int idx);
```

| Output | Parameter | Meaning |
|---|---|---|
| | idx | 0<idx<n from successful queryService |
| IPv6Address | | Service IPv6 address |

#### 2.5.2.15 port

After successful queryService() call use value 0 <-> n returned to get the specific service information

```
uint16_t port(int idx);
```

| Output | Parameter | Meaning |
|---|---|---|
| | idx | 0<idx<n from successful queryService |
| int | | Service port # |

## 2.6 FS

#### 2.6.0.1 Use

This appears to be a support class and there are no included examples so I am not going to document it. If you know more please feel free to add your knowledge.

## 2.7 HTTPClient

### 2.7.1 Use

This is used to allow the ESP32 to access s webserver. This can connect and log into servers.

### 2.7.2 API - Connection

#### 2.7.2.1 begin

This starts up the webserver

```
bool begin(String url);
bool begin(String url, const char* CAcert);
bool begin(String host, uint16_t port, String uri = "/");
bool begin(String host, uint16_t port, String uri, const char* CAcert);
bool begin(String host, uint16_t port, String uri, const char* CAcert, const char* cli_cert, const c
```

| Output | Parameter | Meaning |
|---|---|---|
| | host | Host string (optional) |
| | port | Port # used (optional) |
| | uri | URI of web server |
| | CAcert | ??? (optional) |
| | cli_cert | Web site cert (optional) |
| | cli_key | Web site key (optional) |
| bool | | True if connected |

#### 2.7.2.2 **end** Drop connection to web site

```
        void end(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| None | | Drop connection |

#### 2.7.2.3 **connected** Determine if connection is active

```
        bool connected(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | True if connection is active |

#### 2.7.2.4 **setReuse** This keeps the connection alive

```
        void setReuse(bool reuse); /// keep-alive
```

| Output | Parameter | Meaning |
|---|---|---|
| | reuse | True to keep connection action |
| NONE | | |

#### 2.7.2.5 **setUserAgent** Allows the setting of a UserAgent with a String

```
       void setUserAgent(const String& userAgent);
```

| Output | Parameter | Meaning |
|---|---|---|
| | userAgent | New user agent |
| NONE | | |

#### 2.7.2.6 setAuthorization Log into a web site.

```
    void setAuthorization(const char * user, const char * password);
        void setAuthorization(const char * auth);
```

| Output | Parameter | Meaning |
|---|---|---|
| | user | User name |
| | password | User password |
| NONE | | |

#### 2.7.2.7 setTimeout Set timeout on responses ?

```
        void setTimeout(uint16_t timeout);
```

| Output | Parameter | Meaning |
|---|---|---|
| | timeout | Time out in ms |
| NONE | | |

#### 2.7.2.8 useHTTP10 Legacy HTTP/1.0 support

```
       void useHTTP10(bool usehttp10 = true);
```

| Output | Parameter | Meaning |
|---|---|---|
| | usehttp10 | Enable HTTP/1.0 support DEFAULT = true |
| NONE | | |

### 2.7.3 API - Requests

#### 2.7.3.1 GET Execute a GET HTTP call

```
        int GET();
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | Return HTTP code |

#### 2.7.3.2 POST Send HTTP POST request

```
        int POST(uint8_t * payload, size_t size);
        int POST(String payload);
```

| Output | Parameter | Meaning |
|---|---|---|
| | payload | POST data |
| | size | Data size |
| int | | Return HTTP code |

#### 2.7.3.3 PUT HTTP PUT request

```
int PUT(uint8_t * payload, size_t size);
int PUT(String payload);
```

| Output | Parameter | Meaning |
|---|---|---|
| | payload | PUT data |
| | size | Data size |
| int | | Return HTTP code |

#### 2.7.3.4 sendReq This can send any type of request not just GET/PUT...

```
int sendRequest(const char * type, String payload);
int sendRequest(const char * type, uint8_t * payload = NULL, size_t size
int sendRequest(const char * type, Stream * stream, size_t size = 0);
```

| Output | Parameter | Meaning |
|---|---|---|
| | type | HTTP type |
| | payload | Request data DEFAULT = NULL |
| | size | Size of payload DEFAULT = 0 |
| int | | Return HTTP code |

#### 2.7.3.5 addHeader

```
void addHeader(const String& name, const String& value, bool first = false, bool replace = true);
```

| Output | Parameter | Meaning |
|---|---|---|
| | name | Header name |
| | value | Header value |
| | first | First name DEFAULT = false |
| | replace | Replace entry DEFAULT = true |
| NONE | | |

### 2.7.4 API - Response handling

#### 2.7.4.1 collectHeaders Get all the headers into an array

```
void collectHeaders(const char* headerKeys[], const size_t headerKeysCount);
```

| Output | Parameter | Meaning |
|---|---|---|
| | headerKey | Array of header keys |
| | headerKeysCount | # keys |
| NONE | | |

#### 2.7.4.2 header Get a particular request header/value

```
String header(const char* name);
```

| Output | Parameter | Meaning |
|---|---|---|
| | name | Header name |
| String | | Header item |

#### 2.7.4.3 header Get a header by number

```
String header(size_t i);
```

| Output | Parameter | Meaning |
|---|---|---|
| | i | # in header list |
| String | | Header item |

#### 2.7.4.4 headerName Get header name by number

```
String headerName(size_t i);
```

| Output | Parameter | Meaning |
|---|---|---|
| | i | # in header list |
| String | | Header name |

#### 2.7.4.5 headers Get the number of header items

```
int headers();
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | # Header items |

#### 2.7.4.6 hasHeader

```
bool hasHeader(const char* name);
```

| Output | Parameter | Meaning |
|---|---|---|
| | name | Header name |
| bool | | True if header present |

#### 2.7.4.7 getSize

```
int getSize(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | Length of header |

### 2.7.5 API - MISC

#### 2.7.5.1 getStream

```
WiFiClient& getStream(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| WiFiClient | | WiFi Stream object |

#### 2.7.5.2 getStreamPtr

```
'   WiFiClient* getStreamPtr(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| WiFiClient | | Pointer to WiFi Stream object |

#### 2.7.5.3 writeToStream

```
int writeToStream(Stream* stream);
```

| Output | Parameter | Meaning |
|---|---|---|
| | stream | Pointer to WiFi Stream object |
| int | | # bytes written |

#### 2.7.5.4 getString ?????

```
String getString(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| String | | Data from current stream |

#### 2.7.5.5 errorToString

```
static String errorToString(int error);
```

| Output | Parameter | Meaning |
|---|---|---|
| | error | Error # |
| String | | String error # meaning |

## 2.8 Preferences

### 2.8.1 Use

This library is a wrapper around the Non-Volatile memory in the EPS/32. It uses a KEY/VALUE storage structure. This is NOT the EEPROM.

### 2.8.2 API

#### 2.8.2.1 begin Sets up an area of NV-RAM for use

```
    bool begin(const char * name, bool readOnly=false);
```

| Output | Parameter | Meaning |
|---|---|---|
| | name | Name of NV-RAM area |
| | readOnly | Set area READ ONLY DEFAULT = false |
| bool | | True area setup |

#### 2.8.2.2 end End use of NV-RAM area

```
            void end();
```

| Output | Parameter | Meaning |
|---|---|---|
| NONE | | End use of area |

#### 2.8.2.3 clear Clear the NV-RAM area

```
            bool clear();
```

| Output | Parameter | Meaning |
|---|---|---|
| NONE | | Clear NV-RAM area |

#### 2.8.2.4 remove Remove an stored item

```
            bool remove(const char * key);
```

| Output | Parameter | Meaning |
|---|---|---|
| | key | Key for item to be removed |
| bool | | True if item removed |

#### 2.8.2.5 Particular type stores

```
size_t putChar(const char* key, int8_t value);
size_t putUChar(const char* key, uint8_t value);
size_t putShort(const char* key, int16_t value);
size_t putUShort(const char* key, uint16_t value);
size_t putInt(const char* key, int32_t value);
size_t putUInt(const char* key, uint32_t value);
size_t putLong(const char* key, int32_t value);
size_t putULong(const char* key, uint32_t value);
size_t putLong64(const char* key, int64_t value);
size_t putULong64(const char* key, uint64_t value);
size_t putFloat(const char* key, float_t value);
size_t putDouble(const char* key, double_t value);
size_t putBool(const char* key, bool value);
size_t putString(const char* key, const char* value);
size_t putString(const char* key, String value);
size_t putBytes(const char* key, const void* value, size_t len);
```

#### 2.8.2.6 Particular type reads

```
 int8_t getChar(const char* key, int8_t defaultValue = 0);
uint8_t getUChar(const char* key, uint8_t defaultValue = 0);
int16_t getShort(const char* key, int16_t defaultValue = 0);
uint16_t getUShort(const char* key, uint16_t defaultValue = 0);
int32_t getInt(const char* key, int32_t defaultValue = 0);
uint32_t getUInt(const char* key, uint32_t defaultValue = 0);
int32_t getLong(const char* key, int32_t defaultValue = 0);
uint32_t getULong(const char* key, uint32_t defaultValue = 0);
int64_t getLong64(const char* key, int64_t defaultValue = 0);
uint64_t getULong64(const char* key, uint64_t defaultValue = 0);
float_t getFloat(const char* key, float_t defaultValue = NAN);
double_t getDouble(const char* key, double_t defaultValue = NAN);
bool getBool(const char* key, bool defaultValue = false);
size_t getString(const char* key, char* value, size_t maxLen);
String getString(const char* key, String defaultValue = String());
size_t getBytes(const char* key, void * buf, size_t maxLen);
```

## 2.9 SD

### 2.9.1 Use

This library is the base code for using an SD card. It handles the hardware details. This is an Arduino library.

### 2.9.2 API

Not going to detail the API here, it is well documented in the examples

## 2.10 SD_MMC

### 2.10.1 Use

This library uses the SD library and supports a file system on the SD card.

### 2.10.2 API

Not going to detail the API here, it is well documented in the examples

## 2.11 SimpleBLE

### 2.11.1 Use

This library allows the use of the BlueTooth Low energy protocol.

### 2.11.2 API

#### 2.11.2.1 begin Start BLE advertising.

```
bool begin(String localName=String());
```

| Output | Parameter | Meaning |
|---|---|---|
|  | localName | Advertised name |
| bool |  | True if success |

#### 2.11.2.2 end Stop BLE advertising

```
void end(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| NONE |  |  |

## 2.12 SPI

### 2.12.1 Use

This library is used to run the SPI bus on the ESP32. This is similar to the SPI library on the Arduino but the ESP32 has multiple SPI channels.

### 2.12.2 API - Startup and Setup

#### 2.12.2.1 SPIClass

This instantiates a copy of the SPI class and hooks it to a specific bus.

```
SPIClass(uint8_t spi_bus=HSPI);
```

| Output | Parameter | Meaning |
|---|---|---|
| | spi_bus | SPI bus used by this class DEFAULT = HSPI |
| SPIClass | | SPIClass using specific bus |

#### 2.12.2.2 begin

Setup SPI bus GPIO pins

```
void begin(int8_t sck=-1, int8_t miso=-1, int8_t mosi=-1, int8_t ss=-1);
```

| Output | Parameter | Meaning |
|---|---|---|
| | sck | SPI Clock DEFAULT = -1 (=> Pin 14) |
| | miso | MISO Phase DEFAULT = -1 (=> Pin 12) |
| | mosi | MOSI Phase DEFAULT = -1 (=> Pin 13) |
| | ss | SS Phase DEFAULT = -1 (=> Pin 15) |
| NONE | | |

#### 2.12.2.3 end

Shutdown SPI use

```
void end();
```

| Output | Parameter | Meaning |
|---|---|---|
| NONE | | SPI Off |

#### 2.12.2.4 setHwCs

Set SPI CS line to toggle every byte

```
void setHwCs(bool use);
```

| Output | Parameter | Meaning |
|---|---|---|
| | bool | True set CS line to toggle every byte |
| NONE | | |

#### 2.12.2.5 setBitOrder

Set if MSB or LSB of a byte is sent first

```
void setBitOrder(uint8_t bitOrder);
```

| Output | Parameter | Meaning |
|---|---|---|
| | uint8_t | LSBFIRST or MSBFIRST use defines |
| NONE | | |

#### 2.12.2.6 setDataMode Sets clock polarity and phase on SPI transfer

```
void setDataMode(uint8_t dataMode);
```

| Output | Parameter | Meaning |
|---|---|---|
| | uint8_t | SPI_MODE0,SPI_MODE1,SPI_MODE2 or SPI_MODE3 |
| NONE | | |

#### 2.12.2.7 setFrequency Set the SCK frequency.

```
void setFrequency(uint32_t freq);
```

| Output | Parameter | Meaning |
|---|---|---|
| | freq | Clock frequency in HZ |
| NONE | | |

#### 2.12.2.8 setClockDivider

```
void setClockDivider(uint32_t clockDiv);
```

| Output | Parameter | Meaning |
|---|---|---|
| | clockDiv | Clock divider value |
| NONE | | |

#### 2.12.2.9 beginTransaction Set up SPI for a run. It uses the SPISettings structure with these defaults.

```
class SPISettings
{
public:
    SPISettings() :_clock(1000000),
                  _bitOrder(SPI_MSBFIRST),
                  _dataMode(SPI_MODE0) {}
    SPISettings(uint32_t clock,
                uint8_t bitOrder,
                uint8_t dataMode) :
                _clock(clock),
                _bitOrder(bitOrder),
                _dataMode(dataMode) {}
    uint32_t _clock;
    uint8_t  _bitOrder;
    uint8_t  _dataMode;
};
    void beginTransaction(SPISettings settings);
```

| Output | Parameter | Meaning |
|---|---|---|
| | settings | See SPISettings structure above |
| NONE | | |

#### 2.12.2.10 endTransaction

End use of current settings and shutdown SPI.

```
void endTransaction(void);
```

| Output | Parameter | Meaning |
|---|---|---|
| NONE | | SPI OFF |

### 2.12.3 API - I/O

#### 2.12.3.1 transfer

Send data to/from the SPI device.

```
uint8_t transfer(uint8_t data);
```

| Output | Parameter | Meaning |
|---|---|---|
| | data | Data sent to SPI device |
| uint8_t | | Data returned from SPI device |

```
uint16_t transfer16(uint16_t data);
```

Same but with 16 bit data

```
uint32_t transfer32(uint32_t data);
```

Same but with 32 bit data

#### 2.12.3.2 transferBytes

Transfer a buffer full of data

```
void transferBytes(uint8_t * data, uint8_t * out, uint32_t size);
```

| Output | Parameter | Meaning |
|---|---|---|
| | data | Data sent to SPI device |
| | out | Data returned from SPI device |
| | size | # bytes in transfer |
| NONE | | |

#### 2.12.3.3 transferBits

```
void transferBits(uint32_t data, uint32_t * out, uint8_t bits);
```

| Output | Parameter | Meaning |
|---|---|---|
| | data | Data sent to SPI device |
| | out | Data returned from SPI device |
| | size | # bits in transfer |
| NONE | | |

#### 2.12.3.4 write Just send data to SPI device and ignore return

```
void write(uint8_t data);
```

| Output | Parameter | Meaning |
|---|---|---|
| | data | Data sent to SPI device |
| NONE | | |

```
void write16(uint16_t data);
```

Same with 16 bit data

```
void write32(uint32_t data);
```

Same with 32 bit data

#### 2.12.3.5 writeBytes Send a buffer of data and ignore return.

```
void writeBytes(uint8_t * data, uint32_t size);
```

| Output | Parameter | Meaning |
|---|---|---|
| | data | Data sent to SPI device |
| | size | # bits in transfer |
| NONE | | |

### 2.12.4 API - Display Drivers

These are used to drive LCD displays. Need an example to describe use.

```
void writePixels(const void * data, uint32_t size);//ili9341 compatible
void writePattern(uint8_t * data, uint8_t size, uint32_t repeat);
```

## 2.13 SPIFFS

### 2.13.1 Use

This library uses the SPI library and supports a file system via SPI.

### 2.13.2 API

Not going to detail the API here, it is well documented in the examples

## 2.14 Ticker

### 2.14.1 Use

This is a timer based call back system to run routines at specified intervals. The call back functions are of this form.

```
typedef void (*callback_t)(void);
typedef void (*callback_with_arg_t)(void*);
template<typename TArg>
```

### 2.14.2 API

**2.14.2.1 attach attach_ms** These calls set up the periodic function. The various calls allow different parameters for the time and use of the callling function with and without a parameter

```
void attach(float seconds, callback_t callback)
```

| Output | Parameter | Meaning |
|---|---|---|
| | seconds | Floating point seconds value |
| | callback | Function called with no parameter |
| NONE | | |

```
void attach_ms(uint32_t milliseconds, callback_t callback)
```

| Output | Parameter | Meaning |
|---|---|---|
| | milliseconds | uint32_t millisecond period |
| | callback | Function called with no parameter |
| NONE | | |

```
void attach(float seconds, void (*callback)(TArg), TArg arg)
```

| Output | Parameter | Meaning |
|---|---|---|
| | seconds | Floating point seconds value |
| | callback | Function called |
| | arg | Function parameter |
| NONE | | |

#### 2.14.2.2 once once_ms

These calls set up to call the function only once after a delay

```
void once(float seconds, callback_t callback)
```

| Output | Parameter | Meaning |
|---|---|---|
| | seconds | Floating point seconds value |
| | callback | Function called with no parameter |
| NONE | | |

```
void once_ms(uint32_t milliseconds, callback_t callback)
```

| Output | Parameter | Meaning |
|---|---|---|
| | milliseconds | uint32_t millisecond period |
| | callback | Function called with no parameter |
| NONE | | |

```
void once(float seconds, void (*callback)(TArg), TArg arg)
```

| Output | Parameter | Meaning |
|---|---|---|
| | seconds | Floating point seconds value |
| | callback | Function called |
| | arg | Function parameter |
| NONE | | |

```
void once_ms(uint32_t milliseconds, void (*callback)(TArg), TArg arg)
```

| Output | Parameter | Meaning |
|---|---|---|
| | milliseconds | uint32_t millisecond value |
| | callback | Function called |
| | arg | Function parameter |
| NONE | | |

#### 2.14.2.3 detach

Detach and stop the current periodic function

```
void detach();
```

| Output | Parameter | Meaning |
|---|---|---|
| NONE | | |

#### 2.14.2.4 active

```
    bool active();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | TRUE if periodic function is running |

## 2.15 Update

### 2.15.1 Use

This class is used to update programs on the ESP32. It can check if space is available and if there is backup boot code. It is similar to the ArduioOTA

### 2.15.2 API

Not going to detail the API here as the example programs adequately show how to use this library.

## 2.16 WiFi

### 2.16.1 Use

This library is the main interface to the WiFi and TCP/IP connections of the ESP32. The library has many parts and the API documentation will got into each.API

### 2.16.2 API - ETH

This library is an interface to the underlying light weight IP stack (lwIP). See the example WiFi-ClientStaticIP.ino for use information.

#### 2.16.2.1 begin Setup lwIP parameters

```
            bool begin(uint8_t phy_addr=ETH_PHY_ADDR,
                 int power=ETH_PHY_POWER,
                 int mdc=ETH_PHY_MDC,
                 int mdio=ETH_PHY_MDIO,
                 eth_phy_type_t type=ETH_PHY_TYPE,
                 eth_clock_mode_t clk_mode=ETH_CLK_MODE);
```

| Output | Parameter | Meaning |
|---|---|---|
| | phy_addr | IP Address DEFAULT = ETH_PHY_ADDR |
| | power | Interface Power DEFAULT = ETH_PHY_POWER |
| | mdc | ??? DEFAULT = ETH_PHY_MDC |
| | mdio | ??? DEFAULT = ETH_PHY_MDIO |
| | type | ??? DEFAULT = ETH_PHY_TYPE |
| | clk_mode | ??? DEFAULT = ETH_CLK_MODE |
| bool | | TRUE if successful |

#### 2.16.2.2 config Configure IP path information.

```
bool config(IPAddress local_ip,
            IPAddress gateway,
            IPAddress subnet,
            IPAddress dns1 = (uint32_t)0x00000000,
            IPAddress dns2 = (uint32_t)0x00000000);
```

| Output | Parameter | Meaning |
|---|---|---|
| | local_ip | IP Address |
| | gateway | Router address |
| | subnet | Subnet mask |
| | dns1 | Assigned DNS DEFAULT = 00.00.00.00 |
| | dns2 | Assigned DNS DEFAULT = 00.00.00.00 |
| bool | | TRUE if successful |

#### 2.16.2.3 getHostname Get the current host name

```
const char * getHostname();
```

| Output | Parameter | Meaning |
|---|---|---|
| char * | | Current host name |

#### 2.16.2.4 setHostname Set current host name

```
bool setHostname(const char * hostname);
```

| Output | Parameter | Meaning |
|---|---|---|
| | hostname | New host name |
| bool | | True if successful |

#### 2.16.2.5 fullDuplex Set for full duplex communication

```
bool fullDuplex();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | True if successful |

#### 2.16.2.6 linkUp Check for link connected

```
bool linkUp();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | True if link running |

#### 2.16.2.7 linkSpeed

```
uint8_t linkSpeed();
```

| Output | Parameter | Meaning |
|---|---|---|
| uint8_t | | Current link speed |

#### 2.16.2.8 enableIpV6 Turn on IPv6 processing

```
bool enableIpV6();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | TRUE if enabled |

#### 2.16.2.9 localIPv6 Current IPv6 address (IPv6 enabled)

```
IPv6Address localIPv6();
```

| Output | Parameter | Meaning |
|---|---|---|
| IPv6Address | | Current IPv6 address |

#### 2.16.2.10 localIP

```
IPAddress localIP();
```

| Output | Parameter | Meaning |
|---|---|---|
| IPAddress | | Current IPv4 address |

#### 2.16.2.11 subnetMask

```
IPAddress subnetMask();
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| IPAddress | | Current subnet mask |

#### 2.16.2.12 gatewayIP

```
IPAddress gatewayIP();
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| IPAddress | | Current Gateway IP address |

#### 2.16.2.13 dnsIP

```
IPAddress dnsIP(uint8_t dns_no = 0);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| | dns_no | Which DNS (0/1) DEFAULT = 0 |
| IPAddress | | DNS IP address |

#### 2.16.2.14 macAddress

```
uint8_t * macAddress(uint8_t* mac);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| | mac | New MAC address (6 uint8_t values) |
| uint8_t * | | New MAC address |

#### 2.16.2.15 macAddress

```
String macAddress();
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| String | | Current MAC address |

### 2.16.3 API - WIFISTA

**2.16.3.1 Use** This is analogous to the ETH class above but for for the WiFi connection. The examples for these classes are excellenr ways of seeing how the class functions are used.

These calls setup a connection to an Access Point.

```
        wl_status_t begin(const char* ssid,
                          const char *passphrase = NULL,
                          int32_t channel = 0,
                          const uint8_t* bssid = NULL,
                          bool connect = true);
        wl_status_t begin(char* ssid,
                          char *passphrase = NULL,
                          int32_t channel = 0,
                          const uint8_t* bssid = NULL,
                          bool connect = true);
        wl_status_t begin();
```

| Output | Parameter | Meaning |
|---|---|---|
| | ssid | AP name |
| | passphrase | If encrypted, password DEFAULT = NULL |
| | channel | WiFi Channel to use DEFAULT = 0 (ANY) |
| | bssid | WiFi BSSID for AP DEFAULT = 0 (NONE) |
| | connect | TRUE if connection wanted |
| wl_status | | See WiFiTypes.h for enum |

**2.16.3.2 config** Set TCP/IP parameters for AP connectioin

```
        bool config(IPAddress local_ip,
                    IPAddress gateway,
                    IPAddress subnet,
                    IPAddress dns1 = (uint32_t)0x00000000,
                    IPAddress dns2 = (uint32_t)0x00000000);
```

| Output | Parameter | Meaning |
|---|---|---|
| | local_ip | Local IP of ESP32 |
| | gateway | AP assigned gateway |
| | subnet | AP assigned subnet mask |
| | dns1 | AP assigned DNS DEFAULT = 0x00000000 |
| | dns2 | AP assigned DNS DEFAULT = 0x00000000 |
| bool | | TRUE if set |

#### 2.16.3.3 reconnect

```
bool reconnect();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | If reconnection succesful |

#### 2.16.3.4 disconnect

```
bool disconnect(bool wifioff = false);
```

| Output | Parameter | Meaning |
|---|---|---|
| | wifioff | Turn off WiFi DEFAULT = false |
| bool | | True if connection dropped |

#### 2.16.3.5 isConnected

```
bool isConnected();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | Connection status |

#### 2.16.3.6 setAutoConnect

```
bool setAutoConnect(bool autoConnect);
```

| Output | Parameter | Meaning |
|---|---|---|
| | autoConnect | True to enable |
| bool | | Auto Connect ON |

#### 2.16.3.7 getAutoConnect

```
bool getAutoConnect();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | AutoConnect status |

#### 2.16.3.8 setAutoReconnect

```
bool setAutoReconnect(bool autoReconnect);
```

| Output | Parameter | Meaning |
|---|---|---|
| | autoReonnect | True to enable |
| bool | | Auto Reonnect ON |

#### 2.16.3.9 getAutoReconnect

```
bool getAutoReconnect();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | AutoReonnect status |

#### 2.16.3.10 waitForConnectResult

```
uint8_t waitForConnectResult();
```

| Output | Parameter | Meaning |
|---|---|---|
| uint8_t | | Wait for connection to return a result |

#### 2.16.3.11 localIP

```
IPAddress localIP();
```

| Output | Parameter | Meaning |
|---|---|---|
| IPAddress | | Current IPv4 address |

#### 2.16.3.12 subnetMask

```
IPAddress subnetMask();
```

| Output | Parameter | Meaning |
|---|---|---|
| IPAddress | | Current subnet mask |

#### 2.16.3.13 gatewayIP

```
IPAddress gatewayIP();
```

| Output | Parameter | Meaning |
|---|---|---|
| IPAddress | | Current Gateway IP address |

#### 2.16.3.14 dnsIP

```
IPAddress dnsIP(uint8_t dns_no = 0);
```

| Output | Parameter | Meaning |
|---|---|---|
| | dns_no | Which DNS (0/1) DEFAULT = 0 |
| IPAddress | | DNS IP address |

#### 2.16.3.15 macAddress

```
uint8_t * macAddress(uint8_t* mac);
```

| Output | Parameter | Meaning |
|---|---|---|
| | mac | New MAC address (6 uint8_t values) |
| uint8_t * | | New MAC address |

#### 2.16.3.16 macAddress

```
String macAddress();
```

| Output | Parameter | Meaning |
|---|---|---|
| String | | Current MAC address |

#### 2.16.3.17 enableIpV6 Turn on IPv6 processing

```
bool enableIpV6();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | TRUE if enabled |

#### 2.16.3.18 localIPv6 Current IPv6 address (IPv6 enabled)

```
IPv6Address localIPv6();
```

| Output | Parameter | Meaning |
|---|---|---|
| IPv6Address | | Current IPv6 address |

#### 2.16.3.19 getHostname Get the current host name

```
const char * getHostname();
```

| Output | Parameter | Meaning |
|---|---|---|
| char * | | Current host name |

#### 2.16.3.20 setHostname Set current host name

```
bool setHostname(const char * hostname);
```

| Output | Parameter | Meaning |
|---|---|---|
| | hostname | New host name |
| bool | | True if successful |

### 2.16.3.21 status

```
static wl_status_t status();
```

| Output | Parameter | Meaning |
|---|---|---|
| wl_status | | Link status ENUM |

### 2.16.3.22 SSID

```
String SSID() const;
```

| Output | Parameter | Meaning |
|---|---|---|
| String | | Connected AP's SSID |

### 2.16.3.23 psk

```
String psk() const;
```

| Output | Parameter | Meaning |
|---|---|---|
| String | | Connected AP's pre-shared key |

### 2.16.3.24 BSSID

```
uint8_t * BSSID();
```

| Output | Parameter | Meaning |
|---|---|---|
| uint8_t | | BSSID as a list of uint8_t's |

### 2.16.3.25 BSSIDstr

```
String BSSIDstr();
```

| Output | Parameter | Meaning |
|---|---|---|
| String | | BSSID as a String |

### 2.16.3.26 RSSI

```
int8_t RSSI();
```

| Output | Parameter | Meaning |
|---|---|---|
| uint8_t | | Received Signal Strength |

### 2.16.4 API - Client

**2.16.4.1 Use** This class is used after the connection is up to open sockets,to read and write over the connection. See the example WiFiTelnetToSerial.

**2.16.4.2 connect** Connect to an IP address (after being hooked up to a WiFi access point)

```
int connect(IPAddress ip, uint16_t port);
```

| Output | Parameter | Meaning |
|---|---|---|
| | ip | IP Address of target |
| | port | Port # to connect |
| int | | 1 if connected, < 0 if error |

**2.16.4.3 connect** Connect to a named host.

```
int connect(const char *host, uint16_t port);
```

| Output | Parameter | Meaning |
|---|---|---|
| | host | Named host |
| | port | Port # to connect |
| int | | 1 if connected, < 0 if error |

**2.16.4.4 write** Write data to connected destination.

```
size_t write(uint8_t data);
size_t write(const uint8_t *buf, size_t size);
size_t write_P(PGM_P buf, size_t size);
```

**2.16.4.5 available** Is the connection still up ?

```
int available();
```

**2.16.4.6 read** Read data from the connected device.

```
int read();
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | Read 1 byte or -1 if EOF |

```
int read(uint8_t *buf, size_t size);
```

| Output | Parameter | Meaning |
|---|---|---|
| | buf | Data buffer |
| | size | # Bytes to read |
| int | | # Bytes read |

#### 2.16.4.7 **peek** Get next byte from connected device but don't read it

```
int peek();
```

| Output | Parameter | Meaning |
|---|---|---|
| int | | Read 1 byte or -1 if EOF |

#### 2.16.4.8 **flush** Move all pending write data to connected device

```
void flush();
```

#### 2.16.4.9 **stop** Drop connected devicd

```
void stop();
```

#### 2.16.4.10 **connected** Is the external device still connected.

```
bool connected();
```

#### 2.16.4.11 **fd** Return socket # for connected device

```
int fd() const;
```

#### 2.16.4.12 **setSocketOption** Set options for socket (see BSD socket documentation)

```
int setSocketOption(int option, char* value, size_t len);
```

| Output | Parameter | Meaning |
|---|---|---|
| | option | See TCP/IP documentation |
| | value | New value for option |
| | len | Size of value |
| int | | 0 if OK, <> 0 if not |

#### 2.16.4.13 **setOption** Set an option with a simple value

```
int setOption(int option, int *value);
```

| Output | Parameter | Meaning |
|---|---|---|
| | option | See TCP/IP documentation |
| | value | New value for option |
| int | | 0 if OK, <> 0 if not |

#### 2.16.4.14 **getOption** Return value of an option

```
int getOption(int option, int *value);
```

| Output | Parameter | Meaning |
|---|---|---|
| | option | See TCP/IP documentation |
| | value | Return value for option |
| int | | 0 if OK, <> 0 if not |

#### 2.16.4.15 **setTimeout** Set inactivity timeout

```
int setTimeout(uint32_t seconds);
```

| Output | Parameter | Meaning |
|---|---|---|
| | seconds | New timeout value |
| int | | 0 if OK, <> 0 if not |

#### 2.16.4.16 **setNoDelay** Set TCP/IP NoDelay option

```
int setNoDelay(bool nodelay);
```

| Output | Parameter | Meaning |
|---|---|---|
| | nodelay | True to turn on option |
| int | | 0 if OK, <> 0 if not |

#### 2.16.4.17 **getNoDelay** Get TCP/IP NoDelay option

```
bool getNoDelay();
```

| Output | Parameter | Meaning |
|---|---|---|
| bool | | True if NoDelay if on |

#### 2.16.4.18 remoteIP,remotePort,localIP,localPort

These functions read out the connection parameters and optionally use a supplied socket #, or use the implied current socket

```
IPAddress remoteIP() const;
IPAddress remoteIP(int fd) const;
uint16_t remotePort() const;
uint16_t remotePort(int fd) const;
IPAddress localIP() const;
IPAddress localIP(int fd) const;
uint16_t localPort() const;
uint16_t localPort(int fd) const;
```

### 2.16.5 WifiScan,WiFiMulti,WiFiServer,WiFiUdp

These secrtions are quite adequately convered by their respective examples so will not be detailed here

## 2.17 WiFiClientSecure

The class is much the same as WiFiClient but allows encrypted communication with certificates. The methods are much the same save the certificate handling. As such the class will not be detailed here, the examples show how to use the unique methods.

## 2.18 Wire

### 2.18.1 Use

This library supports I2C on the ESP32 and matches the Wire library on a standard Arduino.

### 2.18.2 API

#### 2.18.2.1 begin

Set up I2C pins and clock.

```
void begin(int sda=-1, int scl=-1, uint32_t frequency=100000);
```

| Output | Parameter | Meaning |
|---|---|---|
| | sda | Set SDA pin (-1 = SDA pin) |
| | scl | Set SCL pin (-1 = SCL pin) |
| | frequency | Set clock freq DEFAULT = 100,000 Hz |
| NONE | | |

#### 2.18.2.2 **setClock** Set clock frequency

```
void setClock(uint32_t);
```

| Output | Parameter | Meaning |
|---|---|---|
| | frequency | Set clock freq |
| NONE | | |

#### 2.18.2.3 **beginTransmission** Sent first byte of I2C transaction (address)

```
void beginTransmission(uint8_t);
void beginTransmission(int);
```

Note only bottom 8 bits used in either case

| Output | Parameter | Meaning |
|---|---|---|
| | address | Send address |
| NONE | | |

#### 2.18.2.4 **endTransmission** End I2C transaction

```
uint8_t endTransmission(void);
uint8_t endTransmission(uint8_t);
```

| Output | Parameter | Meaning |
|---|---|---|
| | stop | Send STOP DEFAULT = True |
| NONE | | |

#### 2.18.2.5 **requestFrom** Master sends request to slave for a number of bytes.

```
size_t requestFrom(uint8_t address, size_t size, bool sendStop);
    uint8_t requestFrom(uint8_t, uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t, uint8_t);
    uint8_t requestFrom(int, int);
    uint8_t requestFrom(int, int, int);
```

| Output | Parameter | Meaning |
|---|---|---|
| | address | Slave address |
| | size | # Bytes requested |
| | sendStop | Send STOP after request DEFAULT = True |
| uint8_t | | # Bytes returned |

#### 2.18.2.6 **write** Write to addressed slave device

```
size_t write(uint8_t);
size_t write(const uint8_t *, size_t);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| | buf | Data to send |
| | size | # Bytes sent |
| uint8_t | | # Bytes sent |

#### 2.18.2.7 **available** Check for data available from slave

```
int available(void);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| int | | # Bytes available |

### 2.18.3 read

```
int read(void);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| int | | Byte from slave |

#### 2.18.3.1 **peek** Look at next byte from slave

```
int peek(void);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| int | | Byte from slave |

#### 2.18.3.2 **flush** Flush write buffers

```
void flush(void);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| NONE | | |

#### 2.18.3.3 **reset** Resets I2C bus

```
void reset(void);
```

| **Output** | **Parameter** | **Meaning** |
|---|---|---|
| NONE | | |

## References

[1] Lady Ada. *Adafruit Feather HUZZA ESP8266*. Adafruit Inc., www.adafruit.com, sept 7 2016 edition, September 2016.

[2] Adafruit. Adafruit huzzah32 – esp32 feather board.

[3] Arduino. https://www.arduino.cc, November 2015.

[4] Wikipedia. Multicast dns.