

WifiOnOff

Generated by Doxygen 1.8.16

1 WIFIOff	1
1.0.1 *	1
1.0.2 *	1
1.0.3 *	1
1.0.4 *	2
1.0.5 *	2
2 File Index	3
2.1 File List	3
3 File Documentation	5
3.1 /home/travis/build/peastone/WIFIOff/README.md File Reference	5
3.2 /home/travis/build/peastone/WIFIOff/src/WIFIOff.ino File Reference	5
3.2.1 *	5
3.2.2 *	7
3.2.3 *	9
3.2.4 Detailed Description	11
3.2.5 Macro Definition Documentation	11
3.2.5.1 CTR_MAX_TRIES_WIFI_CONNECTION	11
3.2.5.2 DEV_OTA_UPDATES	11
3.2.5.3 EEPROM_address_CRC	11
3.2.5.4 EEPROM_address_MQTT_server	12
3.2.5.5 EEPROM_address_MQTT_server_configured	12
3.2.5.6 EEPROM_address_OTA_password	12
3.2.5.7 EEPROM_address_start	12
3.2.5.8 EEPROM_enabled	12
3.2.5.9 EEPROM_enabled_size	12
3.2.5.10 EEPROM_init_value	12
3.2.5.11 EEPROM_length	12
3.2.5.12 EEPROM_size_CRC	13
3.2.5.13 EEPROM_size_MQTT_server	13
3.2.5.14 EEPROM_size_OTA_password	13
3.2.5.15 EEPROM_version_number	13
3.2.5.16 EEPROM_version_number_address	13
3.2.5.17 HTTP_PORT	14
3.2.5.18 MQTT_PORT	14
3.2.5.19 NUMERIC_RESPONSE	14
3.2.5.20 OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED	14
3.2.5.21 SERIAL_PRINTING	14
3.2.5.22 TIME_EEPROM_WARNING	14
3.2.5.23 TIME_HALF_A_SECOND	15
3.2.5.24 TIME_QUARTER_SECOND	15
3.2.5.25 TRIGGER_TIME_FACTORY_RESET	15

3.2.6 Function Documentation	15
3.2.6.1 calculate_crc()	15
3.2.6.2 callbackConfigMode()	16
3.2.6.3 callbackConfigSuccessful()	16
3.2.6.4 changeWiFiCredentials()	17
3.2.6.5 checkCRC()	17
3.2.6.6 checkEEPROMVersionNumber()	18
3.2.6.7 checkMQTTConnected()	19
3.2.6.8 checkWiFiConnected()	19
3.2.6.9 configureMQTT()	20
3.2.6.10 configureOutputs()	20
3.2.6.11 configureWebServer()	21
3.2.6.12 connectRelay()	22
3.2.6.13 disconnectRelay()	22
3.2.6.14 feedbackQuickBlink()	23
3.2.6.15 getClientID()	23
3.2.6.16 getDefaultOtaPassword()	24
3.2.6.17 getFactoryResetRequested()	24
3.2.6.18 getMQTTOutgoingTopic()	24
3.2.6.19 getMQTTServer()	25
3.2.6.20 getStateLEDOn()	25
3.2.6.21 getStateMQTTConfigured()	26
3.2.6.22 getStateRelayConnected()	26
3.2.6.23 getUserActionFeedbackRequest()	27
3.2.6.24 getWiFiCredentialChangeRequested()	27
3.2.6.25 initEEPROM()	27
3.2.6.26 isNotWhitelisted()	28
3.2.6.27 loop()	29
3.2.6.28 mqttConnect()	29
3.2.6.29 mqttControlRelay()	30
3.2.6.30 mqttPublish()	31
3.2.6.31 mqttServerValid()	32
3.2.6.32 paramOtaPassword()	33
3.2.6.33 performFactoryReset()	33
3.2.6.34 pressHandler()	34
3.2.6.35 reboot()	35
3.2.6.36 renderFactoryReset()	35
3.2.6.37 renderFooter()	35
3.2.6.38 renderHeader()	35
3.2.6.39 renderInfo()	36
3.2.6.40 renderMQTTSettings()	36
3.2.6.41 renderOtaPasswordChangeForm()	37

3.2.6.42 renderRelay()	37
3.2.6.43 renderWiFiCredentialChange()	38
3.2.6.44 restoreMQTTConfigurationFromEEPROM()	38
3.2.6.45 restoreOtaPasswordFromEEPROM()	39
3.2.6.46 retrieveCRC()	39
3.2.6.47 saveMQTTConfigurationToEEPROM()	40
3.2.6.48 saveOtaPasswordToEEPROM()	40
3.2.6.49 setFactoryResetRequested()	41
3.2.6.50 setMQTTIncomingTopic()	41
3.2.6.51 setMQTTOutgoingTopic()	41
3.2.6.52 setMQTTServer()	42
3.2.6.53 setStateMQTTConfigured()	42
3.2.6.54 setup()	43
3.2.6.55 setupEEPROM()	44
3.2.6.56 setupMDNS()	45
3.2.6.57 setUserActionFeedbackRequest()	45
3.2.6.58 setWiFiCredentialChangeRequested()	46
3.2.6.59 storeCRC()	46
3.2.6.60 storeEEPROMVersionNumber()	47
3.2.6.61 switchOffLED()	47
3.2.6.62 switchOnLED()	48
3.2.6.63 toggleLED()	48
3.2.6.64 toggleRelay()	49
3.2.6.65 unsetUserActionFeedbackRequest()	49
3.2.7 Variable Documentation	50
3.2.7.1 arduinoOtaPassword	50
3.2.7.2 buttonLastPressed	50
3.2.7.3 buttonPin	50
3.2.7.4 credentialPassword	50
3.2.7.5 credentialSsid	50
3.2.7.6 factoryResetRequested	50
3.2.7.7 mqttClient	50
3.2.7.8 mqttIncomingTopic	51
3.2.7.9 mqttOutgoingTopic	51
3.2.7.10 mqttServer	51
3.2.7.11 pinLED	51
3.2.7.12 pinRelay	51
3.2.7.13 REPOSITORY_URL_STRING	51
3.2.7.14 selectionState	51
3.2.7.15 stateLEDOOn	52
3.2.7.16 stateMQTTConfigured	52
3.2.7.17 stateRelayConnected	52

3.2.7.18 ticker	52
3.2.7.19 timeSincebuttonPressed	52
3.2.7.20 userActionFeedbackRequest	52
3.2.7.21 VERSION	52
3.2.7.22 webserver	53
3.2.7.23 wifiClient	53
3.2.7.24 wifiCredentialChangeRequested	53
3.2.7.25 wifiManager	53
3.3 /home/travis/build/peastone/WIFIONOff/src/wifionoff_version.h File Reference	53
3.3.1 *	53
3.3.2 Macro Definition Documentation	54
3.3.2.1 VERSION_BY_GIT_DESCRIBE	54
Index	55

Chapter 1

WIFIONOff

1.0.1 *

Motivation

This repository was inspired by the article ["Bastelfreundlich"](#) by the German computer magazine "c't". The article there is pretty much introductory. If you speak German and you have difficulties to get started, you can have a look there.

1.0.2 *

Caution: Dragons ahead

The relay in the Sonoff S20 EU can only handle 10 A. Do NOT plug in devices with draw more current. The maximum amount of current is also noted on the backside of the Sonoff S20 EU. It may differ in other variants sold, so have a look.

Never program the Sonoff S20, if it is connected to the mains. In Germany, that's 230 volts and you don't want to get an electric shock, which might even kill you.

That put ahead, go forward and be cautious!

1.0.3 *

Flash it

1. Install Arduino IDE ([instructions](#)).
2. Install Arduino core for ESP8266 ([instructions](#)).
3. Install the library ([instructions](#)) for [MQTT from Joël Gähwiler](#).
4. Get yourself a FTDI-232-USB-TTL converter and some jumper wires.
5. Connect the FTDI to the Sonoff S20 (Pins from top to button: GND, TX, RX, 3.3V; Top is located right underneath the socket. Pin connection may vary for other variants or over time. You flash at your own risk.).
6. Build the software and flash it.

The whole procedure may also be done with [PlatformIO](#). It should be easy to find out the analogous steps for yourself. A platformio.ini is provided but must be adapted.

["Sonoff Pinout"](#) by [ct-Open-Source](#) licensed under CC Attribution Share Alike 4.0 International.

1.0.4 *

Use it

1. Close the Sonoff S20. BE cautious! The device MUST be absolutely closed. You are working at your own risk here. If the device is open, you risk getting an electric shock or worse. If in doubt, ask an expert!
2. Plug the Sonoff S20 into the mains.
3. The LED will light green. An access point with the name "WIFIONOff" will be started. Connect your mobile device and follow the captive portal. There you can enter your WiFi credentials or use WPS.
4. Find the device with an mDNS mobile phone app.
5. Open the web interface by entering the mDNS name into your browser.
6. Configure MQTT, if wanted.
7. Enjoy the web interface, MQTT and the physical button. Toggle the relay.

1.0.5 *

Documentation

Please read the manual([HTML](#), [PDF](#)). It is pretty detailed and should answer most of your questions. Only the latest version is provided. All versions can be generated with [Doxygen](#).

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

/home/travis/build/peastone/WIFLOnOff/src/ WIFLOnOff.ino	5
/home/travis/build/peastone/WIFLOnOff/src/ wifionoff_version.h	53

Chapter 3

File Documentation

3.1 /home/travis/build/peastone/WIFIONOff/README.md File Reference

3.2 /home/travis/build/peastone/WIFIONOff/src/WIFIONOff.ino File Reference

```
#include <ESP8266WebServer.h>
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <EEPROM.h>
#include <Ticker.h>
#include <MQTTClient.h>
#include "wifionoff_version.h"
#include "WiFiManager/WiFiManager.h"
#include <ArduinoOTA.h>
```

Include dependency graph for WIFIONOff.ino:



3.2.1 *

Macros

- **#define SERIAL_PRINTING**
Define SERIAL_PRINTING if you want to enable serial communication.
- **#define NUMERIC_RESPONSE**
Define NUMERIC_RESPONSE if you want the `mqttClient` to publish "1" or "0" instead of "on" or "off".
- **#define OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED**
The state of the relay is visible through the blue LED (connected == LED shines). Define OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED, if you want the state of the relay being visible also on the green LED (connected == LED shines).
- **#define DEV_OTA_UPDATES**

- Define `DEV_OTA_UPDATES` if you want to perform OTA updates for development with Arduino IDE.
- `#define TRIGGER_TIME_FACTORY_RESET 15000`
time in ms to wait until factory reset is triggered when pressing the button (see `pressHandler()`)
 - `#define HTTP_PORT 80`
standard port used for the HTTP protocol
 - `#define MQTT_PORT 1883`
standard port used for the MQTT protocol
 - `#define TIME_HALF_A_SECOND 500`
definition of the timespan of half a second
 - `#define TIME_QUARTER_SECOND 250`
definition of the timespan of a quarter of a second
 - `#define TIME_EEPROM_WARNING 7000`
definition of the time the LED stays on in case of an EEPROM warning (CRC failure, initialization, EEPROM incompatible)
 - `#define CTR_MAX_TRIES_WIFI_CONNECTION 60`
maximum amount of tries to connect to WIFI
 - `#define EEPROM_length 4096`
EEPROM_length defines the maximum amount of bytes to be stored in the EEPROM. At the time of writing, the maximum amount of bytes to be stored is 4096.
 - `#define EEPROM_version_number 1`
A CRC is calculated for the EEPROM values to ensure that the values are untempered. The address of the stored CRC is at the beginning of the EEPROM.
 - `#define EEPROM_version_number_address 0`
This version number is stored in EEPROM. It must not be longer than one byte. The version number defined here and that one stored in EEPROM are compared against each other. In case the two numbers are not equal, the EEPROM is re-initialized. This number must change, if you change the EEPROM layout, basically if you change addresses.
 - `#define EEPROM_address_CRC 1`
The EEPROM version number is stored at this address.
 - `#define EEPROM_size_CRC (sizeof(unsigned long))`
A CRC is calculated for the EEPROM values to ensure that the values are untempered. The size of the stored CRC is defined by this macro.
 - `#define EEPROM_address_start (EEPROM_address_CRC + EEPROM_size_CRC)`
A CRC is calculated for the EEPROM values to ensure that the values are untempered. The CRC takes some space to be stored in the EEPROM. The start address takes this into account.
 - `#define EEPROM_enabled 0xEB`
If this magic number is set, a defined functionality is enabled.
 - `#define EEPROM_enabled_size 1`
The amount of storage needed by `EEPROM_enabled` in EEPROM.
 - `#define EEPROM_init_value 0x00`
If an init value is set, a defined functionality is disabled. The init value is also used to initialize the EEPROM in `setUpEEPROM()`.
 - `#define EEPROM_address_MQTT_server_configured EEPROM_address_start`
This flag is used to check whether `mqttServer` has already been initialized with the value of the MQTT-Server (DNS name or IP)
 - `#define EEPROM_address_MQTT_server (EEPROM_address_MQTT_server_configured + EEPROM_enabled_size)`
The variable `mqttServer` which contains the value of the MQTT-Server (DNS name or IP) is restored after startup from EEPROM, if MQTT is configured (`EEPROM_address_MQTT_server_configured`). The stored bytes can be found at this address.
 - `#define EEPROM_size_MQTT_server 256`
The maximum amount of bytes used to store the MQTT server address. This address can be either an IP or a DNS name. DNS names are restricted to 255 octets. As a string is zero-terminated, one byte more is used. For more information, look at RFC 1035.
 - `#define EEPROM_address_OTA_password (EEPROM_address_MQTT_server + EEPROM_size_MQTT_server)`
The OTA password is stored here.
 - `#define EEPROM_size_OTA_password 256`
Define length for OTA password.

3.2.2 *

Functions

- unsigned long [calculate_crc](#) ()
This function calculates a CRC checksum for the EEPROM. It is taken from: <https://www.arduino.cc/en/Tutorial/EEPRoMCrc>.
- void [storeCRC](#) ()
This function calculates the CRC value of the values in the EEPROM with the help of function [calculate_crc\(\)](#). The resulting CRC checksum is written to [EEPROM_address_CRC](#). The caller must still call `EEPROM.commit()` afterwards to really trigger a write to EEPROM.
- unsigned long [retrieveCRC](#) ()
This function reads the CRC value stored at [EEPROM_address_CRC](#).
- bool [checkCRC](#) ()
This function calculates the CRC value for the bytes stored in the EEPROM with the help of function [calculate_crc\(\)](#) and compares the result to the CRC which was read from EEPROM with the help of function [retrieveCRC\(\)](#).
- void [storeEEPROMVersionNumber](#) ()
This function stores the [EEPROM_version_number](#) in EEPROM.
- bool [checkEEPROMVersionNumber](#) ()
This function checks the [EEPROM_version_number](#) stored in EEPROM against the version number required by the latest EEPROM layout.
- void [initEEPROM](#) ()
This function writes default data ([EEPROM_init_value](#)) to EEPROM and stores CRC and the [EEPROM_version_number](#).
- void [setupEEPROM](#) ()
This function is setting up the EEPROM. If the CRC check fails, the EEPROM will be overwritten with init values. The CRC check makes only sense at initialization phase. Afterwards, the data is buffered by EEPROM lib in RAM.
- void [switchOffLED](#) ()
This function is used to switch the LED off. It keeps track of the internal state [stateLEDOn](#).
- void [switchOnLED](#) ()
This function is used to switch the LED on. It keeps track of the internal state [stateLEDOn](#).
- void [toggleLED](#) ()
This function is used to toggle the LED. Indirectly, it keeps track of the internal state [stateLEDOn](#).
- bool [getStateLEDOn](#) ()
This function is used to get the state of the LED. It returns [stateLEDOn](#).
- void [feedbackQuickBlink](#) ()
This function is called to give the user feedback about the menu the user selected. It will blink fast for a short period of time. The user has the choice to release and select the menu or to wait for the next menu. By counting the [feedbackQuickBlink\(\)](#)-events, the user can determine which menu is selected, when the button is released now.
- bool [getUserActionFeedbackRequest](#) ()
Getter function for [userActionFeedbackRequest](#).
- void [setUserActionFeedbackRequest](#) ()
Setter function for [userActionFeedbackRequest](#). The user should be notified that a menu can be selected.
- void [unsetUserActionFeedbackRequest](#) ()
Setter function for [userActionFeedbackRequest](#). The user has been notified that a menu can be selected.
- String [getClientID](#) ()
This function returns the client name used for MQTT, see [mqttConnect\(\)](#).
- String [getDefaultOtaPassword](#) ()
- WiFiManagerParameter [paramOtaPassword](#) ("ota", "Arduino OTA Password", [getDefaultOtaPassword\(\)](#).c_str(), [EEPROM_size_OTA_password](#)-1, "")
Password to protect OTA updates. This password should be changed in [WiFiManager](#).
- void [setupMDNS](#) ()
This function sets up MDNS.
- void [changeWiFiCredentials](#) ()

- bool [checkWiFiConnected](#) ()
This function checks whether WIFI is connected.
- bool [getWiFiCredentialChangeRequested](#) ()
Getter function for [wifiCredentialChangeRequested](#).
- void [setWiFiCredentialChangeRequested](#) ()
Setter function to set [wifiCredentialChangeRequested](#).
- void [disconnectRelay](#) ()
This function is used to disconnect the relay from the mains. It keeps track of the internal state [stateRelayConnected](#).
- void [connectRelay](#) ()
This function is used to connect the relay to the mains. It keeps track of the internal state [stateRelayConnected](#).
- bool [getStateRelayConnected](#) ()
This function is used to get the connection state of the relay with the mains. It returns [stateRelayConnected](#).
- void [toggleRelay](#) ()
This function is used to toggle the connection of the relay with the mains. Indirectly, it keeps track of the internal state [stateRelayConnected](#).
- void [configureOutputs](#) ()
This function configures the output pins of the microcontroller.
- String [renderHeader](#) ()
This function returns the first part of a HTML file which is reused for all responses.
- String [renderRelay](#) (bool [stateRelayConnected](#))
This function returns the middle part of a HTML file to.
- String [renderMQTTSettings](#) (String [storedServerName](#), String [failureMsg](#), bool [stateMQTTActivated](#), bool [stateMQTTConnected](#))
This function returns the middle part of a HTML file to.
- String [renderOtaPasswordChangeForm](#) (String [infoMsg](#))
This function returns the middle part of a HTML file to.
- String [renderFactoryReset](#) (String [infoMsg](#))
This function returns the middle part of a HTML file to.
- String [renderWiFiCredentialChange](#) (String [infoMsg](#))
This function returns the middle part of a HTML file to.
- String [renderInfo](#) ()
This function returns the middle part of a HTML file to.
- String [renderFooter](#) ()
This function returns the last part of a HTML file which is reused for all responses.
- void [mqttControlRelay](#) (String &topic, String &payload)
Callback which is called when MQTT receives an incoming topic.
- void [restoreMQTTConfigurationFromEEPROM](#) ()
Read back MQTT server / broker name from EEPROM and store it in global variable [mqttServer](#). Read back whether MQTT is configured and store it in global variable [stateMQTTConfigured](#).
- void [saveMQTTConfigurationToEEPROM](#) ()
Store MQTT server / broker name ([mqttServer](#)) and state (configured or not, [stateMQTTConfigured](#)) in EEPROM.
- void [configureMQTT](#) ()
Configure new MQTT server / broker. This will.
- void [setMQTTServer](#) (String input)
Setter function for MQTT server / broker [mqttServer](#).
- String [getMQTTServer](#) ()
Getter function for MQTT server / broker [mqttServer](#).
- void [setStateMQTTConfigured](#) (bool input)
Setter function for [stateMQTTConfigured](#).
- bool [getStateMQTTConfigured](#) ()
Getter function for [stateMQTTConfigured](#).

- void [setMQTTOutgoingTopic](#) (String input)
Setter function for [mqttOutgoingTopic](#).
- String [getMQTTOutgoingTopic](#) ()
Getter function for [mqttOutgoingTopic](#).
- void [setMQTTIncomingTopic](#) (String input)
Setter function for [mqttIncomingTopic](#).
- bool [checkMQTTConnected](#) ()
Get MQTT connection state.
- void [mqttConnect](#) ()
This function connects to broker and.
- void [mqttPublish](#) ()
With the call of this function, [mqttClient](#) publishes the state of the relay ([getStateRelayConnected\(\)](#)) on the outgoing topic ([getMQTTOutgoingTopic\(\)](#)) to the MQTT server / broker. The macro [NUMERIC_RESPONSE](#) is taken into account.
- bool [isNotWhitelisted](#) (char c)
This function is used for filtering out XSS attacks. This is done by whitelisting.
- bool [mqttServerValid](#) (String [mqttServer](#))
This function is used to check whether [mqttServer](#) is valid. Therefore, it is checked that the.
- void [configureWebServer](#) ()
This function is used configure the webserver. The webserver is configured by defining callback functions for handling incoming requests on.
- void [performFactoryReset](#) ()
Perform factory reset.
- void [reboot](#) ()
Perform a reboot.
- bool [getFactoryResetRequested](#) ()
Getter function for [factoryResetRequested](#).
- void [setFactoryResetRequested](#) ()
Setter function to set [factoryResetRequested](#).
- void [pressHandler](#) ()
This function triggers.
- void [restoreOtaPasswordFromEEPROM](#) ()
Read back OTA password from EEPROM and store it in global variable [arduinoOtaPassword](#).
- void [saveOtaPasswordToEEPROM](#) ()
Store OTA password in EEPROM.
- void [callbackConfigSuccessful](#) ()
This function is called when a connection has been established.
- void [callbackConfigMode](#) (WiFiManager *myWiFiManager)
This function is called when WiFiManager is in config mode.
- void [setup](#) (void)
This function is executed once at the startup of the microcontroller.
- void [loop](#) (void)
This function is executed in a loop after [setup\(void\)](#) has been called.

3.2.3 *

Variables

- String [VERSION](#) = [VERSION_BY_GIT_DESCRIBE](#)
Version number - obtained by 'git describe'.

- String `REPOSITORY_URL_STRING` = "https://github.com/peastone/WIFIOnOff"
Contains a link to the code repository which is embedded in the rendered HTML files in the function `renderFooter()`.
- String `arduinoOtaPassword` = ""
Global variable to store the Arduino OTA password.
- const int `pinLED` = 13
Constant to map the green LED of the Sonoff S20.
- bool `stateLEDOn` = false
State to track whether the LED is on.
- bool `userActionFeedbackRequest` = false
State to track whether the user should be notified that a menu can be selected by releasing the button.
- WiFiClient `wifiClient`
Necessary to initialize a MQTTClient object.
- String `credentialSsid`
Global variable SSID for WiFi credentials change.
- String `credentialPassword`
Global variable SSID for WiFi credentials change.
- bool `wifiCredentialChangeRequested` = false
State to track whether a change of WiFi credentials has been requested.
- bool `stateRelayConnected` = false
State to track whether the LED is connected to the mains.
- const int `pinRelay` = 12
Constant to map the relay of the Sonoff S20.
- MQTTClient `mqttClient`
central object to manage MQTT
- String `mqttServer` = ""
Global variable to store the DNS name or IP address of the MQTT broker.
- bool `stateMQTTConfigured` = false
State to track whether MQTT is configured.
- String `mqttOutgoingTopic`
The topic `mqttClient` publishes.
- String `mqttIncomingTopic`
The topic which `mqttClient` subscribes.
- ESP8266WebServer `webserver`
This is the webserver object. It is used to serve the user interface over HTTP. Per default, port 80 is used.
- WiFiManager `wifiManager`
This object is a handle for the captive portal / WiFiManager.
- bool `factoryResetRequested` = false
State to track whether a factory reset has been requested.
- const int `buttonPin` = 0
Constant to map the hardware button of the Sonoff S20.
- bool `buttonLastPressed` = false
State to track whether the button was pressed since `pressHandler()` was called the last time.
- unsigned long `timeSincebuttonPressed` = 0
State to track the time since the button was pressed.
- unsigned long `selectionState` = 0
State to track which menu the user selects, if the button is released.
- Ticker `ticker`
This object is used to trigger the function `press` frequently. The button handler is implemented there.

3.2.4 Detailed Description

WIFIONOFF is an alternative software for the Sonoff S20 which provides a web user interface (HTTP) and an internet of things interface (MQTT). The device can still be controlled by the normal user button. The connection to WiFi will be established by WiFiManager. This device can be used in the local network without any dependency of a cloud.

Copyright (C) 2019 Siegfried Schöfer

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

See also

<http://mqtt.org/>
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>
<https://github.com/esp8266/Arduino>
<https://arduino-esp8266.readthedocs.io/en/latest/>
<https://media.readthedocs.org/pdf/arduino-esp8266/latest/arduino-esp8266.pdf>
https://github.com/espressif/ESP8266_NONOS_SDK

3.2.5 Macro Definition Documentation

3.2.5.1 CTR_MAX_TRIES_WIFI_CONNECTION

```
#define CTR_MAX_TRIES_WIFI_CONNECTION 60
```

maximum amount of tries to connect to WIFI

3.2.5.2 DEV_OTA_UPDATES

```
#define DEV_OTA_UPDATES
```

Define DEV_OTA_UPDATES if you want to perform OTA updates for development with Arduino IDE.

3.2.5.3 EEPROM_address_CRC

```
#define EEPROM_address_CRC 1
```

The EEPROM version number is stored at this address.

3.2.5.4 EEPROM_address_MQTT_server

```
#define EEPROM_address_MQTT_server (EEPROM_address_MQTT_server_configured + EEPROM_enabled_size)
```

The variable `mqttServer` which contains the value of the MQTT-Server (DNS name or IP) is restored after startup from EEPROM, if MQTT is configured (`EEPROM_address_MQTT_server_configured`). The stored bytes can be found at this address.

3.2.5.5 EEPROM_address_MQTT_server_configured

```
#define EEPROM_address_MQTT_server_configured EEPROM_address_start
```

This flag is used to check whether `mqttServer` has already been initialized with the value of the MQTT-Server (DNS name or IP)

3.2.5.6 EEPROM_address_OTA_password

```
#define EEPROM_address_OTA_password (EEPROM_address_MQTT_server + EEPROM_size_MQTT_server)
```

The OTA password is stored here.

3.2.5.7 EEPROM_address_start

```
#define EEPROM_address_start (EEPROM_address_CRC + EEPROM_size_CRC)
```

A CRC is calculated for the EEPROM values to ensure that the values are untempered. The CRC takes some space to be stored in the EEPROM. The start address takes this into account.

3.2.5.8 EEPROM_enabled

```
#define EEPROM_enabled 0xEB
```

If this magic number is set, a defined functionality is enabled.

3.2.5.9 EEPROM_enabled_size

```
#define EEPROM_enabled_size 1
```

The amount of storage needed by `EEPROM_enabled` in EEPROM.

3.2.5.10 EEPROM_init_value

```
#define EEPROM_init_value 0x00
```

If an init value is set, a defined functionality is disabled. The init value is also used to initialize the EEPROM in `setupEEPROM()`.

3.2.5.11 EEPROM_length

```
#define EEPROM_length 4096
```

`EEPROM_length` defines the maximum amount of bytes to be stored in the EEPROM. At the time of writing, the maximum amount of bytes to be stored is 4096.

See also

<https://git.io/vxYHO>

<https://git.io/vxYHG>

3.2.5.12 EEPROM_size_CRC

```
#define EEPROM_size_CRC (sizeof(unsigned long))
```

A CRC is calculated for the EEPROM values to ensure that the values are untempered. The size of the stored CRC is defined by this macro.

3.2.5.13 EEPROM_size_MQTT_server

```
#define EEPROM_size_MQTT_server 256
```

The maximum amount of bytes used to store the MQTT server address. This address can be either an IP or a DNS name. DNS names are restricted to 255 octets. As a string is zero-terminated, one byte more is used. For more information, look at RFC 1035.

See also

<https://www.ietf.org/rfc/rfc1035.txt>

3.2.5.14 EEPROM_size_OTA_password

```
#define EEPROM_size_OTA_password 256
```

Define length for OTA password.

3.2.5.15 EEPROM_version_number

```
#define EEPROM_version_number 1
```

A CRC is calculated for the EEPROM values to ensure that the values are untempered. The address of the stored CRC is at the beginning of the EEPROM.

3.2.5.16 EEPROM_version_number_address

```
#define EEPROM_version_number_address 0
```

This version number is stored in EEPROM. It must not be longer than one byte. The version number defined here and that one stored in EEPROM are compared against each other. In case the two numbers are not equal, the

EEPROM is re-/initialized. This number must change, if you change the EEPROM layout, basically if you change addresses.

3.2.5.17 HTTP_PORT

```
#define HTTP_PORT 80
```

standard port used for the HTTP protocol

3.2.5.18 MQTT_PORT

```
#define MQTT_PORT 1883
```

standard port used for the MQTT protocol

3.2.5.19 NUMERIC_RESPONSE

```
#define NUMERIC_RESPONSE
```

Define NUMERIC_RESPONSE if you want the [mqttClient](#) to publish "1" or "0" instead of "on" or "off".

3.2.5.20 OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED

```
#define OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED
```

The state of the relay is visible through the blue LED (connected == LED shines). Define OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED, if you want the state of the relay being visible also on the green LED (connected == LED shines).

3.2.5.21 SERIAL_PRINTING

```
#define SERIAL_PRINTING
```

Define SERIAL_PRINTING if you want to enable serial communication.

3.2.5.22 TIME_EEPROM_WARNING

```
#define TIME_EEPROM_WARNING 7000
```

definition of the time the LED stays on in case of an EEPROM warning (CRC failure, initialization, EEPROM incompatible)

3.2.5.23 TIME_HALF_A_SECOND

```
#define TIME_HALF_A_SECOND 500
```

definition of the timespan of half a second

3.2.5.24 TIME_QUARTER_SECOND

```
#define TIME_QUARTER_SECOND 250
```

definition of the timespan of a quarter of a second

3.2.5.25 TRIGGER_TIME_FACTORY_RESET

```
#define TRIGGER_TIME_FACTORY_RESET 15000
```

time in ms to wait until factory reset is triggered when pressing the button (see [pressHandler\(\)](#))

3.2.6 Function Documentation

3.2.6.1 calculate_crc()

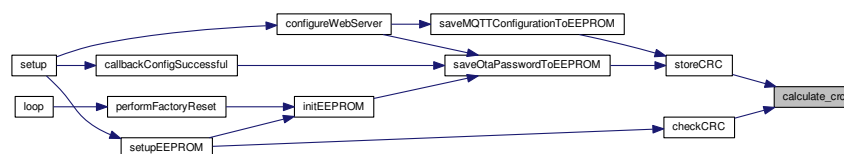
```
unsigned long calculate_crc ( )
```

This function calculates a CRC checksum for the EEPROM. It is taken from: <https://www.arduino.cc/en/Tutorial/EEPROMCrc>.

Returns

calculated CRC

Here is the caller graph for this function:



3.2.6.2 callbackConfigMode()

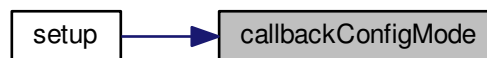
```
void callbackConfigMode (  
    WiFiManager * myWiFiManager )
```

This function is called when WiFiManager is in config mode.

Here is the call graph for this function:



Here is the caller graph for this function:

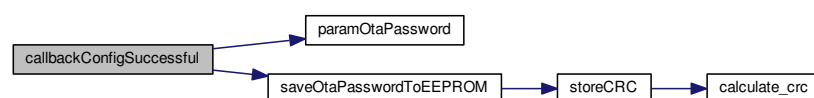


3.2.6.3 callbackConfigSuccessful()

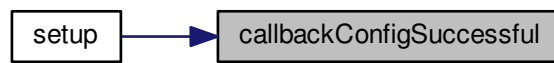
```
void callbackConfigSuccessful ( )
```

This function is called when a connection has been established.

Here is the call graph for this function:



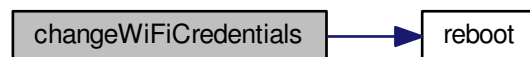
Here is the caller graph for this function:



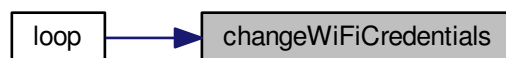
3.2.6.4 `changeWiFiCredentials()`

```
void changeWiFiCredentials ( )
```

This function changes the WiFi credentials. Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.5 `checkCRC()`

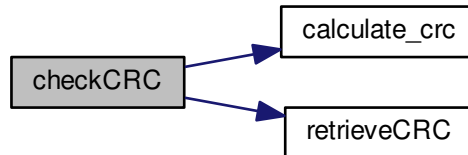
```
bool checkCRC ( )
```

This function calculates the CRC value for the bytes stored in the EEPROM with the help of function [calculate_crc\(\)](#) and compares the result to the CRC which was read from EEPROM with the help of function [retrieveCRC\(\)](#).

Returns

true if calculated CRC matches stored CRC

Here is the call graph for this function:



Here is the caller graph for this function:

**3.2.6.6 checkEEPROMVersionNumber()**

```
bool checkEEPROMVersionNumber ( )
```

This function checks the [EEPROM_version_number](#) stored in EEPROM against the version number required by the latest EEPROM layout.

Returns

true, if the stored and the required EEPROM version number match, false otherwise

Here is the caller graph for this function:



3.2.6.7 checkMQTTConnected()

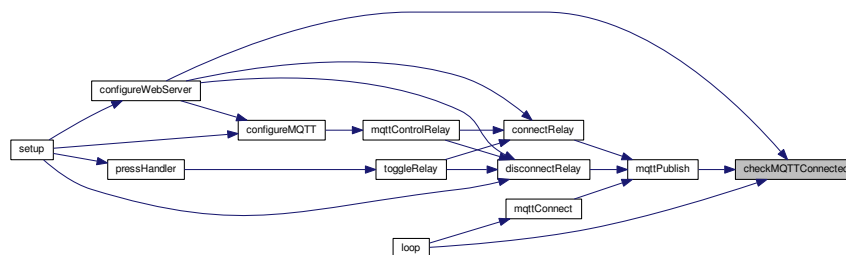
```
bool checkMQTTConnected ( )
```

Get MQTT connection state.

Returns

true, if MQTT is connected, false otherwise

Here is the caller graph for this function:



3.2.6.8 checkWiFiConnected()

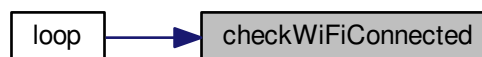
```
bool checkWiFiConnected ( )
```

This function checks whether WIFI is connected.

Returns

true, if WiFi is connected, false otherwise

Here is the caller graph for this function:



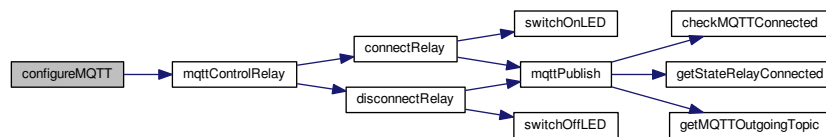
3.2.6.9 configureMQTT()

```
void configureMQTT ( )
```

Configure new MQTT server / broker. This will.

- disconnect the MQTT client if connected
- set the server / broker name
- set the callback mqttControlRelay()

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.10 configureOutputs()

```
void configureOutputs ( )
```

This function configures the output pins of the microcontroller.

Here is the caller graph for this function:



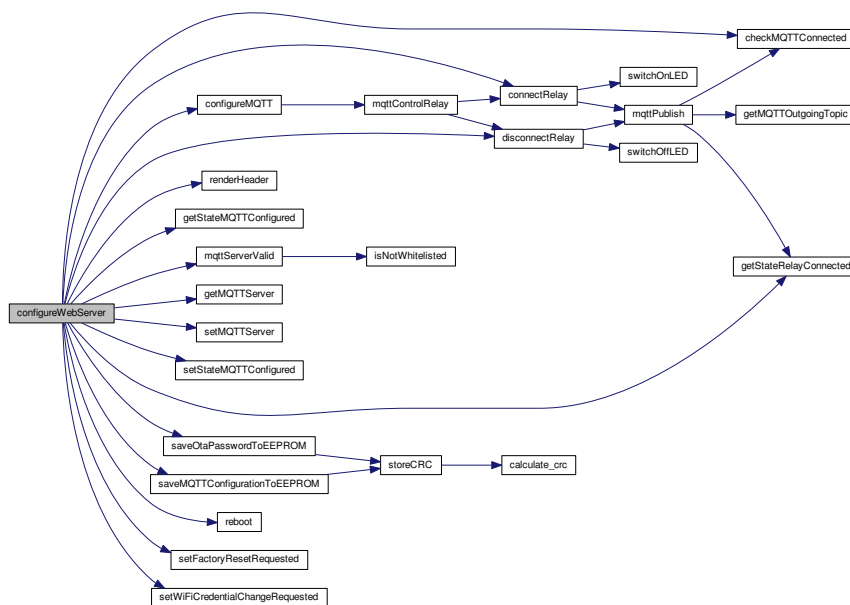
3.2.6.11 configureWebServer()

```
void configureWebServer ( )
```

This function is used configure the webserver. The webserver is configured by defining callback functions for handling incoming requests on.

- /
- /settings.html

Here is the call graph for this function:



Here is the caller graph for this function:

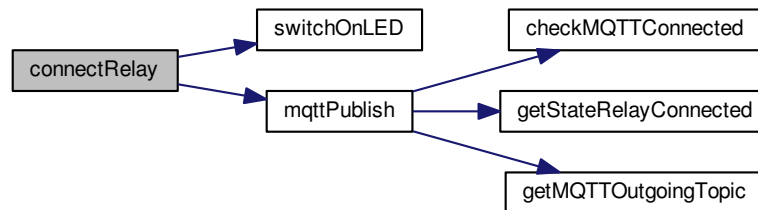


3.2.6.12 connectRelay()

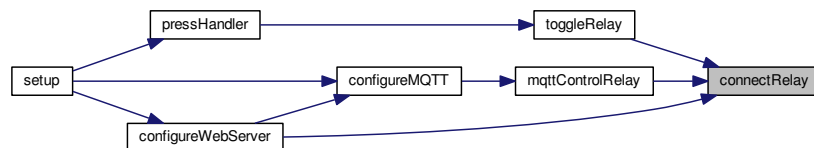
```
void connectRelay ( )
```

This function is used to connect the relay to the mains. It keeps track of the internal state [stateRelayConnected](#).

Here is the call graph for this function:



Here is the caller graph for this function:

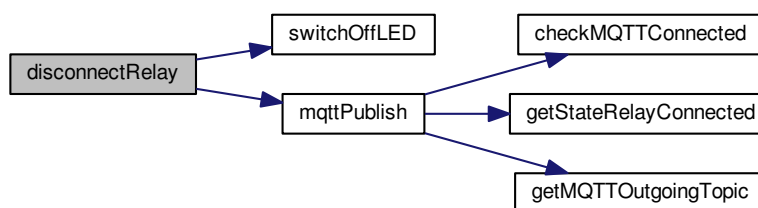


3.2.6.13 disconnectRelay()

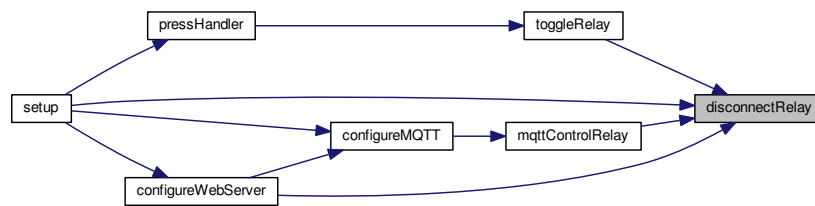
```
void disconnectRelay ( )
```

This function is used to disconnect the relay from the mains. It keeps track of the internal state [stateRelayConnected](#).

Here is the call graph for this function:



Here is the caller graph for this function:

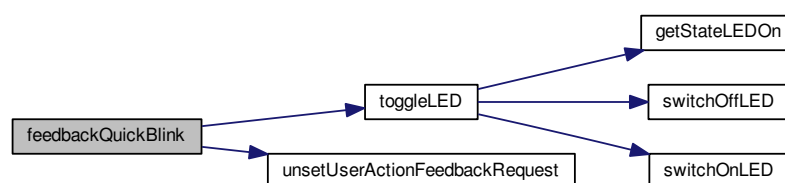


3.2.6.14 feedbackQuickBlink()

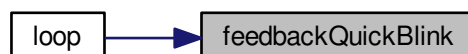
```
void feedbackQuickBlink ( )
```

This function is called to give the user feedback about the menu the user selected. It will blink fast for a short period of time. The user has the choice to release and select the menu or to wait for the next menu. By counting the [feedbackQuickBlink\(\)](#)-events, the user can determine which menu is selected, when the button is released now.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.15 getClientID()

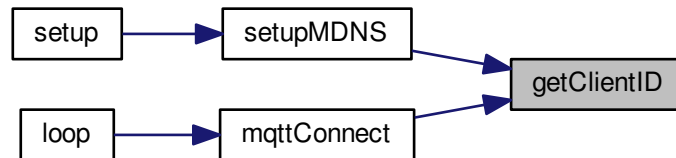
```
String getClientID ( )
```

This function returns the client name used for MQTT, see [mqttConnect\(\)](#).

Returns

the client ID: "WIFIOff" + MAC address

Here is the caller graph for this function:

**3.2.6.16 getDefaultOtaPassword()**

```
String getDefaultOtaPassword ( )
```

3.2.6.17 getFactoryResetRequested()

```
bool getFactoryResetRequested ( )
```

Getter function for [factoryResetRequested](#).

Returns

true, if factory reset is requested, false otherwise.

3.2.6.18 getMQTTOutgoingTopic()

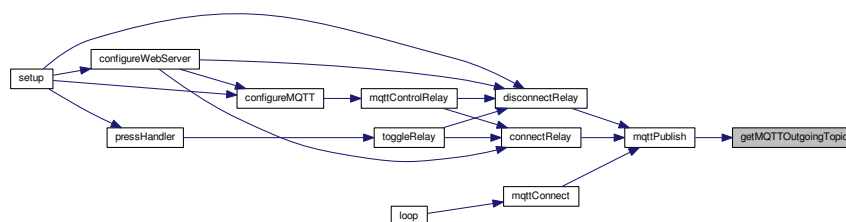
```
String getMQTTOutgoingTopic ( )
```

Getter function for [mqttOutgoingTopic](#).

Returns

String of outgoing MQTT topic

Here is the caller graph for this function:



3.2.6.19 getMQTTServer()

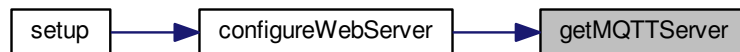
```
String getMQTTServer ( )
```

Getter function for MQTT server / broker [mqttServer](#).

Returns

the latest MQTT server / broker

Here is the caller graph for this function:



3.2.6.20 getStateLEDOn()

```
bool getStateLEDOn ( )
```

This function is used to get the state of the LED. It returns [stateLEDOn](#).

Returns

true, if the LED is on, false otherwise.

Here is the caller graph for this function:



3.2.6.21 getStateMQTTConfigured()

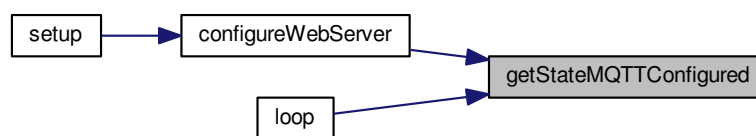
```
bool getStateMQTTConfigured ( )
```

Getter function for [stateMQTTConfigured](#).

Returns

true if MQTT is configured, false otherwise

Here is the caller graph for this function:



3.2.6.22 getStateRelayConnected()

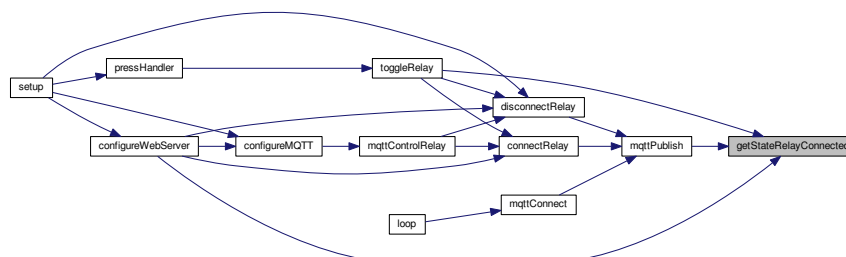
```
bool getStateRelayConnected ( )
```

This function is used to get the connection state of the relay with the mains. It returns [stateRelayConnected](#).

Returns

true, if the relay is connected to the mains, false otherwise.

Here is the caller graph for this function:



3.2.6.23 getUserActionFeedbackRequest()

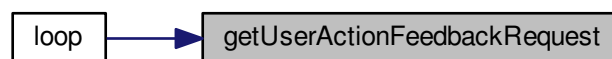
```
bool getUserActionFeedbackRequest ( )
```

Getter function for [userActionFeedbackRequest](#).

Returns

true, if the user should be notified that a menu can be selected, false otherwise

Here is the caller graph for this function:



3.2.6.24 getWiFiCredentialChangeRequested()

```
bool getWiFiCredentialChangeRequested ( )
```

Getter function for [wifiCredentialChangeRequested](#).

Returns

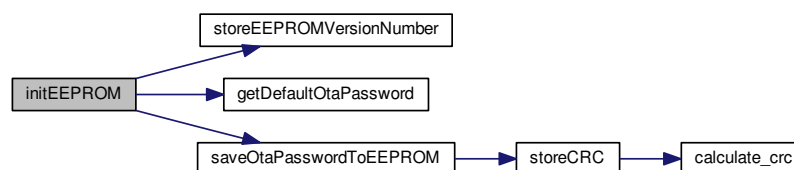
true, if WiFiManager has been requested, false otherwise.

3.2.6.25 initEEPROM()

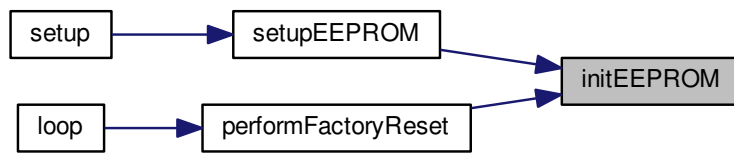
```
void initEEPROM ( )
```

This function writes default data ([EEPROM_init_value](#)) to EEPROM and stores CRC and the [EEPROM_version_number](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.26 isNotWhitelisted()

```
bool isNotWhitelisted (
    char c )
```

This function is used for filtering out XSS attacks. This is done by whitelisting.

- IPv4 addresses consist of numbers and dots (0..9 | .).
- IPv6 addresses consist of hexadecimal numbers, double dots or brackets (0..9 | a..f | A..F | [|])
- DNS names consist of letters, digits and hyphen (0..9 | a..z | A..Z | -). This function is not validating DNS, IPv4, IPv6. Rubbish DNS names can still pass. But characters which could be used for XSS, like <, >, &, " are blocked.

Parameters

in	c	character to check
----	---	--------------------

See also

<https://www.ietf.org/rfc/rfc1035.txt>
<https://wonko.com/post/html-escaping>
[https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

Returns

true, if the argument character is not whitelisted, false otherwise.

Here is the caller graph for this function:



3.2.6.27 loop()

```
void loop (
    void )
```

This function is executed in a loop after [setup\(void\)](#) has been called.

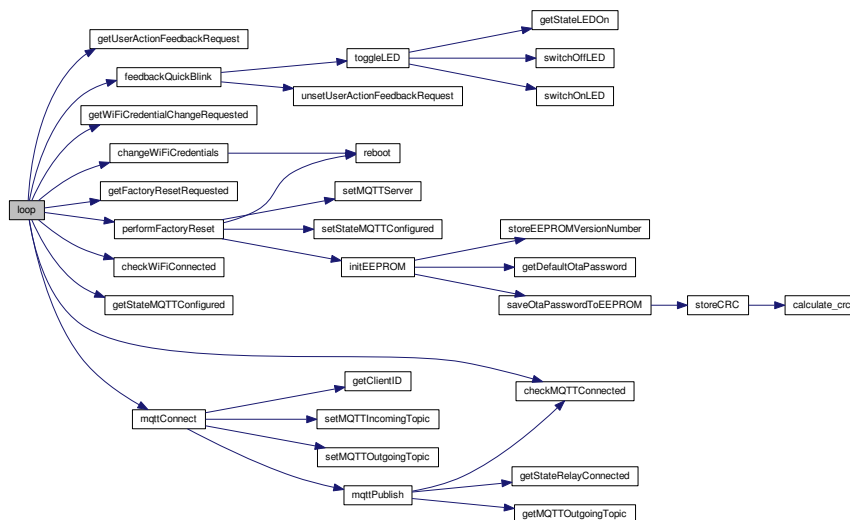
At first it checks for incoming user requests:

- to change WiFi credentials
- to perform a factory reset

It is also checked for requests to inform the user about actions with the LED. These requests are triggered programmatically.

After all requests are handled, it is cyclically checked that the WiFi is connected. If WiFi is not connected, it does not make any sense to check for HTTP, ArduinoOTA or MQTT. For MQTT to be checked, it is also required, that it was enabled by the user.

If MQTT is not connected, try to connect, otherwise handle MQTT. Here is the call graph for this function:



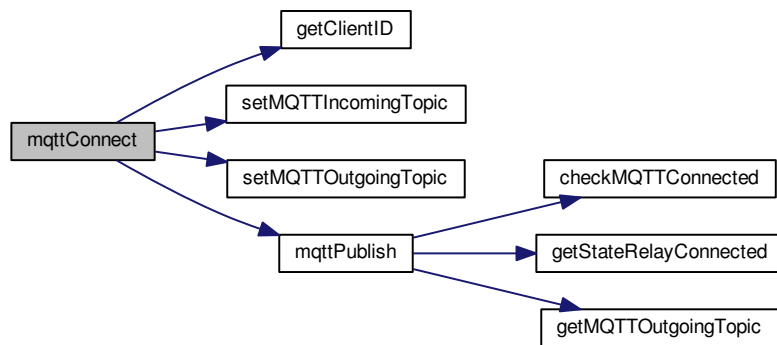
3.2.6.28 mqttConnect()

```
void mqttConnect ( )
```

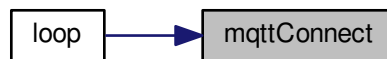
This function connects to broker and.

- subscribes topic wifionoff/getClientID()/set
- sets last will "disconnected" on wifionoff/getClientID()/get
- publishes latest state on wifionoff/getClientID()/get

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.29 mqttControlRelay()

```

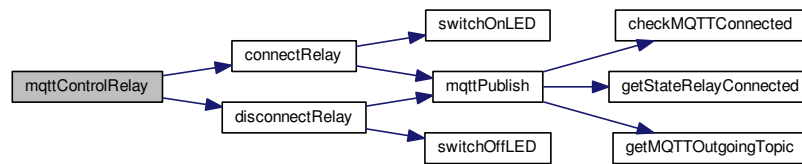
void mqttControlRelay (
    String & topic,
    String & payload )
  
```

Callback which is called when MQTT receives an incoming topic.

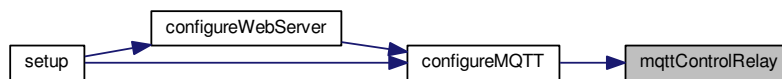
Parameters

in	<i>topic</i>	One of the incoming MQTT topics, which the mqttClient has been subscribed to. Currently there is only one connection, see mqttConnect() . So this parameter is irrelevant.
in	<i>payload</i>	Contains the received payload of the message of the incoming topic. If "on" or "1" is received, the relay is connected (connectRelay()), else if "off" or "0" is received, the relay is disconnected (disconnectRelay()).

Here is the call graph for this function:



Here is the caller graph for this function:

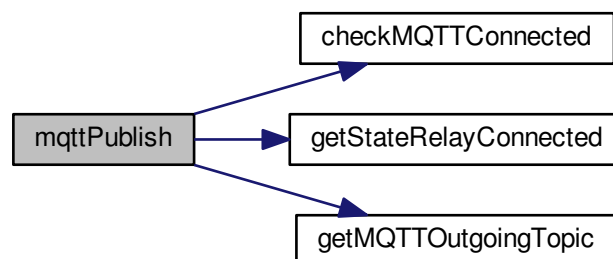


3.2.6.30 mqttPublish()

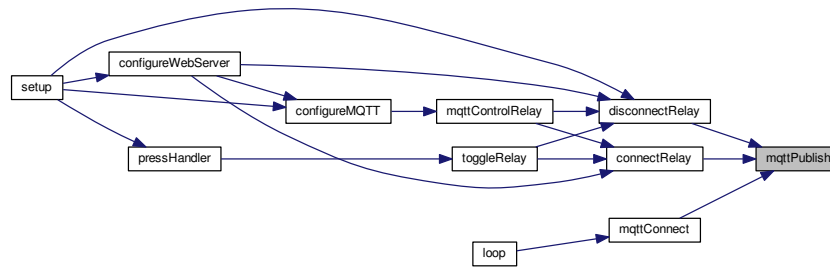
```
void mqttPublish ( )
```

With the call of this function, [mqttClient](#) publishes the state of the relay ([getStateRelayConnected\(\)](#)) on the outgoing topic ([getMQTTOutgoingTopic\(\)](#)) to the MQTT server / broker. The macro [NUMERIC_RESPONSE](#) is taken into account.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.31 mqttServerValid()

```
bool mqttServerValid (
    String mqttServer )
```

This function is used to check whether mqttServer is valid. Therefore, it is checked that the.

- length of mqttServer is less or equal than [EEPROM_size_MQTT_server](#)
- all characters are whitelisted (see [isNotWhitelisted\(\)](#))

Parameters

in	<i>mqttServer</i>	String to check for validity
----	-------------------	------------------------------

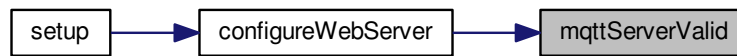
Returns

true, if mqttServer is valid, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.32 paramOtaPassword()

```

WiFiManagerParameter paramOtaPassword (
    "ota" ,
    "Arduino OTA Password" ,
    getDefaultOtaPassword().c_str() ,
    EEPROM_size_OTA_password- 1,
    "" )
  
```

Password to protect OTA updates. This password should be changed in WiFiManager.

See also

<https://media.readthedocs.org/pdf/arduino-esp8266/latest/arduino-esp8266.pdf>

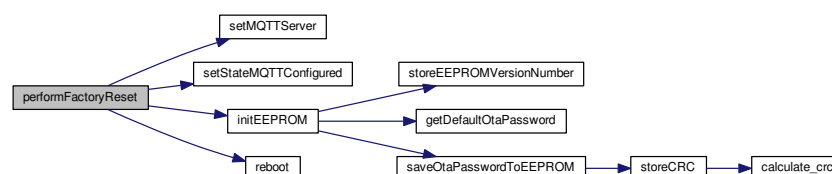
3.2.6.33 performFactoryReset()

```
void performFactoryReset ( )
```

Perform factory reset.

- reset WiFi
- reset MQTT
- reset EEPROM

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.34 pressHandler()

```
void pressHandler ( )
```

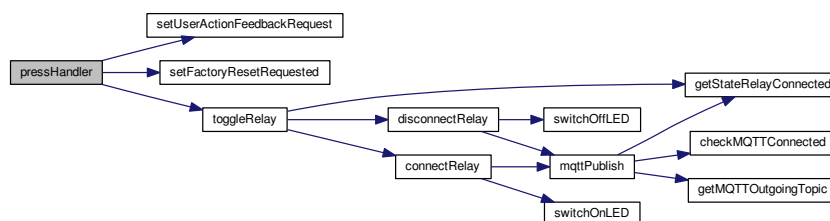
This function triggers.

- the toggling of the relay
- factory reset request. As this function is called regularly with [ticker](#), it should be quick, otherwise the ESP8266 might crash.

See also

<https://arduino-esp8266.readthedocs.io/en/latest/faq/a02-my-esp-crashes.%3Cbr%3Ehtml>

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.35 reboot()

```
void reboot ( )
```

Perform a reboot.

3.2.6.36 renderFactoryReset()

```
String renderFactoryReset (
    String infoMsg )
```

This function returns the middle part of a HTML file to.

- perform a factory reset

Parameters

in	<i>infoMsg</i>	signal whether the factory reset will be performed
----	----------------	--

3.2.6.37 renderFooter()

```
String renderFooter ( )
```

This function returns the last part of a HTML file which is reused for all responses.

Returns

"</body></html>", look inside the function for more information.

3.2.6.38 renderHeader()

```
String renderHeader ( )
```

This function returns the first part of a HTML file which is reused for all responses.

Returns

"<!doctype> ... <body>", look inside the function for more information.

See also

<http://www.html.am/templates/css-templates/>

Here is the caller graph for this function:



3.2.6.39 `renderInfo()`

```
String renderInfo ( )
```

This function returns the middle part of a HTML file to.

- show the version of the WIFIO nOff
- show the default OTA PW

Returns

HTML, look inside the function for more information.

Here is the call graph for this function:



3.2.6.40 `renderMQTTSettings()`

```
String renderMQTTSettings (
    String storedServerName,
    String failureMsg,
    bool stateMQTTActivated,
    bool stateMQTTConnected )
```

This function returns the middle part of a HTML file to.

- show the settings for the MQTT server
- manipulate the settings for the MQTT server This function could be relevant concerning XSS.

Parameters

in	<i>storedServerName</i>	DNS name or IP address which is displayed on the HTML site
in	<i>failureMsg</i>	message that is displayed to the user in case of success or failure
in	<i>stateMQTTActivated</i>	shows the user, whether MQTT is activated
in	<i>stateMQTTConnected</i>	shows the user, whether MQTT is connected

Returns

HTML, look inside the function for more information.

See also

[https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

3.2.6.41 renderOtaPasswordChangeForm()

```
String renderOtaPasswordChangeForm (
    String infoMsg )
```

This function returns the middle part of a HTML file to.

- change the OTA password

Parameters

in	<i>infoMsg</i>	signal whether the change of the OTA password was successful or not
----	----------------	---

3.2.6.42 renderRelay()

```
String renderRelay (
    bool stateRelayConnected )
```

This function returns the middle part of a HTML file to.

- show the state of the relay (disconnected - off / connected - on).
- manipulate the state of the relay.

Parameters

in	<i>stateRelayConnected</i>	if true, the relay is displayed as on, otherwise as off
----	----------------------------	---

Returns

HTML, look inside the function for more information.

3.2.6.43 renderWiFiCredentialChange()

```
String renderWiFiCredentialChange (
    String infoMsg )
```

This function returns the middle part of a HTML file to.

- changeWiFiCredentials

Parameters

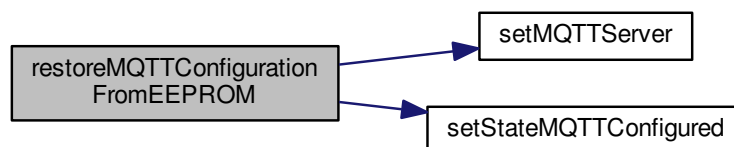
in	<i>infoMsg</i>	signal whether the change of WiFiCredentials will be performed.
----	----------------	---

3.2.6.44 restoreMQTTConfigurationFromEEPROM()

```
void restoreMQTTConfigurationFromEEPROM ( )
```

Read back MQTT server / broker name from EEPROM and store it in global variable [mqttServer](#). Read back whether MQTT is configured and store it in global variable [stateMQTTConfigured](#).

Here is the call graph for this function:



Here is the caller graph for this function:

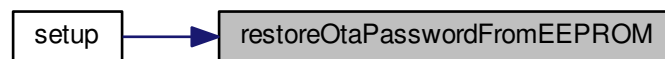


3.2.6.45 restoreOtaPasswordFromEEPROM()

```
void restoreOtaPasswordFromEEPROM ( )
```

Read back OTA password from EEPROM and store it in global variable [arduinoOtaPassword](#).

Here is the caller graph for this function:



3.2.6.46 retrieveCRC()

```
unsigned long retrieveCRC ( )
```

This function reads the CRC value stored at [EEPROM_address_CRC](#).

Returns

CRC value from EEPROM

Here is the caller graph for this function:



3.2.6.47 saveMQTTConfigurationToEEPROM()

```
void saveMQTTConfigurationToEEPROM ( )
```

Store MQTT server / broker name ([mqttServer](#)) and state (configured or not, [stateMQTTConfigured](#)) in EEPROM.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.48 saveOtaPasswordToEEPROM()

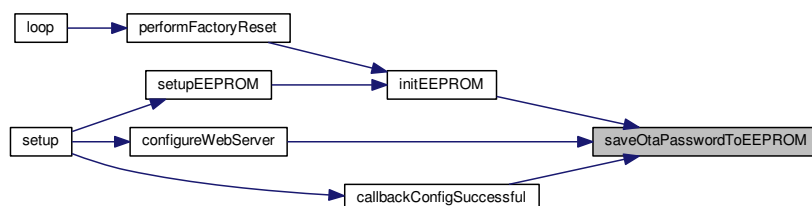
```
void saveOtaPasswordToEEPROM ( )
```

Store OTA password in EEPROM.

Here is the call graph for this function:



Here is the caller graph for this function:

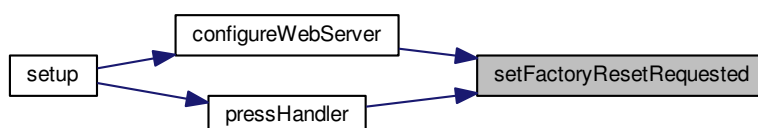


3.2.6.49 setFactoryResetRequested()

```
void setFactoryResetRequested ( )
```

Setter function to set [factoryResetRequested](#).

Here is the caller graph for this function:

**3.2.6.50 setMQTTIncomingTopic()**

```
void setMQTTIncomingTopic (
    String input )
```

Setter function for [mqttIncomingTopic](#).

Parameters

in	<i>input</i>	String to be set
----	--------------	------------------

Here is the caller graph for this function:

**3.2.6.51 setMQTTOutgoingTopic()**

```
void setMQTTOutgoingTopic (
    String input )
```

Setter function for [mqttOutgoingTopic](#).

Parameters

in	<i>input</i>	String to be set
----	--------------	------------------

Here is the caller graph for this function:



3.2.6.52 setMQTTServer()

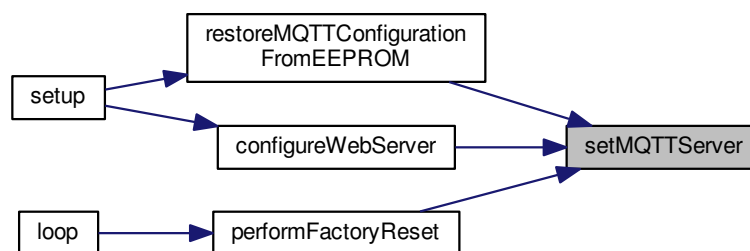
```
void setMQTTServer (
    String input )
```

Setter function for MQTT server / broker [mqttServer](#).

Parameters

in	<i>input</i>	MQTT server / broker
----	--------------	----------------------

Here is the caller graph for this function:



3.2.6.53 setStateMQTTConfigured()

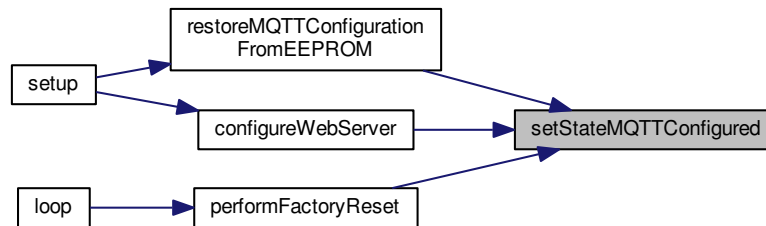
```
void setStateMQTTConfigured (
    bool input )
```

Setter function for [stateMQTTConfigured](#).

Parameters

in	input	true if MQTT is configured, false otherwise
----	-------	---

Here is the caller graph for this function:

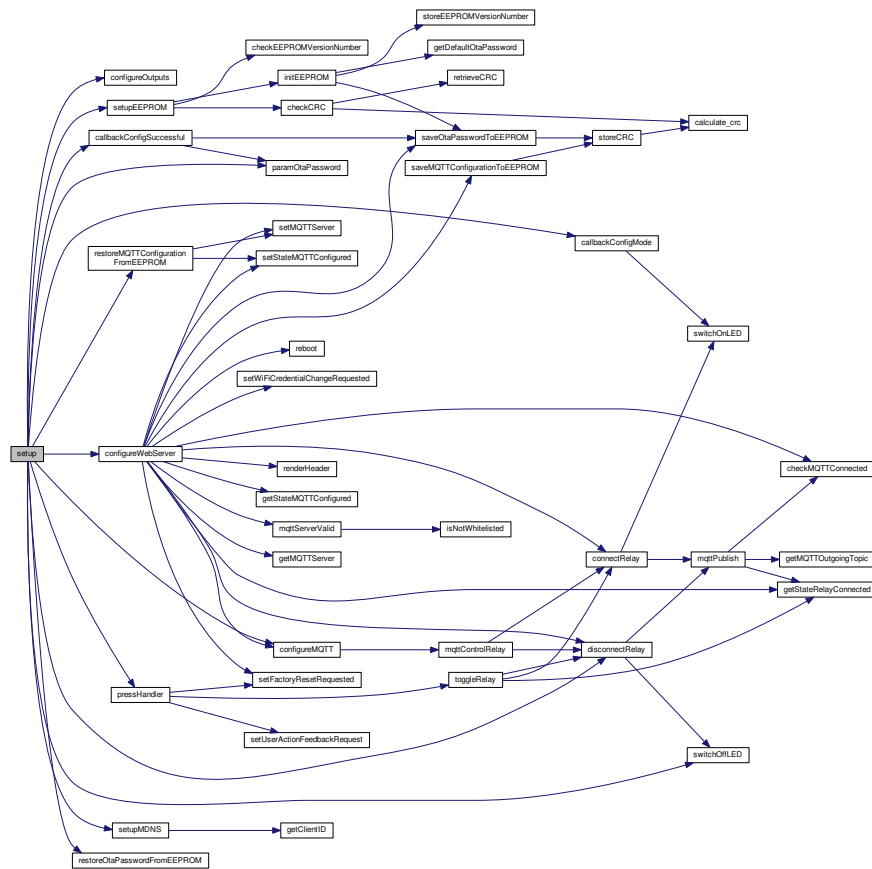
3.2.6.54 `setup()`

```
void setup (
    void )
```

This function is executed once at the startup of the microcontroller.

- The serial communication is setup with 115200 bauds. It is usefull for debugging or just information.
- The output pins are configured.
- The EEPROM is setup.
- The MQTT configuration is restored from EEPROM.
- MQTT is configured.
- The webserver is configured.
- MDNS is set up.
- The LED is switched off.
- The relay is disconnected. This is thought to be the natural experience, if you plug in a socket.
- The button handler is started.

Here is the call graph for this function:



3.2.6.55 setupEEPROM()

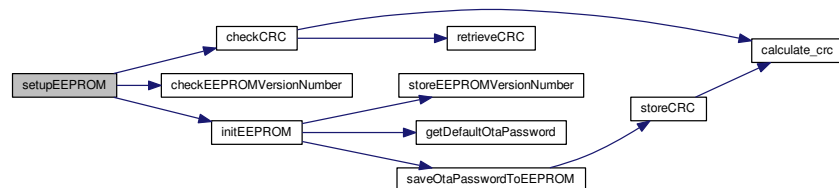
```
void setupEEPROM ( )
```

This function is setting up the EEPROM. If the CRC check fails, the EEPROM will be overwritten with init values. The CRC check makes only sense at initialization phase. Afterwards, the data is buffered by EEPROM lib in RAM.

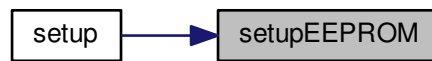
See also

<https://git.io/vxOPf>

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.56 setupMDNS()

```
void setupMDNS ( )
```

This function sets up MDNS.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.57 setUserActionFeedbackRequest()

```
void setUserActionFeedbackRequest ( )
```

Setter function for [userActionFeedbackRequest](#). The user should be notified that a menu can be selected.

Here is the caller graph for this function:



3.2.6.58 `setWiFiCredentialChangeRequested()`

```
void setWiFiCredentialChangeRequested ( )
```

Setter function to set [wifiCredentialChangeRequested](#).

Here is the caller graph for this function:



3.2.6.59 `storeCRC()`

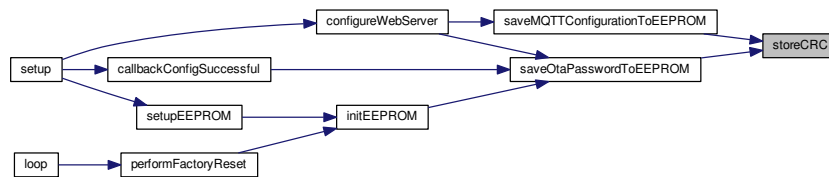
```
void storeCRC ( )
```

This function calculates the CRC value of the values in the EEPROM with the help of function [calculate_crc\(\)](#). The resulting CRC checksum is written to [EEPROM_address_CRC](#). The caller must still call `EEPROM.commit()` afterwards to really trigger a write to EEPROM.

Here is the call graph for this function:



Here is the caller graph for this function:

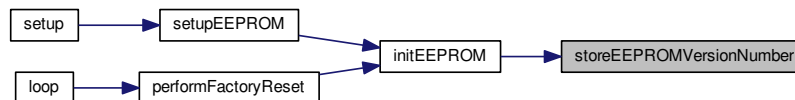


3.2.6.60 storeEEPROMVersionNumber()

```
void storeEEPROMVersionNumber ( )
```

This function stores the [EEPROM_version_number](#) in EEPROM.

Here is the caller graph for this function:

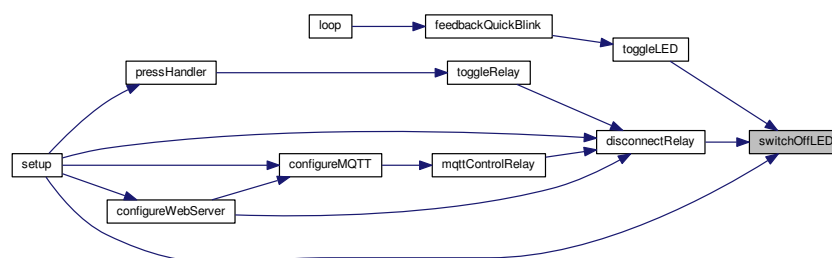


3.2.6.61 switchOffLED()

```
void switchOffLED ( )
```

This function is used to switch the LED off. It keeps track of the internal state [stateLEDOn](#).

Here is the caller graph for this function:

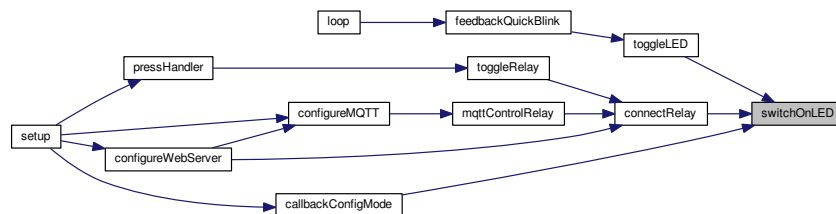


3.2.6.62 switchOnLED()

```
void switchOnLED ( )
```

This function is used to switch the LED on. It keeps track of the internal state [stateLEDOn](#).

Here is the caller graph for this function:

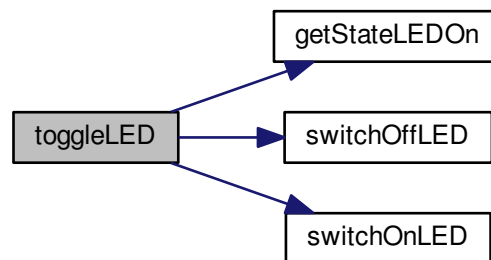


3.2.6.63 toggleLED()

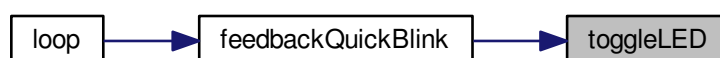
```
void toggleLED ( )
```

This function is used to toggle the LED. Indirectly, it keeps track of the internal state [stateLEDOn](#).

Here is the call graph for this function:



Here is the caller graph for this function:

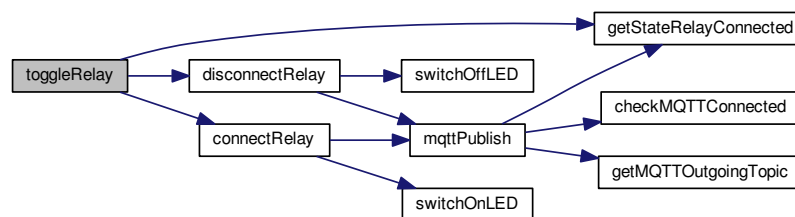


3.2.6.64 toggleRelay()

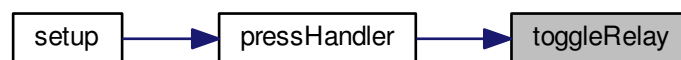
```
void toggleRelay ( )
```

This function is used to toggle the connection of the relay with the mains. Indirectly, it keeps track of the internal state [stateRelayConnected](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.6.65 unsetUserActionFeedbackRequest()

```
void unsetUserActionFeedbackRequest ( )
```

Setter function for [userActionFeedbackRequest](#). The user has been notified that a menu can be selected.

Here is the caller graph for this function:



3.2.7 Variable Documentation

3.2.7.1 `arduinoOtaPassword`

```
String arduinoOtaPassword = ""
```

Global variable to store the Arduino OTA password.

3.2.7.2 `buttonLastPressed`

```
bool buttonLastPressed = false
```

State to track whether the button was pressed since `pressHandler()` was called the last time.

3.2.7.3 `buttonPin`

```
const int buttonPin = 0
```

Constant to map the hardware button of the Sonoff S20.

3.2.7.4 `credentialPassword`

```
String credentialPassword
```

Global variable SSID for WiFi credentials change.

3.2.7.5 `credentialSsid`

```
String credentialSsid
```

Global variable SSID for WiFi credentials change.

3.2.7.6 `factoryResetRequested`

```
bool factoryResetRequested = false
```

State to track whether a factory reset has been requested.

3.2.7.7 `mqttClient`

```
MQTTClient mqttClient
```

central object to manage MQTT

See also

MQTT protocol <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>

3.2.7.8 mqttIncomingTopic

```
String mqttIncomingTopic
```

The topic which [mqttClient](#) subscribes.

3.2.7.9 mqttOutgoingTopic

```
String mqttOutgoingTopic
```

The topic [mqttClient](#) publishes.

3.2.7.10 mqttServer

```
String mqttServer = ""
```

Global variable to store the DNS name or IP address of the MQTT broker.

3.2.7.11 pinLED

```
const int pinLED = 13
```

Constant to map the green LED of the Sonoff S20.

3.2.7.12 pinRelay

```
const int pinRelay = 12
```

Constant to map the relay of the Sonoff S20.

3.2.7.13 REPOSITORY_URL_STRING

```
String REPOSITORY_URL_STRING = "https://github.com/peastone/WIFIOff"
```

Contains a link to the code repository which is embedded in the rendered HTML files in the function [renderFooter\(\)](#).

3.2.7.14 selectionState

```
unsigned long selectionState = 0
```

State to track which menu the user selects, if the button is released.

3.2.7.15 stateLEDOn

```
bool stateLEDOn = false
```

State to track whether the LED is on.

3.2.7.16 stateMQTTConfigured

```
bool stateMQTTConfigured = false
```

State to track whether MQTT is configured.

3.2.7.17 stateRelayConnected

```
bool stateRelayConnected = false
```

State to track whether the LED is connected to the mains.

3.2.7.18 ticker

```
Ticker ticker
```

This object is used to trigger the function press frequently. The button handler is implemented there.

3.2.7.19 timeSincebuttonPressed

```
unsigned long timeSincebuttonPressed = 0
```

State to track the time since the button was pressed.

3.2.7.20 userActionFeedbackRequest

```
bool userActionFeedbackRequest = false
```

State to track whether the user should be notified that a menu can be selected by releasing the button.

3.2.7.21 VERSION

```
String VERSION = VERSION\_BY\_GIT\_DESCRIBE
```

Version number - obtained by 'git describe'.

3.2.7.22 webserver

```
ESP8266WebServer webserver
```

This is the webserver object. It is used to serve the user interface over HTTP. Per default, port 80 is used.

3.2.7.23 wifiClient

```
WiFiClient wifiClient
```

Necessary to initialize a MQTTClient object.

3.2.7.24 wifiCredentialChangeRequested

```
bool wifiCredentialChangeRequested = false
```

State to track whether a change of WiFi credentials has been requested.

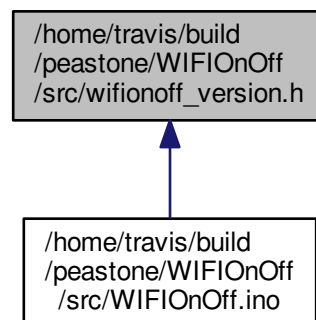
3.2.7.25 wifiManager

```
WiFiManager wifiManager
```

This object is a handle for the captive portal / WiFiManager.

3.3 /home/travis/build/peastone/WIFIOff/src/wifionoff_version.h File Reference

This graph shows which files directly or indirectly include this file:



3.3.1 *

Macros

- `#define VERSION_BY_GIT_DESCRIBE "base-7-g2ee59ff"`
Version number - obtained by 'git describe'.

3.3.2 Macro Definition Documentation

3.3.2.1 VERSION_BY_GIT_DESCRIBE

```
#define VERSION_BY_GIT_DESCRIBE "base-7-g2ee59ff"
```

Version number - obtained by 'git describe'.

Index

[/home/travis/build/peastone/WIFIOff/README.md](#), 5

[/home/travis/build/peastone/WIFIOff/src/WIFIOff.ino](#), 5

[/home/travis/build/peastone/WIFIOff/src/wifionoff_version.ino](#), 53

[arduinoOtaPassword](#)
WIFIOff.ino, 50

[buttonLastPressed](#)
WIFIOff.ino, 50

[buttonPin](#)
WIFIOff.ino, 50

[calculate_crc](#)
WIFIOff.ino, 15

[callbackConfigMode](#)
WIFIOff.ino, 15

[callbackConfigSuccessful](#)
WIFIOff.ino, 16

[changeWiFiCredentials](#)
WIFIOff.ino, 17

[checkCRC](#)
WIFIOff.ino, 17

[checkEEPROMVersionNumber](#)
WIFIOff.ino, 18

[checkMQTTConnected](#)
WIFIOff.ino, 18

[checkWiFiConnected](#)
WIFIOff.ino, 19

[configureMQTT](#)
WIFIOff.ino, 19

[configureOutputs](#)
WIFIOff.ino, 20

[configureWebServer](#)
WIFIOff.ino, 21

[connectRelay](#)
WIFIOff.ino, 21

[credentialPassword](#)
WIFIOff.ino, 50

[credentialSsid](#)
WIFIOff.ino, 50

[CTR_MAX_TRIES_WIFI_CONNECTION](#)
WIFIOff.ino, 11

[DEV_OTA_UPDATES](#)
WIFIOff.ino, 11

[disconnectRelay](#)
WIFIOff.ino, 22

[EEPROM_address_CRC](#)
WIFIOff.ino, 11

[EEPROM_address_MQTT_server](#)
WIFIOff.ino, 12

[EEPROM_address_MQTT_server_configured](#)
WIFIOff.ino, 12

[EEPROM_address_OTA_password](#)
WIFIOff.ino, 12

[EEPROM_address_start](#)
WIFIOff.ino, 12

[EEPROM_enabled](#)
WIFIOff.ino, 12

[EEPROM_enabled_size](#)
WIFIOff.ino, 12

[EEPROM_init_value](#)
WIFIOff.ino, 12

[EEPROM_length](#)
WIFIOff.ino, 12

[EEPROM_size_CRC](#)
WIFIOff.ino, 13

[EEPROM_size_MQTT_server](#)
WIFIOff.ino, 13

[EEPROM_size_OTA_password](#)
WIFIOff.ino, 13

[EEPROM_version_number](#)
WIFIOff.ino, 13

[EEPROM_version_number_address](#)
WIFIOff.ino, 13

[factoryResetRequested](#)
WIFIOff.ino, 50

[feedbackQuickBlink](#)
WIFIOff.ino, 23

[getClientID](#)
WIFIOff.ino, 23

[getDefaultOtaPassword](#)
WIFIOff.ino, 24

[getFactoryResetRequested](#)
WIFIOff.ino, 24

[getMQTTOutgoingTopic](#)
WIFIOff.ino, 24

[getMQTTServer](#)
WIFIOff.ino, 24

[getStateLEDOn](#)
WIFIOff.ino, 25

[getStateMQTTConfigured](#)
WIFIOff.ino, 25

[getStateRelayConnected](#)
WIFIOff.ino, 26

[getUserActionFeedbackRequest](#)

- WIFIOff.ino, [26](#)
- getWiFiCredentialChangeRequested
 - WIFIOff.ino, [27](#)
- HTTP_PORT
 - WIFIOff.ino, [14](#)
- initEEPROM
 - WIFIOff.ino, [27](#)
- isNotWhitelisted
 - WIFIOff.ino, [28](#)
- loop
 - WIFIOff.ino, [28](#)
- MQTT_PORT
 - WIFIOff.ino, [14](#)
- mqttClient
 - WIFIOff.ino, [50](#)
- mqttConnect
 - WIFIOff.ino, [29](#)
- mqttControlRelay
 - WIFIOff.ino, [30](#)
- mqttIncomingTopic
 - WIFIOff.ino, [50](#)
- mqttOutgoingTopic
 - WIFIOff.ino, [51](#)
- mqttPublish
 - WIFIOff.ino, [31](#)
- mqttServer
 - WIFIOff.ino, [51](#)
- mqttServerValid
 - WIFIOff.ino, [32](#)
- NUMERIC_RESPONSE
 - WIFIOff.ino, [14](#)
- OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED
 - WIFIOff.ino, [14](#)
- paramOtaPassword
 - WIFIOff.ino, [33](#)
- performFactoryReset
 - WIFIOff.ino, [33](#)
- pinLED
 - WIFIOff.ino, [51](#)
- pinRelay
 - WIFIOff.ino, [51](#)
- pressHandler
 - WIFIOff.ino, [34](#)
- reboot
 - WIFIOff.ino, [34](#)
- renderFactoryReset
 - WIFIOff.ino, [35](#)
- renderFooter
 - WIFIOff.ino, [35](#)
- renderHeader
 - WIFIOff.ino, [35](#)
- renderInfo
 - WIFIOff.ino, [36](#)
- renderMQTTSettings
 - WIFIOff.ino, [36](#)
- renderOtaPasswordChangeForm
 - WIFIOff.ino, [37](#)
- renderRelay
 - WIFIOff.ino, [37](#)
- renderWiFiCredentialChange
 - WIFIOff.ino, [38](#)
- REPOSITORY_URL_STRING
 - WIFIOff.ino, [51](#)
- restoreMQTTConfigurationFromEEPROM
 - WIFIOff.ino, [38](#)
- restoreOtaPasswordFromEEPROM
 - WIFIOff.ino, [39](#)
- retrieveCRC
 - WIFIOff.ino, [39](#)
- saveMQTTConfigurationToEEPROM
 - WIFIOff.ino, [39](#)
- saveOtaPasswordToEEPROM
 - WIFIOff.ino, [40](#)
- selectionState
 - WIFIOff.ino, [51](#)
- SERIAL_PRINTING
 - WIFIOff.ino, [14](#)
- setFactoryResetRequested
 - WIFIOff.ino, [41](#)
- setMQTTIncomingTopic
 - WIFIOff.ino, [41](#)
- setMQTTOutgoingTopic
 - WIFIOff.ino, [41](#)
- setMQTTServer
 - WIFIOff.ino, [42](#)
- setStateMQTTConfigured
 - WIFIOff.ino, [42](#)
- setup
 - WIFIOff.ino, [43](#)
- setupEEPROM
 - WIFIOff.ino, [44](#)
- setupMDNS
 - WIFIOff.ino, [45](#)
- setUserActionFeedbackRequest
 - WIFIOff.ino, [45](#)
- setWiFiCredentialChangeRequested
 - WIFIOff.ino, [46](#)
- stateLEDOn
 - WIFIOff.ino, [51](#)
- stateMQTTConfigured
 - WIFIOff.ino, [52](#)
- stateRelayConnected
 - WIFIOff.ino, [52](#)
- storeCRC
 - WIFIOff.ino, [46](#)
- storeEEPROMVersionNumber
 - WIFIOff.ino, [47](#)
- switchOffLED
 - WIFIOff.ino, [47](#)
- switchOnLED
 - WIFIOff.ino, [47](#)

- WIFIONOff.ino, [48](#)
- ticker
 - WIFIONOff.ino, [52](#)
- TIME_EEPROM_WARNING
 - WIFIONOff.ino, [14](#)
- TIME_HALF_A_SECOND
 - WIFIONOff.ino, [14](#)
- TIME_QUARTER_SECOND
 - WIFIONOff.ino, [15](#)
- timeSincebuttonPressed
 - WIFIONOff.ino, [52](#)
- toggleLED
 - WIFIONOff.ino, [48](#)
- toggleRelay
 - WIFIONOff.ino, [49](#)
- TRIGGER_TIME_FACTORY_RESET
 - WIFIONOff.ino, [15](#)
- unsetUserActionFeedbackRequest
 - WIFIONOff.ino, [49](#)
- userActionFeedbackRequest
 - WIFIONOff.ino, [52](#)
- VERSION
 - WIFIONOff.ino, [52](#)
- VERSION_BY_GIT_DESCRIBE
 - wifionoff_version.h, [54](#)
- webserver
 - WIFIONOff.ino, [52](#)
- wifiClient
 - WIFIONOff.ino, [53](#)
- wifiCredentialChangeRequested
 - WIFIONOff.ino, [53](#)
- wifiManager
 - WIFIONOff.ino, [53](#)
- WIFIONOff.ino
 - arduinoOtaPassword, [50](#)
 - buttonLastPressed, [50](#)
 - buttonPin, [50](#)
 - calculate_crc, [15](#)
 - callbackConfigMode, [15](#)
 - callbackConfigSuccessful, [16](#)
 - changeWiFiCredentials, [17](#)
 - checkCRC, [17](#)
 - checkEEPROMVersionNumber, [18](#)
 - checkMQTTConnected, [18](#)
 - checkWiFiConnected, [19](#)
 - configureMQTT, [19](#)
 - configureOutputs, [20](#)
 - configureWebServer, [21](#)
 - connectRelay, [21](#)
 - credentialPassword, [50](#)
 - credentialSsid, [50](#)
 - CTR_MAX_TRIES_WIFI_CONNECTION, [11](#)
 - DEV_OTA_UPDATES, [11](#)
 - disconnectRelay, [22](#)
 - EEPROM_address_CRC, [11](#)
 - EEPROM_address_MQTT_server, [12](#)
 - EEPROM_address_MQTT_server_configured, [12](#)
 - EEPROM_address_OTA_password, [12](#)
 - EEPROM_address_start, [12](#)
 - EEPROM_enabled, [12](#)
 - EEPROM_enabled_size, [12](#)
 - EEPROM_init_value, [12](#)
 - EEPROM_length, [12](#)
 - EEPROM_size_CRC, [13](#)
 - EEPROM_size_MQTT_server, [13](#)
 - EEPROM_size_OTA_password, [13](#)
 - EEPROM_version_number, [13](#)
 - EEPROM_version_number_address, [13](#)
 - factoryResetRequested, [50](#)
 - feedbackQuickBlink, [23](#)
 - getClientID, [23](#)
 - getDefaultOtaPassword, [24](#)
 - getFactoryResetRequested, [24](#)
 - getMQTTOutgoingTopic, [24](#)
 - getMQTTServer, [24](#)
 - getStateLEDOn, [25](#)
 - getStateMQTTConfigured, [25](#)
 - getStateRelayConnected, [26](#)
 - getUserActionFeedbackRequest, [26](#)
 - getWiFiCredentialChangeRequested, [27](#)
 - HTTP_PORT, [14](#)
 - initEEPROM, [27](#)
 - isNotWhitelisted, [28](#)
 - loop, [28](#)
 - MQTT_PORT, [14](#)
 - mqttClient, [50](#)
 - mqttConnect, [29](#)
 - mqttControlRelay, [30](#)
 - mqttIncomingTopic, [50](#)
 - mqttOutgoingTopic, [51](#)
 - mqttPublish, [31](#)
 - mqttServer, [51](#)
 - mqttServerValid, [32](#)
 - NUMERIC_RESPONSE, [14](#)
 - OUTPUT_RELAY_STATE_ON_GREEN_STATUS_LED, [14](#)
 - paramOtaPassword, [33](#)
 - performFactoryReset, [33](#)
 - pinLED, [51](#)
 - pinRelay, [51](#)
 - pressHandler, [34](#)
 - reboot, [34](#)
 - renderFactoryReset, [35](#)
 - renderFooter, [35](#)
 - renderHeader, [35](#)
 - renderInfo, [36](#)
 - renderMQTTSettings, [36](#)
 - renderOtaPasswordChangeForm, [37](#)
 - renderRelay, [37](#)
 - renderWiFiCredentialChange, [38](#)
 - REPOSITORY_URL_STRING, [51](#)
 - restoreMQTTConfigurationFromEEPROM, [38](#)
 - restoreOtaPasswordFromEEPROM, [39](#)

- retrieveCRC, [39](#)
- saveMQTTConfigurationToEEPROM, [39](#)
- saveOtaPasswordToEEPROM, [40](#)
- selectionState, [51](#)
- SERIAL_PRINTING, [14](#)
- setFactoryResetRequested, [41](#)
- setMQTTIncomingTopic, [41](#)
- setMQTTOutgoingTopic, [41](#)
- setMQTTServer, [42](#)
- setStateMQTTConfigured, [42](#)
- setup, [43](#)
- setupEEPROM, [44](#)
- setupMDNS, [45](#)
- setUserActionFeedbackRequest, [45](#)
- setWiFiCredentialChangeRequested, [46](#)
- stateLEDOn, [51](#)
- stateMQTTConfigured, [52](#)
- stateRelayConnected, [52](#)
- storeCRC, [46](#)
- storeEEPROMVersionNumber, [47](#)
- switchOffLED, [47](#)
- switchOnLED, [48](#)
- ticker, [52](#)
- TIME_EEPROM_WARNING, [14](#)
- TIME_HALF_A_SECOND, [14](#)
- TIME_QUARTER_SECOND, [15](#)
- timeSincebuttonPressed, [52](#)
- toggleLED, [48](#)
- toggleRelay, [49](#)
- TRIGGER_TIME_FACTORY_RESET, [15](#)
- unsetUserActionFeedbackRequest, [49](#)
- userActionFeedbackRequest, [52](#)
- VERSION, [52](#)
- webserver, [52](#)
- wifiClient, [53](#)
- wifiCredentialChangeRequested, [53](#)
- wifiManager, [53](#)
- wifionoff_version.h
 - VERSION_BY_GIT_DESCRIBE, [54](#)