WifiOnOff

Generated by Doxygen 1.8.16

Chapter 1

WIFIOnOff

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.

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!

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.

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. Press the button until it flashes the first time. Press the button of the router immediately after that. The WPS procedure starts.
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.

Last advice

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.

Also thanks to

- Jeroen de Bruijn for his gist on how to auto-deploy Doxygen documentation on Github pages with Travis CI

Chapter 2

Use Case Analysis

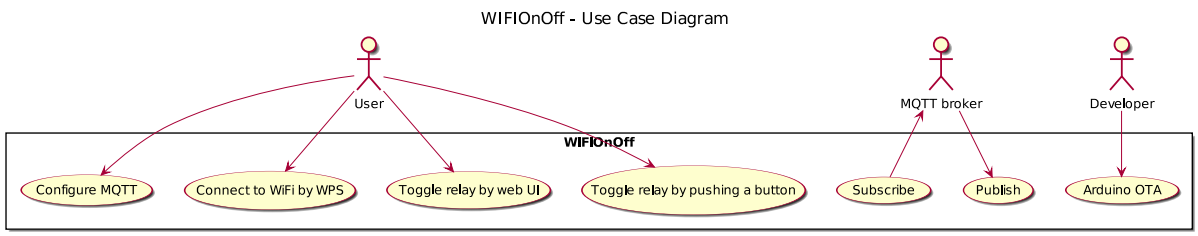
Features Implemented

- WPS Push Button Configuration: I really ask myself, why WiFi Protected Setup is used so infrequently. It is convenient and you just need to push a button on your router and on your device. Many routers also offer the opportunity to press the push button over the web interface. So, you don't even have to walk. This is a must-have feature.
- HTTP interface: The user must be able to control the WIFIOnOff with a simple web interface. You don't want to walk to switch the caffee machine on or off. This is a must-have feature. It is also needed for MQTT support.
- MQTT support: If configured over the HTTP interface, the WIFIOnOff publishes the state of the relay and subscribes to a command channel. This is a nice-to-have feature. It is very interesting to experiment with MQTT to produce internet-of-things-like networks. In future, this feature might become more important.
- Arduino OTA: With over-the-air updates enabled, the WIFIOnOff can be flashed wirelessly. This is a nice-to-have feature. One can still flash over serial wire, but OTA is so much more convenient. This feature is NOT recommeded to be used in an untrusted environment.

Features Not to Be Implemented

- Cloud: The WIFIOnOff does not need a cloud or an external internet connection to be useful. This is quite nice as your device does not become worthless, if the your cloud service provider discontinues service. It also offers some advantage in terms of data protection. You are still free to use external service providers.
- Timer: I tried to follow the UNIX philosophy when designing this program. The WIFIOnOff does one thing well: toggling the relay. It can be connected with other devices by MQTT. If you want to implement a timer, you can do this, as a script on a separate computer connected to the MQTT broker. You also get a GUI for free with an MQTT smartphone app.

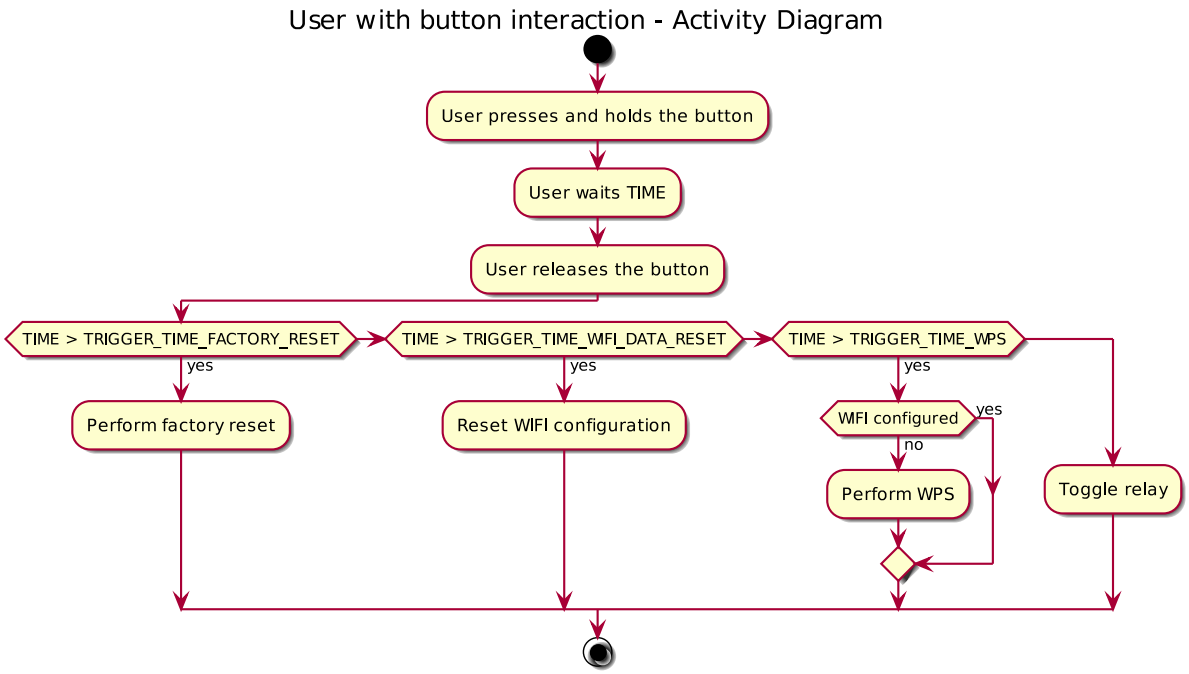
Use Case Diagram



Boundary Conditions

- XSS protection: The HTTP web interface accepts user input which must be sanitized. The approach that is used in this project is whitelisting. The goal is to prevent XSS attacks. For more information about XSS, look here.

Button interaction



Chapter 3

Todo List

Member DEV_OTA_PASSWD

Despite the fact that over-the-air (OTA) updates are not highly secure, it is highly recommended to change the password. This is just a very bad default.

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

File Documentation

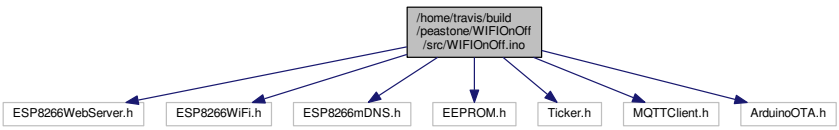
5.1 /home/travis/build/peastone/WIFIOnOff/README.md File Reference

5.2 /home/travis/build/peastone/WIFIOnOff/RequirementsAndDesign.md File Reference

5.3 /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 <ArduinoOTA.h>
```

Include dependency graph for WIFIOnOff.ino:



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_STAT↩ E_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 DEV_OTA_PASSWD "CwpvVzR33gKY"

  Define the password to protect OTA with DEV_OTA_PASSWD. This is not a high security procedure.
- #define TRIGGER_TIME_WPS 7000

  time in ms to wait until WPS request is triggered when pressing the button (see pressHandler())
- #define TRIGGER_TIME_WIFI_DATA_RESET 30000

  time in ms to wait until the WIFI configuration reset
- #define TRIGGER_TIME_FACTORY_RESET 60000

  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 amout of bytes to be stored is 4096.
- #define EEPROM_version_number 0

  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 adresses.
- #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_WPS_configured EEPROM_address_start

  WPS is the only choice to connect to WIFI. This shall improve usability. The state of WPS is stored in EEPROM at this address. The value 0x00 was chosen as it terminates Strings, which leads to perfect default data.
- #define EEPROM_address_MQTT_server_configured (EEPROM_address_WPS_configured + EEPROM_enabled_size)

  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.

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 feedbackEEPROMInit ()

  This function is called to give the user feedback that the EEPROM has been re-/initialized. This means that all the user-entered data is gone and the device needs to be reconfigured.
- 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.
- void feedbackWIFIisConnecting ()

  This function is called to give the user feedback that WIFI is about to connect. It will blink at a medium rate.
- 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.

- bool checkWiFiConfigured ()

  This function checks whether WPS has already been performed once. In this case, one can directly connect to WiFi.

- void setWiFiConfigured ()

  This function is a setter function which sets WiFi to configured, which means that WPS has already been performed successfully. This function is only called in performWPS(). The configuration is saved to EEPROM.

- void unsetWiFiConfigured ()

  This function is a setter function which sets WiFi to not configured, which means that WPS is necessary before connecting to WIFI. The configuration is saved to EEPROM.

- void performWPS ()

  This function is used to trigger WPS. If WPS has been successful, setWiFiConfigured() is executed and unsetWPSRequest() is called.

- String getClientID ()

  This function returns the client name used for MQTT, see mqttConnect().

- bool connectToWiFi ()

  This function tries to connect to WIFI.

- void setupMDNS ()

  This function sets up MDNS.

- bool checkWiFiConnected ()

  This function checks whether WIFI is connected.

- bool getWPSRequest ()

  Getter function for wpsRequested.

- void unsetWPSRequest ()

  Setter function to unset wpsRequested.

- void setWPSRequest ()

  Setter function to set wpsRequested.

- bool getWifiResetRequested ()

  Getter function for wifiResetRequested.

- void setWifiResetRequested ()

  Setter function to set wifiResetRequested.

- void unsetWifiResetRequested ()

  Setter function to unsset wifiResetRequested.

- 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 renderMQTTServerSettings (String storedServerName, String failureMsg, bool stateMQTTActivated, bool stateMQTTConnected)

  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.
- bool getFactoryResetRequested ()

  Getter function for factoryResetRequested.
- void setFactoryResetRequested ()

  Setter function to set factoryResetRequested.
- void unsetFactoryResetRequested ()

  Setter function to unset factoryResetRequested.
- void pressHandler ()

  This function triggers.
- 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.

Variables

- 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().
- 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.
- bool wpsRequested = false

  State to track whether the WPS has been requested.
- bool wifiResetRequested = false

  State to track whether the deletion of WIFI data 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.
- 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.

5.3.1 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 WPS for reasons of a better user experience. This device can be used in the local network without any dependency of a cloud.

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

5.3.2 Macro Definition Documentation

5.3.2.1 CTR_MAX_TRIES_WIFI_CONNECTION

```
#define CTR_MAX_TRIES_WIFI_CONNECTION 60
```

maximum amount of tries to connect to WIFI

5.3.2.2 DEV_OTA_PASSWD

```
#define DEV_OTA_PASSWD "CwpvVzR33gKY"
```

Define the password to protect OTA with DEV_OTA_PASSWD. This is not a high security procedure.

Todo Despite the fact that over-the-air (OTA) updates are not highly secure, it is highly recommended to change the password. This is just a very bad default.

See also

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

5.3.2.3 DEV_OTA_UPDATES

```
#define DEV_OTA_UPDATES
```

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

5.3.2.4 EEPROM_address_CRC

```
#define EEPROM_address_CRC 1
```

The EEPROM version number is stored at this address.

5.3.2.5 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.

5.3.2.6 EEPROM_address_MQTT_server_configured

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

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

5.3.2.7 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.

5.3.2.8 EEPROM_address_WPS_configured

```
#define EEPROM_address_WPS_configured EEPROM_address_start
```

WPS is the only choice to connect to WIFI. This shall improve usability. The state of WPS is stored in EEPROM at this address. The value 0x00 was chosen as it terminates Strings, which leads to perfect default data.

5.3.2.9 EEPROM_enabled

```
#define EEPROM_enabled 0xEB
```

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

5.3.2.10 EEPROM_enabled_size

```
#define EEPROM_enabled_size 1
```

The amount of storage needed by EEPROM_enabled in EEPROM.

5.3.2.11 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().

5.3.2.12 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 amout of bytes to be stored is 4096.

See also

https://git.io/vxYHO
https://git.io/vxYHG

5.3.2.13 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.

5.3.2.14 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

5.3.2.15 EEPROM_version_number

```
#define EEPROM_version_number 0
```

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.

5.3.2.16 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 adresses.

5.3.2.17 HTTP_PORT

```
#define HTTP_PORT 80
```

standard port used for the HTTP protocol

5.3.2.18 MQTT_PORT

```
#define MQTT_PORT 1883
```

standard port used for the MQTT protocol

5.3.2.19 NUMERIC_RESPONSE

```
#define NUMERIC_RESPONSE
```

Define NUMERIC_RESPONSE if you want the mqttClient to publish "1" or "0" instead of "on" or "off".

5.3.2.20 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_STA↩ TE_ON_GREEN_STATUS_LED, if you want the state of the relay being visible also on the green LED (connected == LED shines).

5.3.2.21 SERIAL_PRINTING

```
#define SERIAL_PRINTING
```

Define SERIAL_PRINTING if you want to enable serial communication.

5.3.2.22 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)

5.3.2.23 TIME_HALF_A_SECOND

```
#define TIME_HALF_A_SECOND 500
```

definition of the timespan of half a second

5.3.2.24 TIME_QUARTER_SECOND

```
#define TIME_QUARTER_SECOND 250
```

definition of the timespan of a quarter of a second

5.3.2.25 TRIGGER_TIME_FACTORY_RESET

```
#define TRIGGER_TIME_FACTORY_RESET 60000
```

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

5.3.2.26 TRIGGER_TIME_WIFI_DATA_RESET

```
#define TRIGGER_TIME_WIFI_DATA_RESET 30000
```

time in ms to wait until the WIFI configuration reset

5.3.2.27 TRIGGER_TIME_WPS

```
#define TRIGGER_TIME_WPS 7000
```

time in ms to wait until WPS request is triggered when pressing the button (see pressHandler())

5.3.3 Function Documentation

5.3.3.1 calculate_crc()

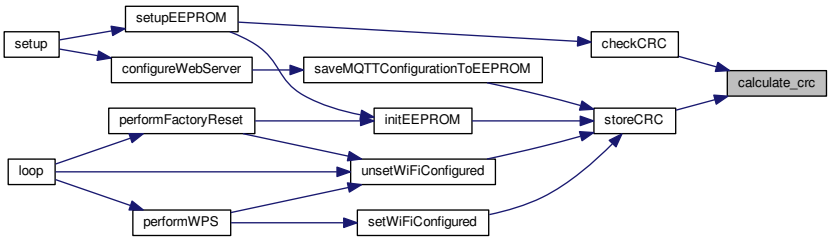
```
unsigned long calculate_crc ( )
```

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:



5.3.3.2 checkCRC()

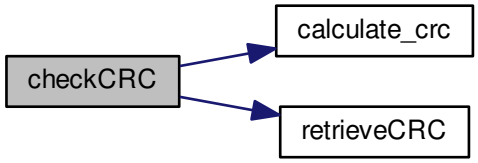
```
bool checkCRC ( )
```

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:

5.3.3.3 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:

5.3.3.4 checkMQTTConnected()

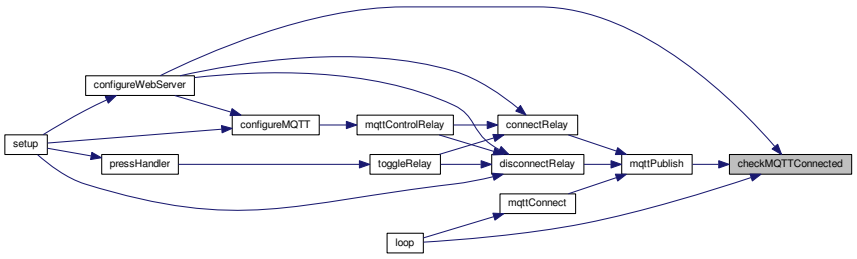
```
bool checkMQTTConnected ( )
```

Get MQTT connection state.

Returns

true, if MQTT is connected, false otherwise

Here is the caller graph for this function:



5.3.3.5 checkWiFiConfigured()

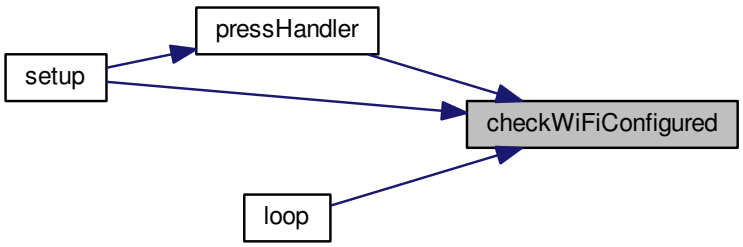
```
bool checkWiFiConfigured ( )
```

This function checks whether WPS has already been performed once. In this case, one can directly connect to WiFi.

Returns

true if WPS has already been performed, false otherwise

Here is the caller graph for this function:



5.3.3.6 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:

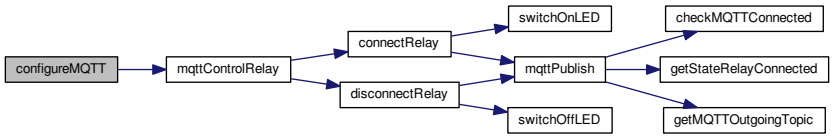
5.3.3.7 configureMQTT()

```
void configureMQTT ( )
```

Configure new MQTT server / broker. This will.

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

Here is the call graph for this function:



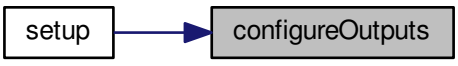
Here is the caller graph for this function:

5.3.3.8 configureOutputs()

```
void configureOutputs ( )
```

This function configures the output pins of the microcontroller.

Here is the caller graph for this function:



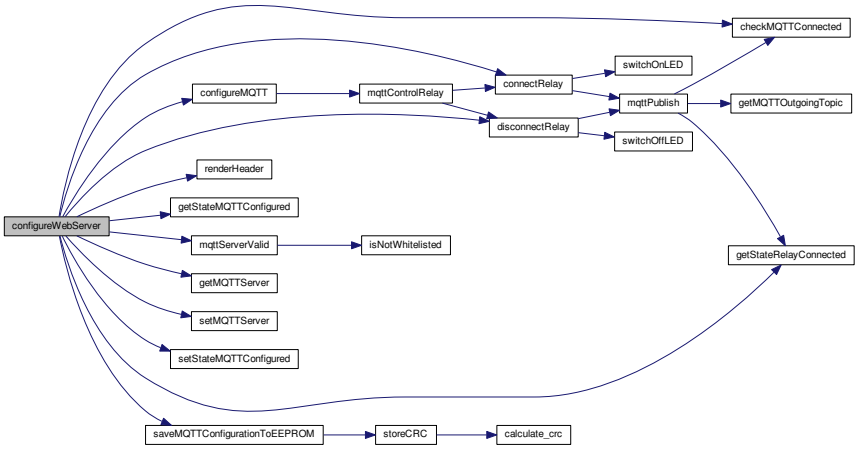
5.3.3.9 configureWebServer()

```
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:



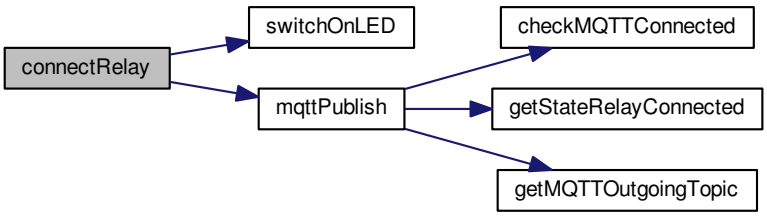
Here is the caller graph for this function:

5.3.3.10 connectRelay()

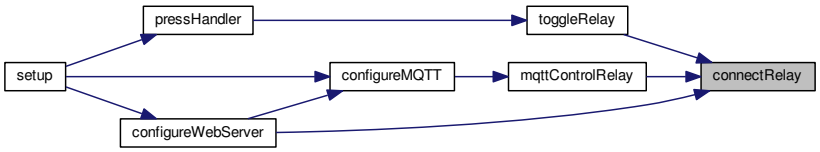
```
void connectRelay ( )
```

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:



5.3.3.11 connectToWiFi()

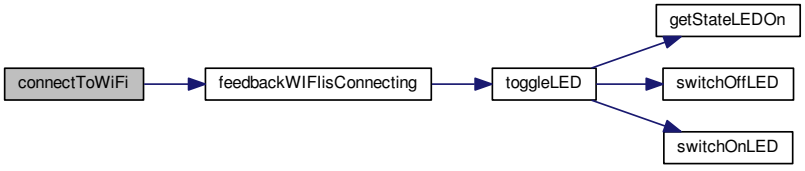
```
bool connectToWiFi ( )
```

This function tries to connect to WIFI.

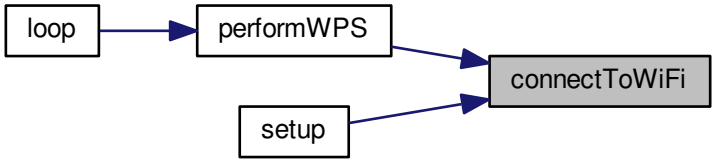
Returns

true, if the connection was successful, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:

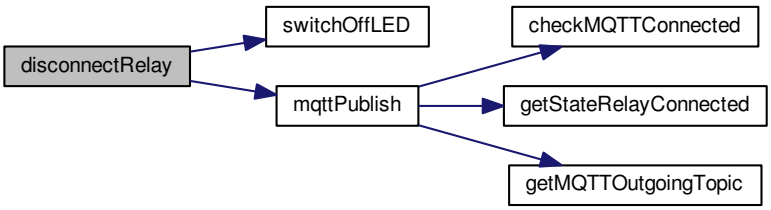


5.3.3.12 disconnectRelay()

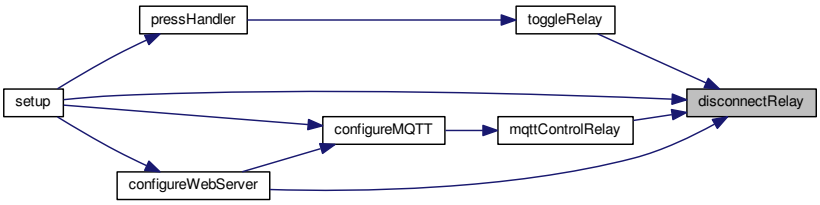
```
void disconnectRelay ( )
```

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:

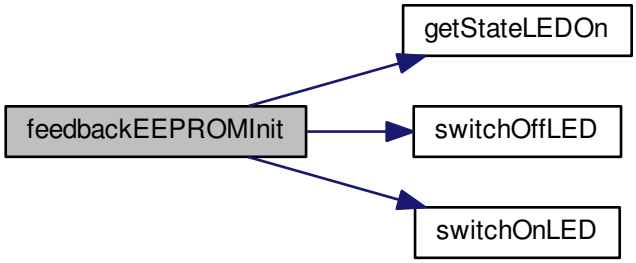


5.3.3.13 feedbackEEPROMInit()

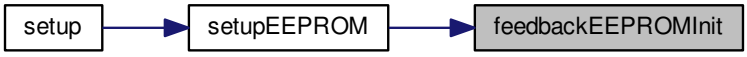
```
void feedbackEEPROMInit ( )
```

This function is called to give the user feedback that the EEPROM has been re-/initialized. This means that all the user-entered data is gone and the device needs to be reconfigured.

Here is the call graph for this function:



Here is the caller graph for this function:

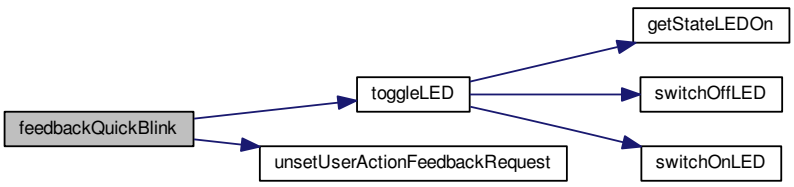


5.3.3.14 feedbackQuickBlink()

```
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:



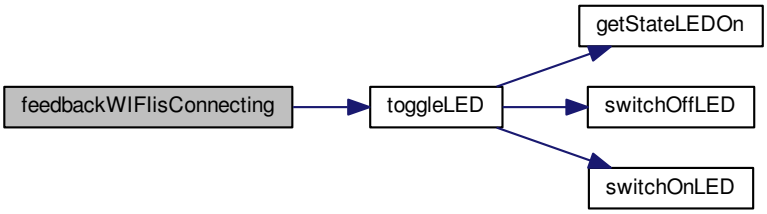
Here is the caller graph for this function:

5.3.3.15 feedbackWIFIisConnecting()

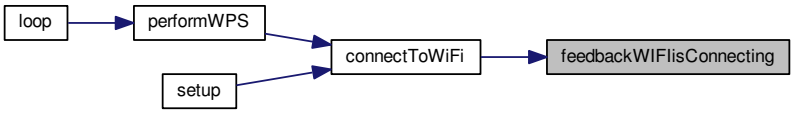
```
void feedbackWIFIisConnecting ( )
```

This function is called to give the user feedback that WIFI is about to connect. It will blink at a medium rate.

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.16 getClientID()

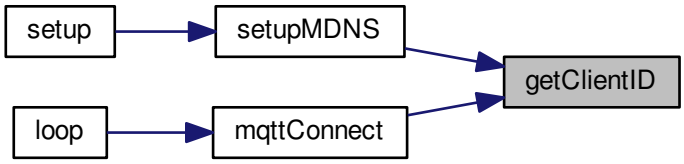
```
String getClientID ( )
```

This function returns the client name used for MQTT, see mqttConnect().

Returns

the client ID: "WIFIOnOff" + MAC address

Here is the caller graph for this function:



5.3.3.17 getFactoryResetRequested()

```
bool getFactoryResetRequested ( )
```

Getter function for factoryResetRequested.

Returns

true, if factory reset is requested, false otherwise.

5.3.3.18 getMQTTOutgoingTopic()

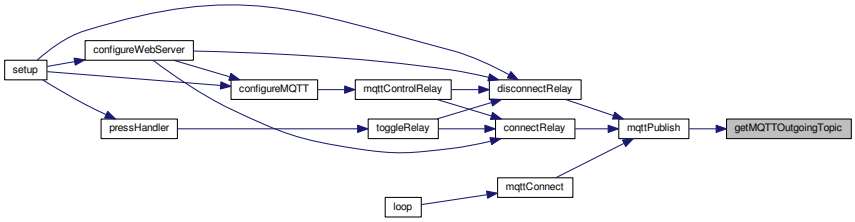
```
String getMQTTOutgoingTopic ( )
```

Getter function for mqttOutgoingTopic.

Returns

String of outgoing MQTT topic

Here is the caller graph for this function:



5.3.3.19 getMQTTServer()

```
String getMQTTServer ( )
```

Getter function for MQTT server / broker mqttServer.

Returns

the latest MQTT server / broker

Here is the caller graph for this function:

5.3.3.20 getStateLEDOn()

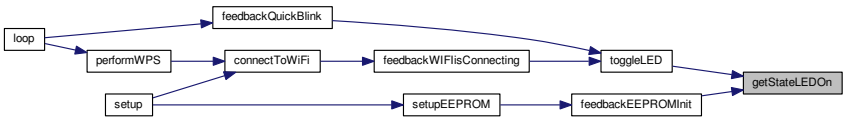
```
bool getStateLEDOn ( )
```

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:



5.3.3.21 getStateMQTTConfigured()

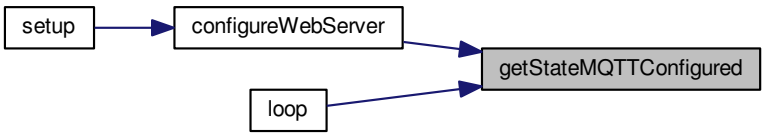
```
bool getStateMQTTConfigured ( )
```

Getter function for stateMQTTConfigured.

Returns

true if MQTT is configured, false otherwise

Here is the caller graph for this function:



5.3.3.22 getStateRelayConnected()

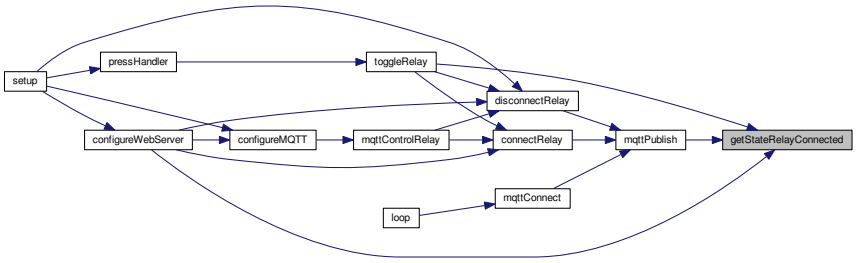
```
bool getStateRelayConnected ( )
```

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:



5.3.3.23 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:

5.3.3.24 getWifiResetRequested()

```
bool getWifiResetRequested ( )
```

Getter function for wifiResetRequested.

Returns

true, if a deletion of WIFI data has been requested, false otherwise.

5.3.3.25 getWPSRequest()

```
bool getWPSRequest ( )
```

Getter function for wpsRequested.

Returns

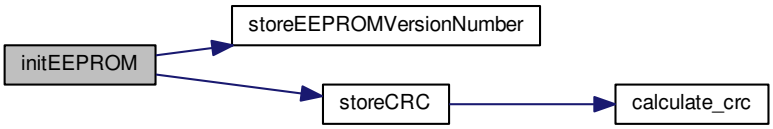
true, if WPS has been requested, false otherwise.

5.3.3.26 initEEPROM()

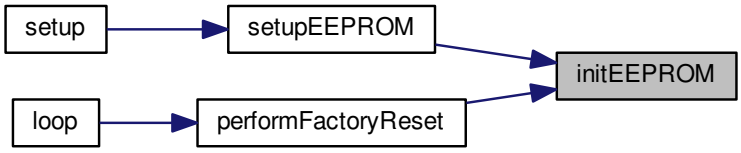
```
void initEEPROM ( )
```

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:



5.3.3.27 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 hexadeximal numbers, double dots or brackets (0..9 | a..f | A..F | [|])
- DNS names consist of letters, digits and hypen (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

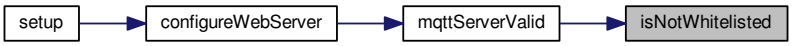
See also

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

Returns

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

Here is the caller graph for this function:



5.3.3.28 loop()

```
void loop (
            void  )
```

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

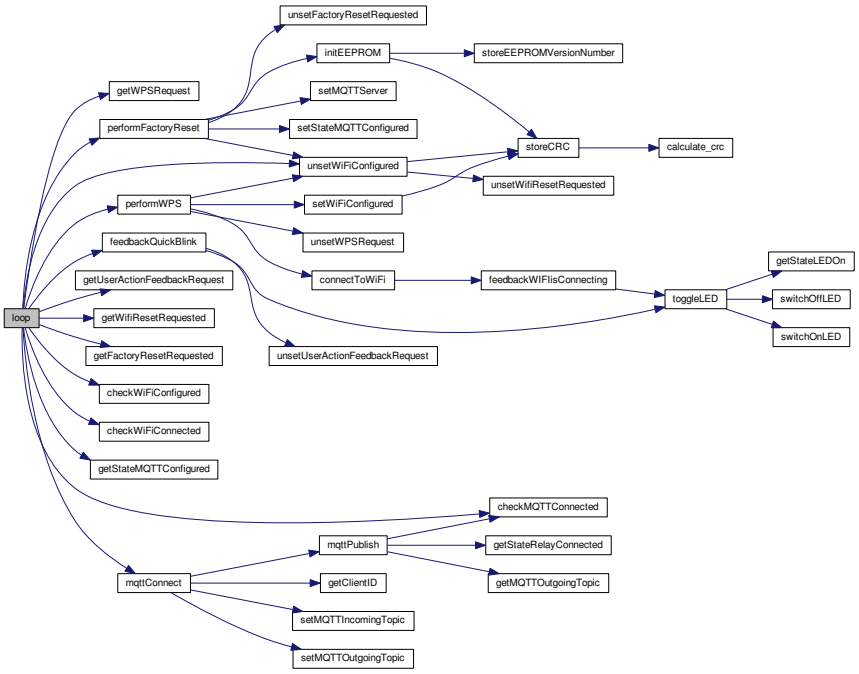
```
At first it checks for incoming user requests:
- to perform WPS (performWPS())
- to delete the WiFi configuration (unsetWiFiConfigured())
- to perform a factory reset (performFactoryReset())

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 WPS has been
performed and that the WiFi is connected.
If WPS has not been done or 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:



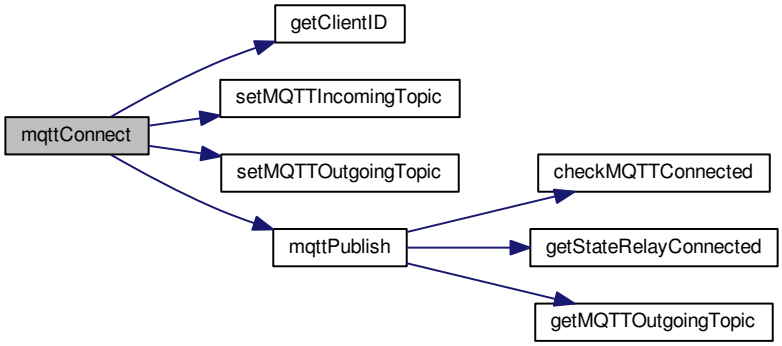
5.3.3.29 mqttConnect()

```
void mqttConnect ( )
```

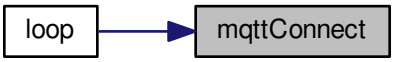
This function connects to broker and.

- 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:



5.3.3.30 mqttControlRelay()

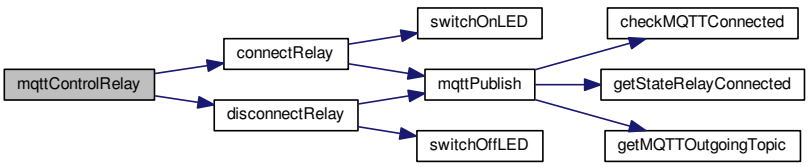
```
void mqttControlRelay (
            String & topic,
            String & payload )
```

Callback which is called when MQTT receives an incoming topic.

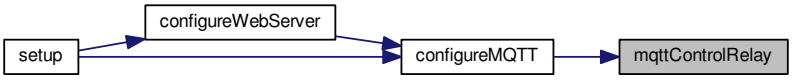
Parameters

in	*topic*	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	*payload*	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:

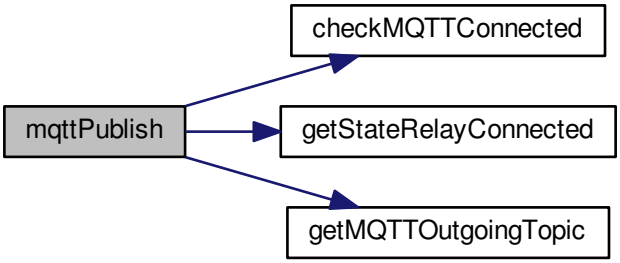


5.3.3.31 mqttPublish()

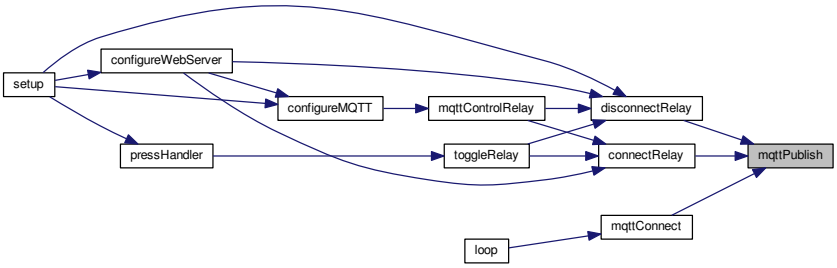
```
void mqttPublish ( )
```

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:



5.3.3.32 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	*mqttServer*	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:

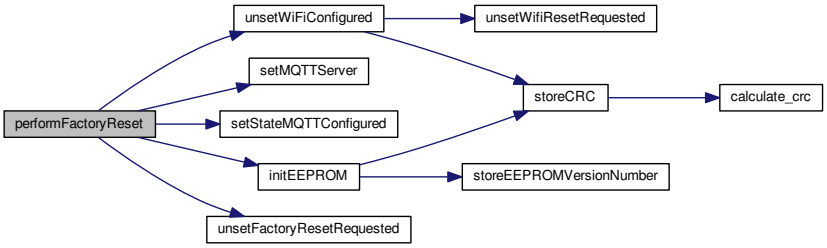
5.3.3.33 performFactoryReset()

```
void performFactoryReset ( )
```

Perform factory reset.

- reset WiFi
- reset MQTT
- reset EEPROM

Here is the call graph for this function:



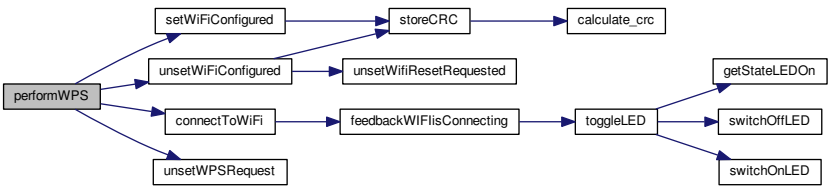
Here is the caller graph for this function:

5.3.3.34 performWPS()

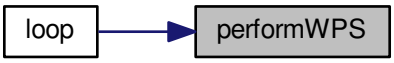
```
void performWPS ( )
```

This function is used to trigger WPS. If WPS has been successful, setWiFiConfigured() is executed and unsetWPSRequest() is called.

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.35 pressHandler()

```
void pressHandler ( )
```

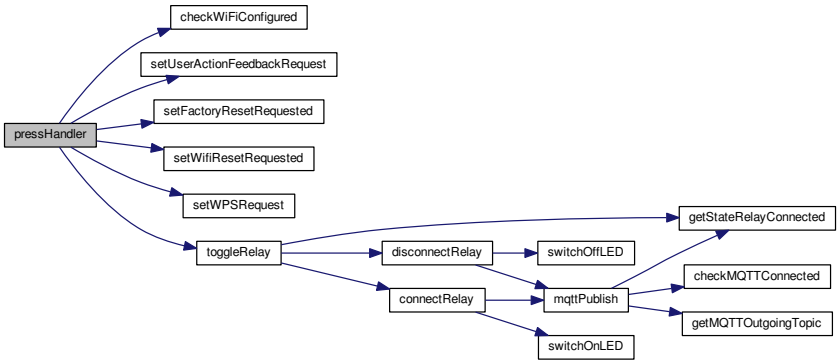
This function triggers.

- the toggling of the relay
- WPS button method request
- factory reset request. As this function is called regularly with ticker, it should be quick, otherwise the ESP8266 might crash.

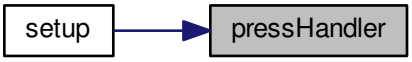
See also

https://arduino-esp8266.readthedocs.io/en/latest/faq/a02-my-esp-crashes.html

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.36 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.

5.3.3.37 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:

5.3.3.38 renderMQTTServerSettings()

```
String renderMQTTServerSettings (
            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	*storedServerName*	DNS name or IP address which is displayed on the HTML site
in	*failureMsg*	message that is displayed to the user in case of success or failure
in	*stateMQTTActivated*	shows the user, whether MQTT is activated
in	*stateMQTTConnected*	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

5.3.3.39 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	*stateRelayConnected*	if true, the relay is displayed as on, otherwise as off

Returns

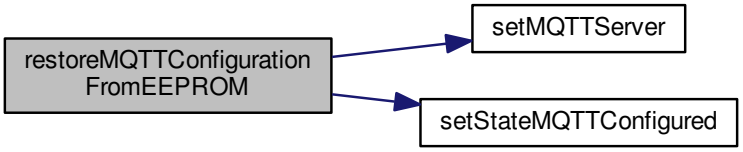
HTML, look inside the function for more information.

5.3.3.40 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:

5.3.3.41 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:

5.3.3.42 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:

5.3.3.43 setFactoryResetRequested()

```
void setFactoryResetRequested ( )
```

Setter function to set factoryResetRequested.

Here is the caller graph for this function:

5.3.3.44 setMQTTIncomingTopic()

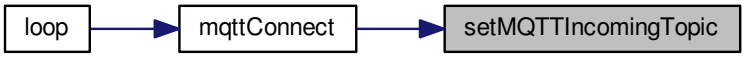
```
void setMQTTIncomingTopic (
            String input )
```

Setter function for mqttIncomingTopic.

Parameters

in	*input*	String to be set

Here is the caller graph for this function:



5.3.3.45 setMQTTOutgoingTopic()

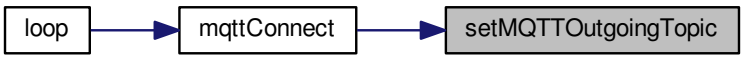
```
void setMQTTOutgoingTopic (
            String input )
```

Setter function for mqttOutgoingTopic.

Parameters

in	*input*	String to be set

Here is the caller graph for this function:



5.3.3.46 setMQTTServer()

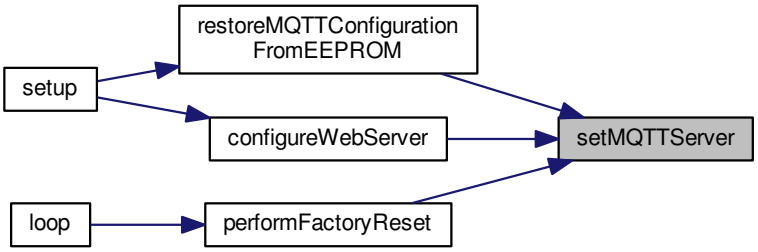
```
void setMQTTServer (
            String input )
```

Setter function for MQTT server / broker mqttServer.

Parameters

in	*input*	MQTT server / broker

Here is the caller graph for this function:



5.3.3.47 setStateMQTTConfigured()

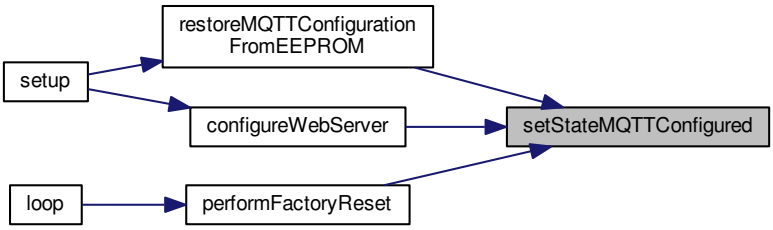
```
void setStateMQTTConfigured (
            bool input )
```

Setter function for stateMQTTConfigured.

Parameters

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

Here is the caller graph for this function:



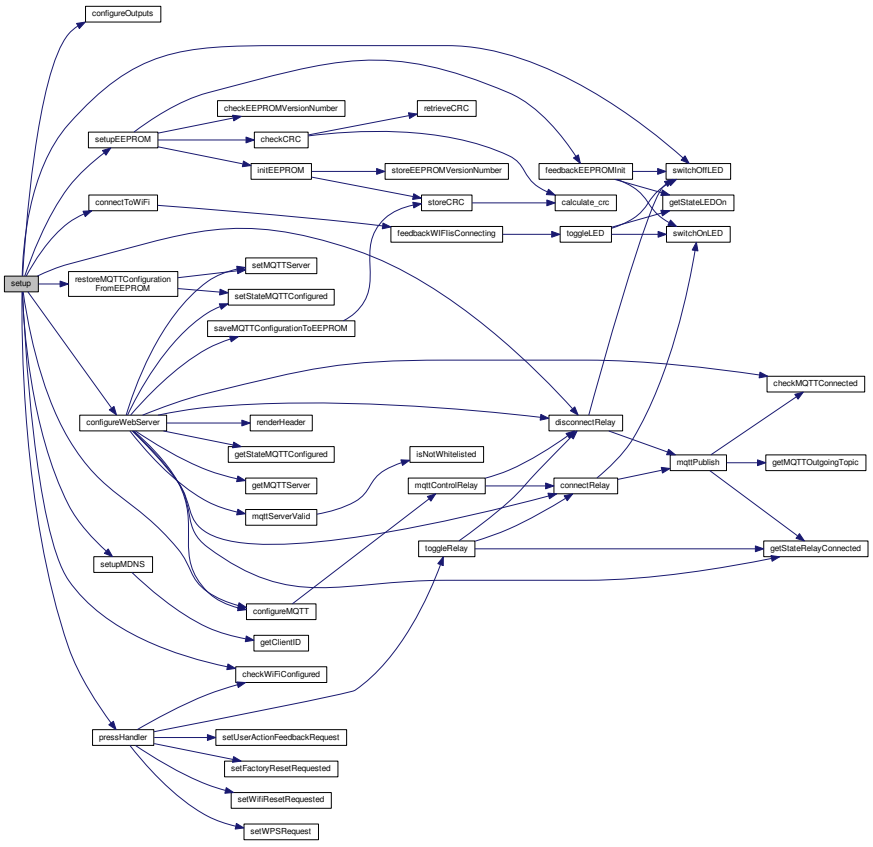
5.3.3.48 setup()

```
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:



5.3.3.49 setupEEPROM()

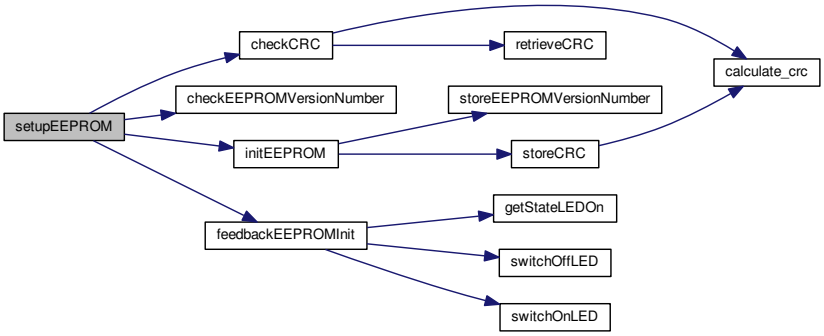
```
void setupEEPROM ( )
```

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.

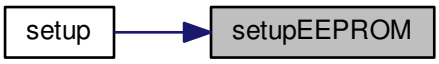
See also

https://git.io/vxOPf

Here is the call graph for this function:



Here is the caller graph for this function:



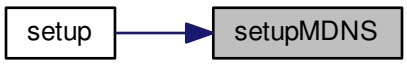
5.3.3.50 setupMDNS()

```
void setupMDNS ( )
```

This function sets up MDNS.

Here is the call graph for this function:

Here is the caller graph for this function:



5.3.3.51 setUserActionFeedbackRequest()

```
void setUserActionFeedbackRequest ( )
```

Setter function for userActionFeedbackRequest. The user should be notified that a menu can be selected.

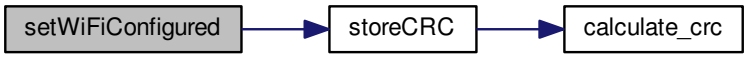
Here is the caller graph for this function:

5.3.3.52 setWiFiConfigured()

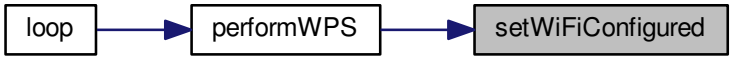
```
void setWiFiConfigured ( )
```

This function is a setter function which sets WiFi to configured, which means that WPS has already been performed successfully. This function is only called in performWPS(). The configuration is saved to EEPROM.

Here is the call graph for this function:



Here is the caller graph for this function:

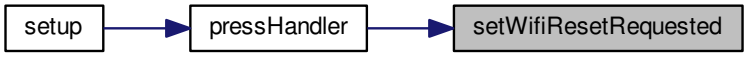


5.3.3.53 setWifiResetRequested()

```
void setWifiResetRequested ( )
```

Setter function to set wifiResetRequested.

Here is the caller graph for this function:

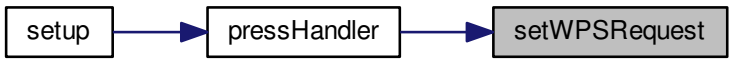


5.3.3.54 setWPSRequest()

```
void setWPSRequest ( )
```

Setter function to set wpsRequested.

Here is the caller graph for this function:



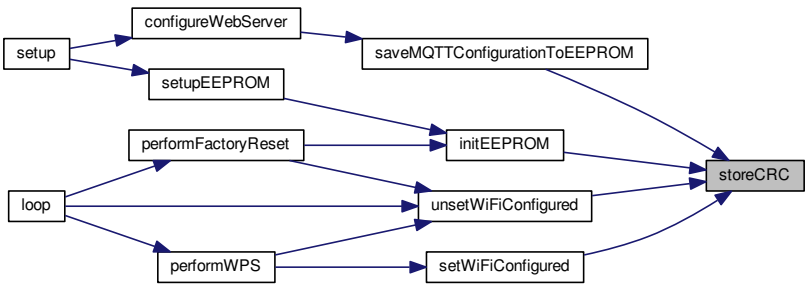
5.3.3.55 storeCRC()

```
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:



5.3.3.56 storeEEPROMVersionNumber()

```
void storeEEPROMVersionNumber ( )
```

This function stores the EEPROM_version_number in EEPROM.

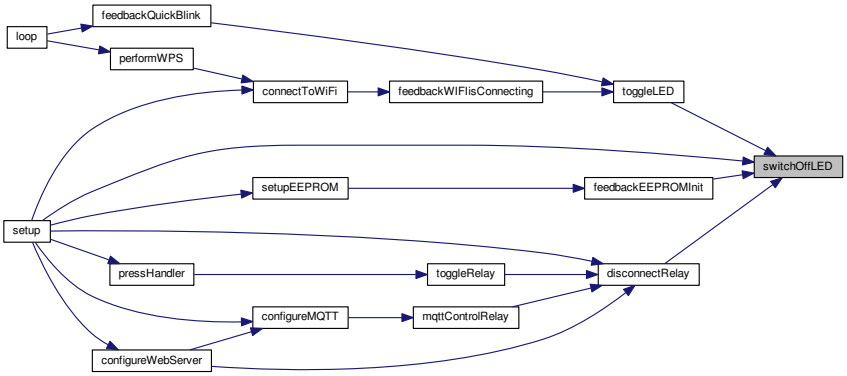
Here is the caller graph for this function:

5.3.3.57 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:

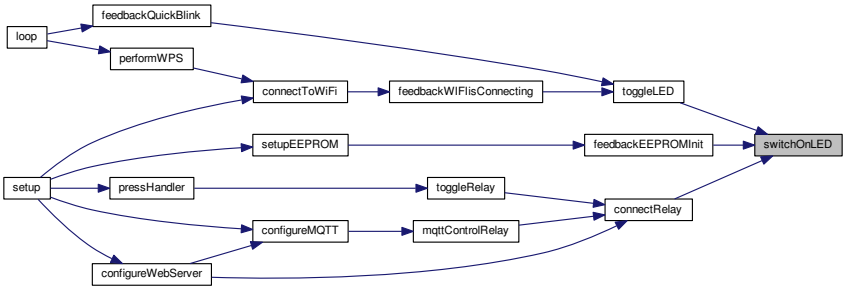


5.3.3.58 switchOnLED()

```
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:

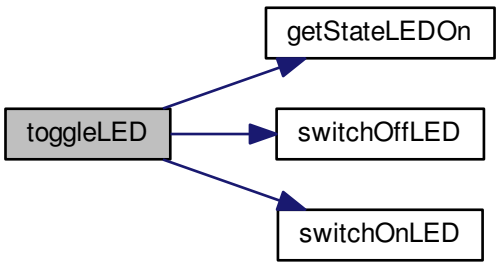


5.3.3.59 toggleLED()

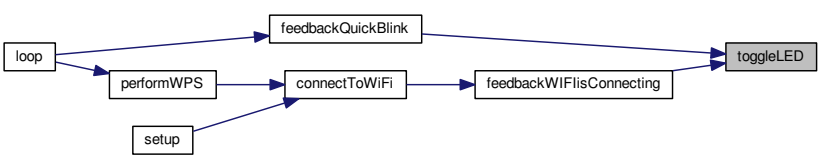
```
void toggleLED ( )
```

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:

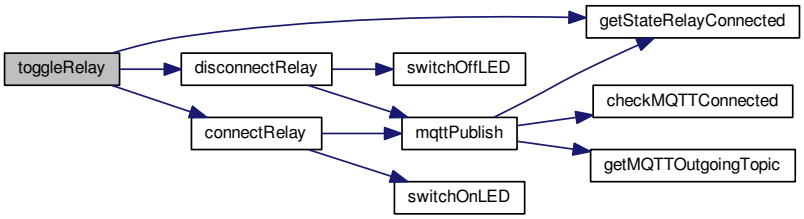


5.3.3.60 toggleRelay()

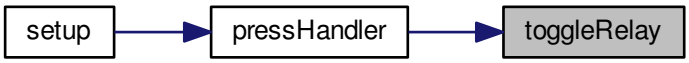
```
void toggleRelay ( )
```

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:

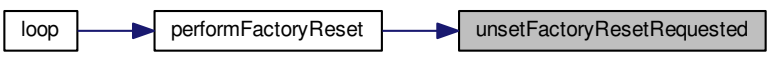


5.3.3.61 unsetFactoryResetRequested()

```
void unsetFactoryResetRequested ( )
```

Setter function to unset factoryResetRequested.

Here is the caller graph for this function:



5.3.3.62 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:

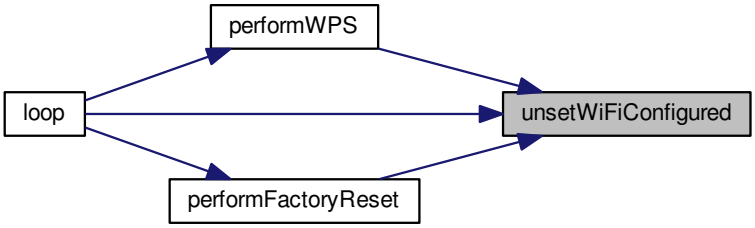
5.3.3.63 unsetWiFiConfigured()

```
void unsetWiFiConfigured ( )
```

This function is a setter function which sets WiFi to not configured, which means that WPS is necessary before connecting to WIFI. The configuration is saved to EEPROM.

Here is the call graph for this function:

Here is the caller graph for this function:

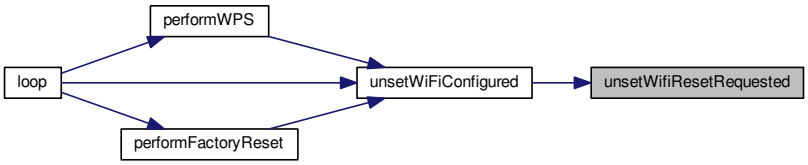


5.3.3.64 unsetWifiResetRequested()

```
void unsetWifiResetRequested ( )
```

Setter function to unsset wifiResetRequested.

Here is the caller graph for this function:

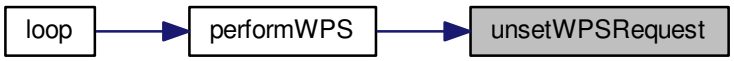


5.3.3.65 unsetWPSRequest()

```
void unsetWPSRequest ( )
```

Setter function to unset wpsRequested.

Here is the caller graph for this function:



5.3.4 Variable Documentation

5.3.4.1 buttonLastPressed

```
bool buttonLastPressed = false
```

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

5.3.4.2 buttonPin

```
const int buttonPin = 0
```

Constant to map the hardware button of the Sonoff S20.

5.3.4.3 factoryResetRequested

```
bool factoryResetRequested = false
```

State to track whether a factory reset has been requested.

5.3.4.4 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

5.3.4.5 mqttIncomingTopic

```
String mqttIncomingTopic
```

The topic which mqttClient subscribes.

5.3.4.6 mqttOutgoingTopic

```
String mqttOutgoingTopic
```

The topic mqttClient publishes.

5.3.4.7 mqttServer

```
String mqttServer = ""
```

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

5.3.4.8 pinLED

```
const int pinLED = 13
```

Constant to map the green LED of the Sonoff S20.

5.3.4.9 pinRelay

```
const int pinRelay = 12
```

Constant to map the relay of the Sonoff S20.

5.3.4.10 REPOSITORY_URL_STRING

```
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().

5.3.4.11 selectionState

```
unsigned long selectionState = 0
```

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

5.3.4.12 stateLEDOn

```
bool stateLEDOn = false
```

State to track whether the LED is on.

5.3.4.13 stateMQTTConfigured

```
bool stateMQTTConfigured = false
```

State to track whether MQTT is configured.

5.3.4.14 stateRelayConnected

```
bool stateRelayConnected = false
```

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

5.3.4.15 ticker

```
Ticker ticker
```

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

5.3.4.16 timeSincebuttonPressed

```
unsigned long timeSincebuttonPressed = 0
```

State to track the time since the button was pressed.

5.3.4.17 userActionFeedbackRequest

```
bool userActionFeedbackRequest = false
```

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

5.3.4.18 webserver

```
ESP8266WebServer webserver
```

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

5.3.4.19 wifiClient

```
WiFiClient wifiClient
```

Necessary to initialize a MQTTClient object.

5.3.4.20 wifiResetRequested

```
bool wifiResetRequested = false
```

State to track whether the deletion of WIFI data has been requested.

5.3.4.21 wpsRequested

```
bool wpsRequested = false
```

State to track whether the WPS has been requested.

Index