IR-Remote

by Marc Ubbelohde

Generated by Doxygen 1.8.17

Chapter 1

Main Page

This repository contains the software for an infrared controller, the purpose of which is to be able to control devices with an infrared interface using a smartphone.

In this guide you find information about following topics:

- Software
  1. Project Structure
  2. Using this Repository
  3. Libraries
  4. Logical Structure
  5. Programs
- Hardware
  1. Components
  2. Schematic

1.1 Software

Lets talk about the software which you can find in this repository. I want to start by explaining the general structure of this projects software and list the libraries used while explaining a bit which role each of them plays. After that I will go into more detail about the logical structure of the software and explain the different parts of it. Finally I will go into more detail about the programs and how they work.
If you are interested in a detailed documentation of the code you can find it here.

1.1.1 Project Structure

The project is a PlatformIO project and is for the most parts structured as such. The Microcontroller I used is an ESP8266 and the software is based on the Arduino framework.
The project is structured as follows:

```
IR-Controller
.pio                            // PlatformIO
|   ...
.vscode                         // VSCode
|   ...
assets                          // assets for documentation
|   ...
docs                            // Doxygen output
   html
|   |   index.html              // Main Page of Code Documentation
|   |   ...
   latex
```

```
|   ...
examples                          // Examples (may be deleted)
|   ...
include                           // Header Files
   main.h
   base.h
   ...
lib                               // Libraries
   ArduinoJson
|   |   ...
   IRremoteESP8266
|   |   ...
   NTPClient
|   |   ...
   Regexp
|   |   ...
   WiFiManager
|   |   ...
src                               // Source Files
   tests                         // Unit Tests
      test_main.cpp
      test_filesystem.cpp
      ...
   main.cpp
   filesystem.cpp
   ...
Doxyfile                          // Doxygen Configuration
platformio.ini                    // PlatformIO Configuration
```

As you can see the project structure is kept quiet simple and should look familiar if you already worked with PlatformIO. The only difference is that the unit tests are located in the src folder instead of the test folder. This is because out of simplicity I decided to write the tests by myself instead of using a framework like Unity.

1.1.2 Using this Repository

1.1.2.1 Prerequisites

To use this project you need to have PlatformIO installed. You can find the installation instructions here. You also need to have a ESP8266 Microcontroller. I used a WEMOS D1 Mini but any other ESP8266 should work as well.

1.1.2.2 Setup

To setup the project you need to clone the repository and open it in PlatformIO. You should now be able to build the project and upload it to your ESP8266. The WiFiManager should start autmatically to allow you to connect to your WiFi. Alternatively you can connect via WPS by pressing first the button on your router and then the button on the UI or you switch to Access Point mode to connect directly to the ESP8266.

1.1.2.3 Testing

As mentioned I wrote my own unit tests for simplicity. You can run the tests by simply uncommenting the marked lines in the main.cpp file. The tests are run automatically on startup directly on the device and the results are printed to the serial monitor. If they fail the execution of the program is halted.

1.1.2.4 Documentation

For the detailed Code Documentation I use Doxygen. The documentation is generated by running the command `doxygen Doxyfile` in the root directory of the project. The output of the documentation is located in the docs folder. Please note that in order to generate the documentation you need to have Doxygen installed. You can find the installation instructions here. If you are a Linux user you can also install doxygen by running the command `sudo apt-get install doxygen doxygen-doc doxygen-gui graphviz` in your terminal.

1.1.3 Libraries

In addition to the Arduino framwork the following libraries helped me to realize this project:

1.1.3.1 ArduinoJson

ArduinoJson is a library for parsing and generating JSON. It is used to store IR-Signals, time data and other data in orderly fashion. The JSON format allowed me to easily write and read the data to and from the LittleFS (filesystem) without the use of complex string manipulation or making up my own format.

1.1.3.2 IRremoteESP8266

IRremoteESP8266 is a library for receiving and sending IR-Signals. It is used to receive IR-Signals from the IR-Receiver and to send IR-Signals to the IR-LED.

1.1.3.3 NTPClient

NTPClient is a library for getting the current time from an NTP-Server. It is exlusively used in timed programs (where a signal is sent at a specific time). It is used to initialize the time on boot.

1.1.3.4 Regexp

Regexp is a library that allowed me to use regular expressions in my code. It is used to scan a user written program for the correct syntax.

1.1.3.5 WiFiManager

WiFiManager is a library that allows you to connect to a WiFi network by entering the credentials in an UI instead of hard coding them. It is used to connect to the users WiFi network. I modified the library slightly to allow the user to connect via WPS and to switch to Access Point mode.

1.1.4 Logical Structure

In this section I want to give you a high level overview of the logical structure of the software. I grouped them into different parts that I found to be the most important. I will explain each section in detail.

1.1.4.1 Setup

The device setup includes every step that is necessarey to reach the normal operation state which in best case will after the initial setup be the starting point after rebooting.
Diagram of the Setup:
As you can see in the diagram the device can operate in 2 different modes: AP-mode or STA-mode.
By default the device will enter STA-mode which means that it wants to connect to a WiFi network. In order to do so the device makes use of the WiFiManager library which enables it to briefly creates an access point to which the user has to connect. The user can then enter the credentials of the WiFi network he wants to connect to. The device will then try to connect to the network. If the connection is successful the device will now operate in STA-mode. If the connection fails the device will reboot and ask you again to enter the credentials. Alternatively you can also connect to your router via WPS. In order to do so you have to press the WPS on your router and then the button on the UI. If you want to connect to the device directly without using a WiFi network you can switch to AP-mode by pressing the button in the UI.
In AP-mode the device creates an access point in which it operates. The user connects to the access point via smartphone or computer. The access point is password secured and the user can change the password in the UI. Finally the time has to be set manually as the device does not have an external RTC.

1.1.4.2 LittleFS

The LittleFS is the filesystem of the ESP8266. It is used to stors the signals and program the user creates. It is also used to store time data, the password of the access point and the mode the device is currently in. Let me start by explaining the structure of the filesystem.
Diagram of the LittleFS:

```
ESP8266 LittleFS
signals              // Folder for the signals
   signal1.json
   signal2.json
   ...
programs            // Folder for the programs
   program1.txt
   program2.txt
   ...
time.json           // File for the time data
password.txt        // File for the password of the access point
config.txt          // File for the mode the device is currently in (AP or STA)
```

As you can see the signals and programs are stored in their designated folders. What does the data inside the files look like? This differs from file to file. The data in the signal files is stored in json format and looks like this:

```
{
  "name": <signal_name>,
  "length": <signal_length>,
  "sequence": <signal_sequence>
}
```

The data in the program files however is stored as a normal C-String. This is because only the program code has to be stored in the file and this makes handling the data easier.
In general the signal and program files are modified by the functions in the `filesystem.cpp` file. The functions are called by functions from the `workflows.cpp` file which is responsible for the high level logic of the device.
The files from the root directory define the configuration and state of the device and are modified partly directly by the handler functions in `main.cpp`.

1.1.4.3 Webserver

The webserver is responsible for the communication between the device and the user. It therfore includes receive commands from the user and displaying the current state of the device to the user.
Diagram of the Webserver:
Since I used a synchronous webserver UI updates are only possible after the user reloads the website. This means that after each input the website will be reloaded. This is not a problem since the website is very lightweight and the user will not notice any delay.
As you can see in the `website.html` file the website makes heavy use of the nature of the HTML form element. Since HTML form elements automatically trigger a get request on their specific action url containing the specified data they make it very easy to send data from the website to the device. To display the current state of the device which includes saved signals and programs or if the device is in Access Point or Station mode the website sends a get request to the device each time the website is reloaded which the device then responds to with the current state.
There is one exception which I would like to point out here. The edit function of the website is the only function that involves the device sending data which is dependant on the websites state. This means that the website hast to send a get request via the form element to the device which then responds with the data. Since the dropdown menu is part of a html form element that triggers a redirect to the url of the get the device has to answer with a redirect to the root url. So there is no space for another http header in the response. The solution I came up with is to let the backend set the variable PROGRAMNAME to the selected program whenever the edit button is pressed and then send the code of that program every time the website is reloaded. After each reload the variable is set to "" again. This results in the desired behavior.

1.1.4.4 Time Management

Time Management turned out to be more complicated than I initially thought. This is mostly due to the fact that the millis() function overflows after about 49 days and that one requirement was to be able to execute timed programs even without internet connection. Thats why I want to dedicate this section to it.
The time is saved in the /time.json and has following format:

```
{
  "hours": <hh>,
  "minutes": <mm>,
  "seconds": <ss>,
  "weekday": <w>,
  "timezone": GMT+<timezone>,
  "init_offset": <offset>,
  "last_offset": <offset>
}
```

Hours, minutes and seconds dont need any explenation, weekday is saved as a number from 0 to 6 where 0 is Sunday and 6 is Saturday. The timezone is saved in seconds i. e. GMT+1 is saved as 3600. The init_offset is the offset that was used to initialize the time. The last_offset is the offset at which the last overflow check took place (this becomes important later).
The time gets initialized with time from an NTP server (saved timezone is respected if no timezone is saved GMT is used). If the device is not connected to the internet the request to the NTP server will fail and by NTPClient library default the time will be initialized with millis(). If that happens the user will have to update the time manually via the web interface. If the device is in AP-mode the user updates the time completly. If the device is in STA-mode the user can only update the timezone. The rest was done automatically by the NTPClient library. Lets look at a diagram explaining both this an how the millis() overflow is handled.
Diagram of the time management:
As mentioned before the millis() funciton overflows after about 49 days. In order to prevent this in `time_management` you can find the funcion check_and_update_offset at the end of the file which is called every time

long waiting periods are expected to occur. The function compares the current value of millis() to the last_offset and if the current value is smaller than the last_offset it means that millis() overflowed and the time and the init_offset have to be updated. Since we wont hit exactly the moment of overflow the function now checks millis() to see how much time passed since the overflow and reinitializes the time with the current offset.

1.1.5 Programs

At the end of this section I want to give you a brief overview of the different commands that are available in the program. Before I go into detail about each command there are 4 points to consider:

1. Programs are executed successively,
2. each command is written in a new line,
3. empthy lines are skipped and
4. programs can be aborted by pressing the designated button on the device.

1.1.5.1 play

The play command plays a signal. The syntax is as follows:
`play <signal name>`
It is important to not that there is a small break between sending 2 signals after each other. This is because the device is still processing the signal and so it can take up to 100ms to send the next signal.

1.1.5.2 wait

The wait command waits a specified amount of milliseconds. The syntax is as follows:
`wait <milliseconds>`
It is important to note that the maximum amount of milliseconds that can be waited is 4294967295. (about 49 days)

1.1.5.3 time

The time command waits until a specified time before sending a signal. The syntax is as follows:
`<hour>:<minute>:<second> <signal name>`

1.1.5.4 day

The day command is similar to the time command but it waits until a specified day and a specified time before sending the signal. The syntax is as follows:
`<day> <hour>:<minute>:<second> <signal name>`
The day is writte in english and can be capitalized or not.

1.1.5.5 skip

The skip command skips a specified amount of days and can be useful in timed Programs. The syntax is as follows:
`skip <days>`
Similar to the wait command the maximum amount of days that can be skipped is 49.

1.1.5.6 loop

The loop command loops the lines between the loop command and the end command a specified amount of times or infinitely often. The syntax is as follows:

```
loop <times> or "inf"
(code to be repeated)
end
```

1.2 Hardware

In this section I want to talk a bit about the inner workings of the device. I will start by giving you an overview of some of the most important components I used, why I used them and what role they play in the device. After that you will find the circuit diagram.

1.2.1 Components

1.2.1.1 Microcontroller (ESP8266)

The ESP8266 is a cheap and powerful microcontroller that is perfect for this project. It has a lot of GPIOs, a lot of memory, lots of processing power and often comes with a build in wifi antenna. It is also very easy to program and has a lot of libraries available. The ESP8266 is also very cheap and can be bought for less than 2€. In this project I used the ESP8266-12F which comes in a small form factor and features a build in wifi antenna.

1.2.1.2 IR-LED

The nature of IR light is probably the big botleneck of this project. To make the best out of it the IR-LED has to be as powerfull and multidirectional as possible. I used the so called "WTN-3W-IR940" which is a 3W 940nm IR-LED which has a 360 degree beam angle and a 180 degree viewing angle. The IR-LED is controlled by the ESP with a logic level transistor and is equipped with a small heatsink to prevent overheating. Since the IR-LED is pulsed very quickly it can easily resist currents of more than 1A.

1.2.1.3 IR-Receiver

The TL1838 is a very cheap IR-Receiver that is trimmed to 38kHz though it is possible to decode signals with frequencies from 30 kHz up to 60kHz with this sensor. Therfore it is perfect to decode most IR-Signals. The IR-↩ Receiver is directly connected to the Microcontroller and the output is processed by the IRremoteESP8266 library.

1.2.1.4 Power Supply

The Powersupply is the last component that I want to talk about. It is the Mean Well EPS-15-3.3 which can output between 3.1V and 3.6V with a maximum current of 3A. It is a very cheap and easy to use but still solid power supply that in the end made up by far for the most space in the device.

1.2.2 Schematic

The schematic of the device is shown below. It is a very simple circuit that is easy to understand. All devices are powered by the power supply which is filtered by several condensators. The IR-LED is controlled by a n-channel transistor (IRLML6344) which is pulled down. The device features 2 push buttons and 1 on/off switch. Both the reset and stop button are pulled down. The ESP8266-12F can be programmed via a USB to serial converter (i.e. FT232RL) that has to be connected to the J1 Header Pin connector. Note that RXD has to be connected to TXD on the Serial to USB converter (and TXD to RXD). Additionally the device has to be powered by the power supply since the Serial to USB Converter is not able to power the ESP by itself. Also make sure the Serial to USB Converter is set to 3.3V.
Schematic:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

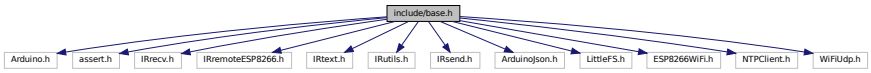
File Documentation

3.1 include/base.h File Reference

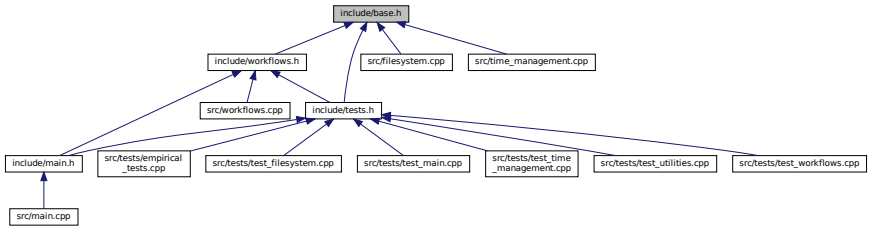
Header file for filesystem.cpp and time_management.cpp.

```
#include <Arduino.h>
#include <assert.h>
#include <IRrecv.h>
#include <IRremoteESP8266.h>
#include <IRtext.h>
#include <IRutils.h>
#include <IRsend.h>
#include <ArduinoJson.h>
#include <LittleFS.h>
#include <ESP8266WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
```

Include dependency graph for base.h:



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



Functions

- String capture_signal ()

  This function captures a signal and returns it as a String.

- String save_signal (String result_string, String name)

  This function saves a captured signal.

- void save_json (String filename, DynamicJsonDocument doc)

  This function saves a JSON document to a specified file.
- DynamicJsonDocument load_json (String filename)

  This function loads a JSON document from a specified file.
- String send_signal (DynamicJsonDocument doc)

  This function sends a signal provided in JSON format.
- String get_files (String folder_signals, String folder_programs)

  Returns List of saved signals and programs.
- boolean check_if_file_exists (String filename)

  Checks if a file exists.
- String read_program (String program_name)

  Reads a program file and returns its content as a String.
- void control_led_output (String signal)

  Controls the LED output.
- boolean check_if_string_is_alphanumeric (String word)

  Checks if a String is alphanumeric.
- String weekday_to_num (String weekday)

  Converts a weekday String to a weekday number.
- boolean compare_time (String time, boolean weekday_included)

  Compare specified time with current time.
- void update_time (String time, boolean AP_mode)

  Updates the time in the LittleFS.
- String get_current_time ()

  Loads the current time from LittleFS.
- String turn_seconds_in_time (unsigned long input_seconds)

  This function converts seconds to time format.
- String add_time (String time, String offset_time)

  adds two times together
- void init_time ()

  Gets time from NTP server.
- void check_and_update_offset ()

  Checks if millis() overflowed and updates time if necessary.

3.1.1 Detailed Description

Header file for filesystem.cpp and time_management.cpp.

Author

Marc Ubbelohde

This file includes the dependencies for the filesystem.cpp and time_management.cpp files.

3.1.2 Function Documentation

3.1.2.1 add_time()

```
String add_time (
            String time,
            String offset_time )
```

adds two times together

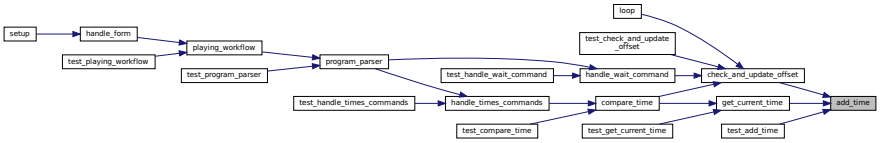
Parameters

time	- time to add to in format "hh:mm:ss weekday"
offset_time	- time to add in format "hh:mm:ss"

Returns

String - time in format "hh:mm:ss weekday"

This function adds two times together. The order of the parameters is important. The first parameter contains the weekday, the second parameter does not.
This function does not call other functions. Here is the caller graph for this function:



3.1.2.2 capture_signal()

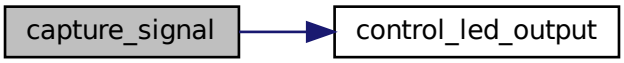
```
String capture_signal ( )
```

This function captures a signal and returns it as a String.

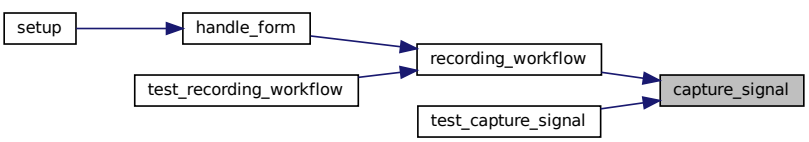
Returns

String - String containing the captured signal in the format:
"uint16_t rawData[67] = {1234, 5678, ...};" - if signal was receiver, format defined by resultToSourceCode() in IRremoteESP8266/src/IRutils.cpp
"no_signal" - if no signal was captured

This function uses the IRremoteESP8266 library and is based on the IRrecvDumpV2 example from the library. Here is the call graph for this function:



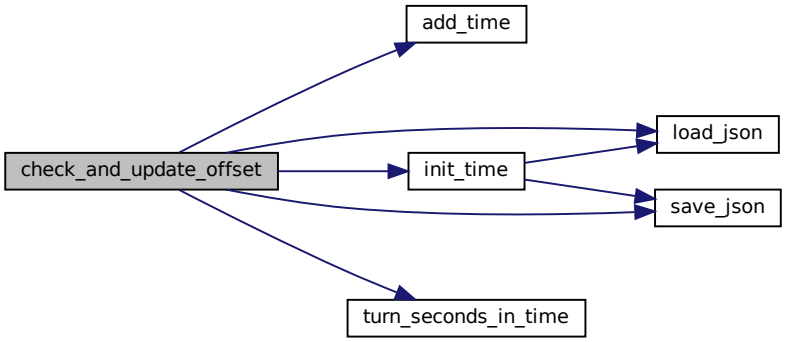
Here is the caller graph for this function:



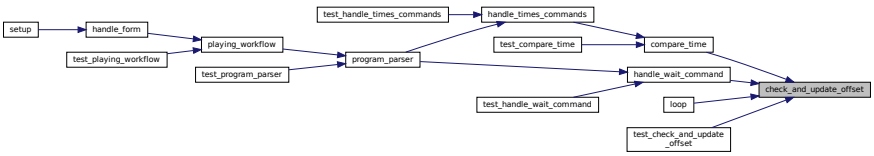
3.1.2.3 check_and_update_offset()

```
void check_and_update_offset ( )
```

Checks if millis() overflowed and updates time if necessary.
Since millis() overflows after 49.7 days, this function checks if an overflow occured and updates the time saved in "time.json" every time it happens to still be able to calculate teh current time. This function is used whenever long waiting times are expected (e.g. in timed programs, wait/skip command or user inactivity). It is important to note that this function has to be able to work offline (no NTP call) since it should be possible to run programs without internet connection. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.4 check_if_file_exists()

```
boolean check_if_file_exists (
            String filename )
```

Checks if a file exists.

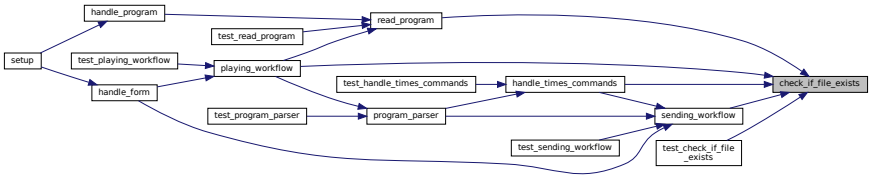
Parameters

filename	- name of the file to be checked

Returns

boolean - true if the file exists, false if not

This function checks if a file exists in the LittleFS.
This function does not call any other function. Here is the caller graph for this function:



3.1.2.5 check_if_string_is_alphanumeric()

```
boolean check_if_string_is_alphanumeric (
            String word )
```

Checks if a String is alphanumeric.

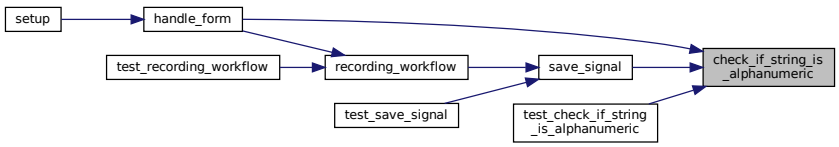
Parameters

word	- String to be checked

Returns

boolean - true if the String is alphanumeric, false if not

This function checks if a String is alphanumeric. Spaces, dashes and underscores are also allowed.
This function does not call any other function. Here is the caller graph for this function:



3.1.2.6 compare_time()

```
boolean compare_time (
            String time,
            boolean weekday_included )
```

Compare specified time with current time.

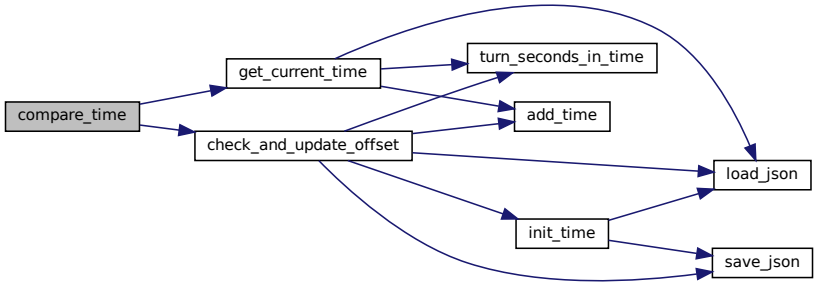
Parameters

time	- String in format "weekday hh:mm:ss timezone"
weekday_included	- true if weekday is included in time, false if not

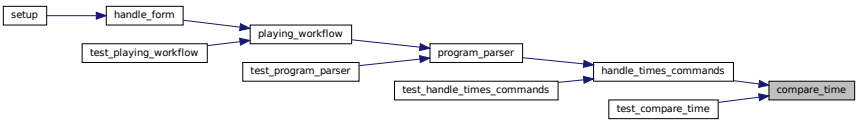
Returns

boolean - true if time is equal to current time, false if not

This elementary function checks if the current time is equal to the time in the program. It is used in timed programs and handles millis() overflow. The function has a delay of 500ms to reduce the number of operations inside the while(true) loop. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.7 control_led_output()

```
void control_led_output (
            String signal )
```

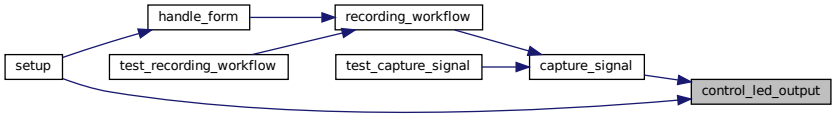
Controls the LED output.

Parameters

signal	- String containing the signal to be send: "no_signal" - LED blinks 3 times "no_mDNS" - LED blinks 3 times "signal_received" - LED blinks once

This function controls the LED output via codewords to specify the kind of signal to be send. The LED is connected to GPIO 5 (D1) on the ESP8266 and is ment as a way to communicate errors to the user and singal when the ESP is ready to receive a signal.

This function does not call any other function. Here is the caller graph for this function:



3.1.2.8 get_current_time()

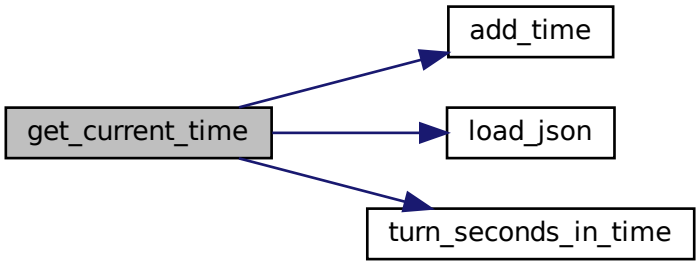
```
String get_current_time ( )
```

Loads the current time from LittleFS.

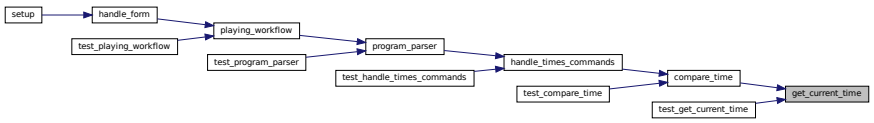
Returns

String - current time in format "hh:mm:ss weekday"

This function loads the time from the LittleFS, adds the relative offset between the offset of initialization and the current offset and returns the current time. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.9 get_files()

```
String get_files (
            String folder_signals,
            String folder_programs )
```

Returns List of saved signals and programs.

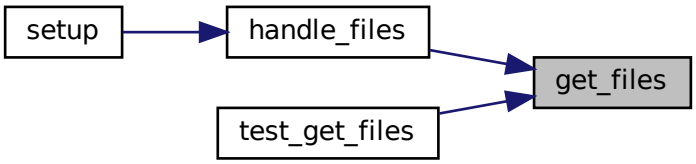
Parameters

folder_signals	- name of the folder containing the signals
folder_programs	- name of the folder containing the programs

Returns

String - String containing all files in the specified folders, separated by a semicolon:
"signal1, signal2, ...;program1, program2, ..."

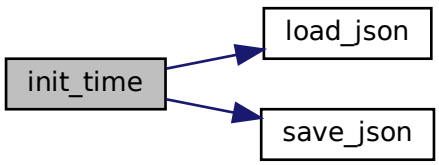
This function scans the signal and program folders for files and returns a String containing all files. It uses LittleFS.
This function does not call any other function. Here is the caller graph for this function:



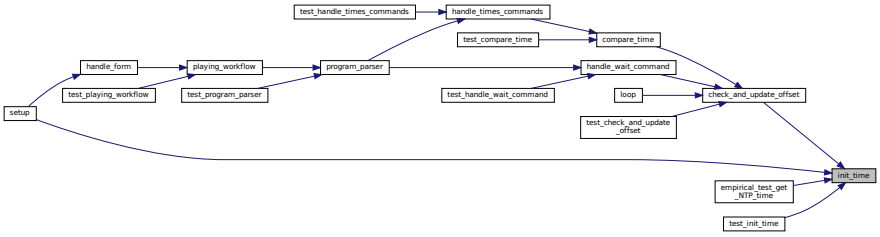
3.1.2.10 init_time()

```
void init_time ( )
```

Gets time from NTP server.
This function initiates the time by getting the time from the NTP server. It then saves it ot the LittlfeFS. It also passes the saved timezone to the NTP server or 0 if no timezone is saved. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.11 load_json()

```
DynamicJsonDocument load_json (
            String filename )
```

This function loads a JSON document from a specified file.

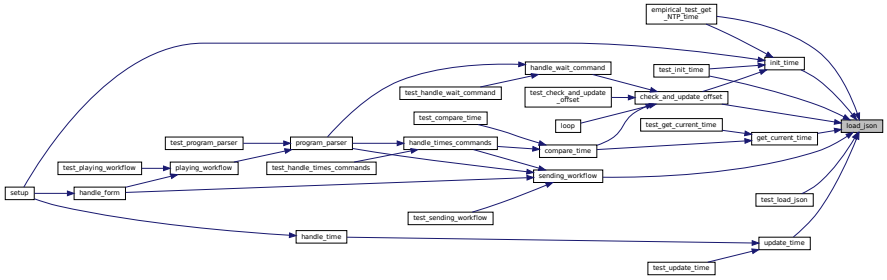
Parameters

filename	- name of the file

Returns

DynamicJsonDocument - JSON document containing unspecified data

This function uses LittleFS and the ArduinoJson library.
This function does not call any other function. Here is the caller graph for this function:



3.1.2.12 read_program()

```
String read_program (
            String program_name )
```

Reads a program file and returns its content as a String.

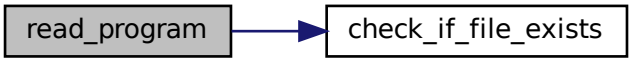
Parameters

program_name	- name of the program to be read

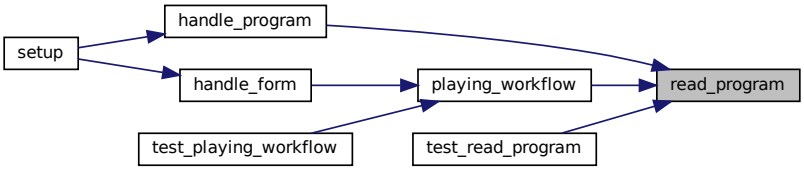
Returns

String - String containing the program code

This function has its reason of existence next to load_json because in the frontend, after pressing the "edit" button, the program code of the specified program is displayed in the textarea as a string.
This function does not call any other function. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.13 save_json()

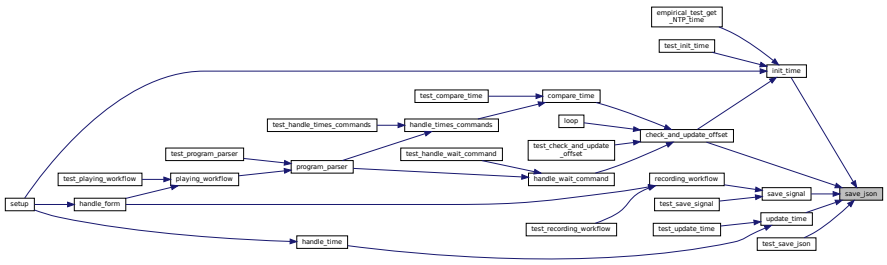
```
void save_json (
            String filename,
            DynamicJsonDocument doc )
```

This function saves a JSON document to a specified file.

Parameters

filename	- name of the file
doc	- JSON document containing unspecified data

This function uses LittleFS and the ArduinoJson library.
This function does not call any other function. Here is the caller graph for this function:



3.1.2.14 save_signal()

```
String save_signal (
            String result_string,
            String name )
```

This function saves a captured signal.

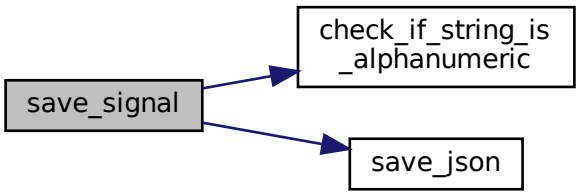
Parameters

result_string	- String containing the captured signal in the format: "uint16_t rawData[67] = {1234, 5678, ...};" (format defined by resultToSourceCode() in IRremoteESP8266/src/IRutils.cpp)
name	- name of the signal

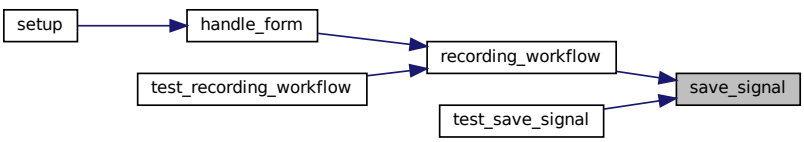
Returns

String - "success" - if signal was saved successfully
"Error: ..." - if an error occurred

This function saves a captured signal in json format to a file. It uses the ArduinoJson library. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.15 send_signal()

```
String send_signal (
            DynamicJsonDocument doc )
```

This function sends a signal provided in JSON format.

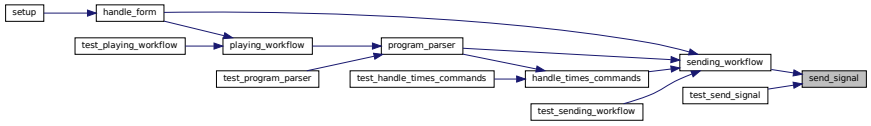
Parameters

doc	- JSON document containing the signal to be sent:
	{
	"name": "name",
	"length": 67,
	"sequence": "1234, 5678, ..."
	}

Returns

String - "success" if sending was successful
"Error: ..." if sending failed

This function sends a signal provided in JSON format. It uses the IRremoteESP8266 library and is based on the example code provided by the library.
This function does not call any other function. Here is the caller graph for this function:



3.1.2.16 turn_seconds_in_time()

```
String turn_seconds_in_time (
            unsigned long input_seconds )
```

This function converts seconds to time format.

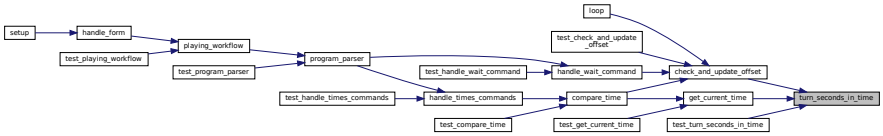
Parameters

input_seconds	- seconds to convert

Returns

String - time in format "hh:mm:ss"

This function converts seconds to time format. It is used in get_current_time() to prepare millis() offset for comparison with saved time.
This function does not call other functions. Here is the caller graph for this function:



3.1.2.17 update_time()

```
void update_time (
```

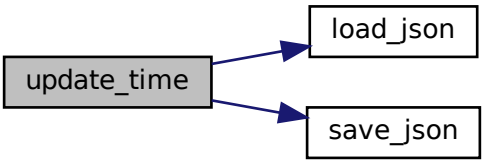
```
            String time,
            boolean AP_mode )
```

Updates the time in the LittleFS.

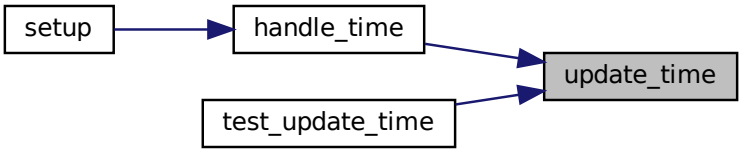
Parameters

time	- String in format "weekday hh:mm:ss timezone"
AP_mode	- true if the device is in AP mode, false if not

This function is called when the user presses the "sync" button on the website. It updates only the timezone to the LittleFS since the time from the NTP request is more precise than the time from the user. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.18 weekday_to_num()

```
String weekday_to_num (
            String weekday )
```

Converts a weekday String to a weekday number.

Parameters

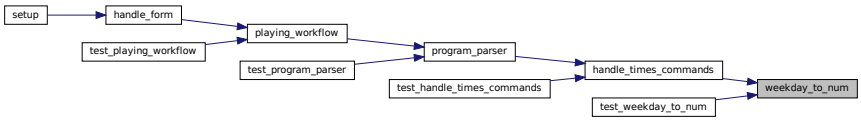
weekday	- weekday as a String

Returns

String - weekday as a number:
"Monday" - "1"
"Tuesday" - "2"

"Wednesday" - "3"
"Thursday" - "4"
"Friday" - "5"
"Saturday" - "6"
"Sunday" - "0"
error - "error"

This function converts a weekday String to a weekday number.
This function does not call any function. Here is the caller graph for this function:

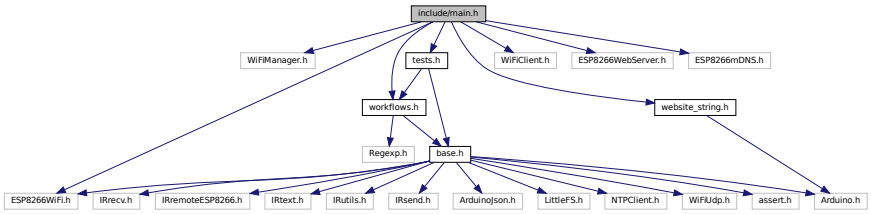


3.2 include/main.h File Reference

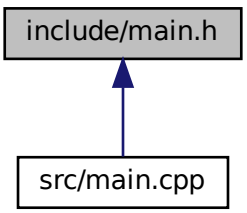
Header File for main.cpp.

```
#include <WiFiManager.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include "workflows.h"
#include "website_string.h"
#include "tests.h"
```

Include dependency graph for main.h:



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



Functions

- void handle_root ()

  Handler function for the root page.
- void handle_not_found ()

  Handler function for any page that is not found.
- void handle_program ()

  Handler function to display the program the user wants to edit.
- void handle_error ()

  Handler function to display the error message.
- void handle_files ()

  Handler function to send a list of all signals and programs to the forntend.
- void handle_time ()

  Handler funciton to synchronize the time and/or timezone provided by the user.
- void handle_credentials ()

  Handler function to erase the wifi credentials saved on the ESP.
- void handle_apmode ()

  Handler function to switch between AP mode and normal mode.
- void handle_apinfo ()

  Handler function to send the current AP mode setting to the frontend.
- void handle_password ()

  Handler function that receives new password entries.
- void handle_form ()

  Handler function that receives GET requests from all form elements related to signals and programs.
- ESP8266WebServer server (80)

  Constructor of the webserver that is used to serve the website.

Variables

- String PROGRAMNAME = ""

  Holds the name of the currently selected program if the edit button was pressed. Gets updated on /form and called on /program.
- String MESSAGE = ""

  Holds the message that is displayed on the website and updated on reload.
- boolean SESSION_AP = true

  Stores the session value (if AP-Mode is activated or not).
- boolean AP_SETTING = true

  Tracks which setting to be used on reboot.

3.2.1 Detailed Description

Header File for main.cpp.

Author

Marc Ubbelohde

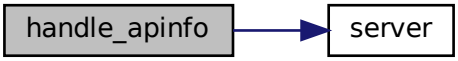
This file includes dependencies for the wifi setup and webserver. Also the HTML code for the website is included and the variables that are shared between the handler functions are declared.

3.2.2 Function Documentation

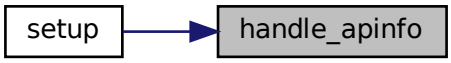
3.2.2.1 handle_apinfo()

```
void handle_apinfo ( )
```

Handler function to send the current AP mode setting to the frontend.
Sends the current AP mode setting to the frontend. This is used to display the correct setting at the top the website.
This function is called on a GET request to /apinfo. Here is the call graph for this function:



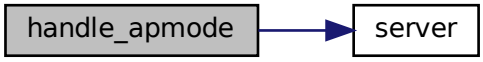
Here is the caller graph for this function:



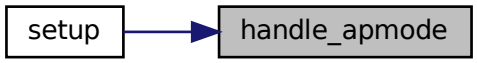
3.2.2.2 handle_apmode()

```
void handle_apmode ( )
```

Handler function to switch between AP mode and normal mode.
Checks the config file and switches between AP mode and normal mode. The possibel reconfiguration of the wifi credentails is done after restart.
This function is called on a GET request to /apmode. Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.3 handle_credentials()

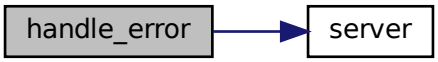
`void handle_credentials ( )`
Handler function to erase the wifi credentials saved on the ESP.
Erases the wifi credentials saved on the ESP. This is useful if the user wants to connect to a different wifi network. This function is called on a GET request to /credentials. Here is the call graph for this function:

Here is the caller graph for this function:

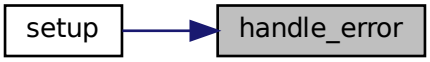
3.2.2.4 handle_error()

`void handle_error ( )`
Handler function to display the error message.
Similarly to handle_program(), the website will send a get request on /error to update the error message on reload. The MESSAGE is then written again by handle_form().

This function is called on a GET request to /error. Here is the call graph for this function:



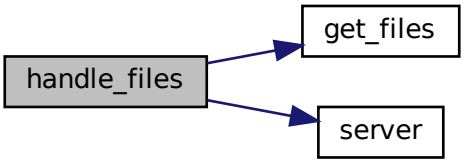
Here is the caller graph for this function:



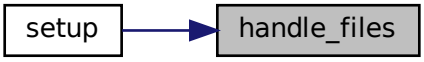
3.2.2.5 handle_files()

`void handle_files ( )`

Handler function to send a list of all signals and programs to the forntend.
Sends a list of all files in /signals and /programs on reload to be displayed on the website.
This function is called on a GET request to /files. Here is the call graph for this function:



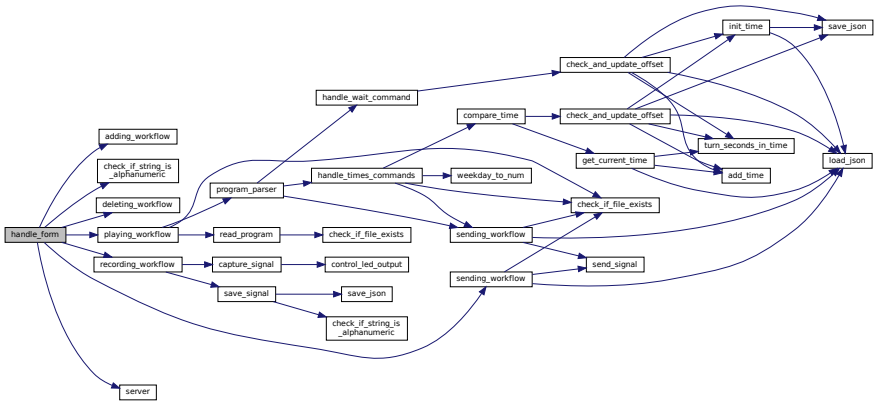
Here is the caller graph for this function:



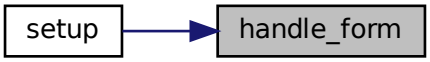
3.2.2.6 handle_form()

`void handle_form ( )`

Handler function that receives GET requests from all form elements related to signals and programs.
Handles all form elements on the website (signals and programs) also updates the error message and PROGRA↩
MNAME. All the user interaction with the website is handled here and the functions from workflows.h are called.
This function is called on a GET request to /form. Here is the call graph for this function:



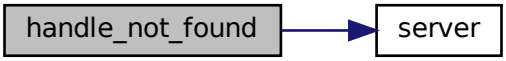
Here is the caller graph for this function:



3.2.2.7 handle_not_found()

`void handle_not_found ( )`

Handler function for any page that is not found.
This function is called when a page is requested that is not found. It serves a 404 error to the client.
This function is called on a GET request to a page that is not found. Here is the call graph for this function:

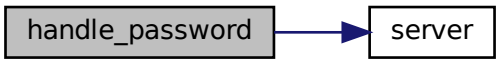


Here is the caller graph for this function:

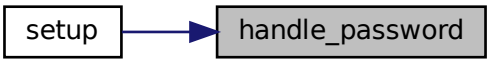
3.2.2.8 handle_password()

```
void handle_password ( )
```

Handler function that receives new password entries.
Receives new password entries from frontend and changes password if entries are the same (to prevent typos).
This function is called on a GET request to /password. Here is the call graph for this function:



Here is the caller graph for this function:

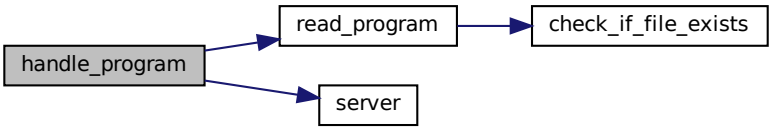


3.2.2.9 handle_program()

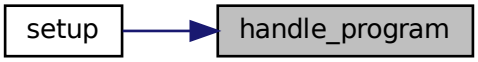
```
void handle_program ( )
```

Handler function to display the program the user wants to edit.
This function is called when the user wants to edit a program. It reads the program from the file system and sends it back to the website. The website is designed with form elements that trigger forwarding to /form. This is a problem because we can only communicate on that channel and we already have to communicate a back to the website to update the page. So in order to still be able to send data back to the website, on every reload the website will send a get request on /program which triggers this function. The data that is sent back is the name of the currently selected program and the code of that program. It is only sent back if the "edit" button was pressed. (if not, the variables "PROGRAMNAME" and "PROGRAMCODE" will be empty)

This function is called on a GET request to /program. Here is the call graph for this function:



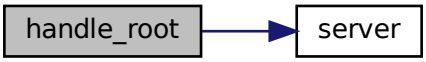
Here is the caller graph for this function:



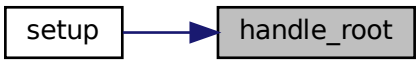
3.2.2.10 handle_root()

```
void handle_root ( )
```

Handler function for the root page.
This function is called when the root page is requested. It serves the content of the index_html string from the website_string.h file to the client.
This function is called on a GET request to the root page. Here is the call graph for this function:



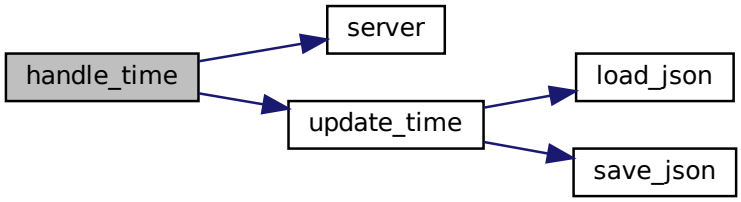
Here is the caller graph for this function:



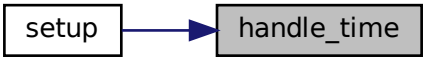
3.2.2.11 handle_time()

```
void handle_time ( )
```

Handler funciton to synchronize the time and/or timezone provided by the user.
Gets the time from the client via Date() and saves it together with the millis() offset to the LittleFS. The offset is important because only with millis() we can calculate the time that has passed between the synchronisation and the time of program execution. This enabled the ESP to execute timed programs even if the wifi connection is lost.
This function is called on a GET request to /time. Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.12 server()

```
ESP8266WebServer server (
            80  )
```

Constructor of the webserver that is used to serve the website.

3.2.3 Variable Documentation

3.2.3.1 AP_SETTING

```
boolean AP_SETTING = true
```

Tracks which setting to be used on reboot.

3.2.3.2 MESSAGE

```
String MESSAGE = ""
```

Holds the message that is displayed on the website and updated on reload.

3.2.3.3 PROGRAMNAME

`String PROGRAMNAME = ""`
Holds the name of the currently selected program if the edit button was pressed. Gets updated on /form and called on /program.

3.2.3.4 SESSION_AP

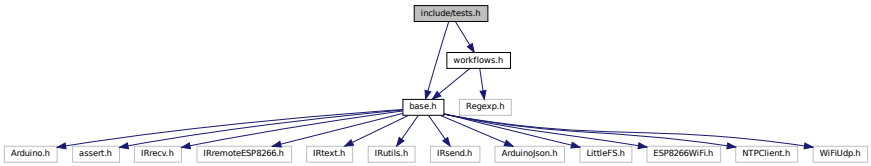
`boolean SESSION_AP = true`
Stores the session value (if AP-Mode is activated or not).

3.3 include/tests.h File Reference

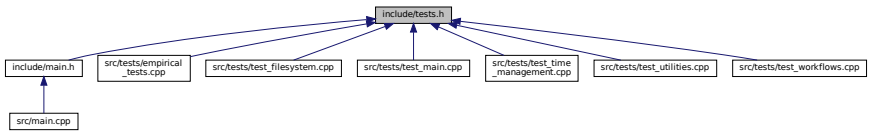
Header File for all test files.

```
#include "base.h"
#include "workflows.h"
```

Include dependency graph for tests.h:



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



Functions

- void clean_LittleFS ()

  Deletes all files in "/", "/signals" and "/programs" in the LittleFS.
- boolean test_capture_signal ()

  Unit test for the function "capture_signal".
- boolean test_save_signal ()

  Unit test for the function "save_signal".
- boolean test_save_json ()

  Unit test for the function "save_json".
- boolean test_load_json ()

  Unit test for the function "load_json".
- boolean test_send_signal ()

  Unit test for the function "send_signal".
- boolean test_get_files ()

  Unit test for the function "get_files".
- boolean test_check_if_file_exists ()

  Unit test for the function "check_if_file_exists".

- boolean test_read_program ()

  Unit test for the function "read_program".
- boolean test_control_led_output ()

  Unit test for the function "test_control_led_output".
- boolean test_check_if_string_is_alphanumeric ()

  Unit test for the function "test_check_if_string_is_alphanumeric".
- boolean test_weekday_to_num ()

  Unit test for the function "weekday_to_num".
- boolean test_compare_time ()

  Unit test for the function "compare_time".
- boolean test_update_time ()

  Unit test for the function "update_time".
- boolean test_get_current_time ()

  Unit test for the function "get_current_time".
- boolean test_turn_seconds_in_time ()

  Unit test for the function "turn_seconds_in_time".
- boolean test_add_time ()

  Unit test for the function "add_time".
- boolean test_init_time ()

  Unit test for the function "get_NTP_time".
- boolean test_check_and_update_offset ()

  Unit test for the function "check_and_update_offset".
- boolean test_deleting_workflow ()

  Unit test for the function "deleting_workflow".
- boolean test_recording_workflow ()

  Unit test for the function "recording_workflow".
- boolean test_sending_workflow ()

  Unit test for the function "sending_workflow".
- boolean test_adding_workflow ()

  Unit test for the function "adding_workflow".
- boolean test_playing_workflow ()

  Unit test for the function "playing_workflow".
- boolean test_program_parser ()

  Unit test for the function "program_parser".
- boolean test_handle_wait_command ()

  Unit test for the function "handle_wait_command".
- boolean test_handle_times_commands ()

  Unit test for the function "handle_times_commands".
- boolean run_all_filesystem_tests (boolean stop_on_error)

  runs all tests for filesystem.cpp
- boolean run_all_time_management_tests (boolean stop_on_error)

  runs all tests for time_management.cpp
- boolean run_all_workflows_tests (boolean stop_on_error)

  runs all tests for workflows.cpp
- void run_all_tests (boolean stop_on_error)

  runs all tests for all files
- void run_all_empirical_tests (boolean stop_on_error)

  runs all empirical tests
- boolean empirical_test_get_NTP_time ()

  Tests empirically the function get_NTP_time()

3.3.1 Detailed Description

Header File for all test files.

Author

Marc Ubbelohde

This file provides access to the source code for all unit test funcitons. It also provides access to the unit test functions for the functions in test_main.cpp.

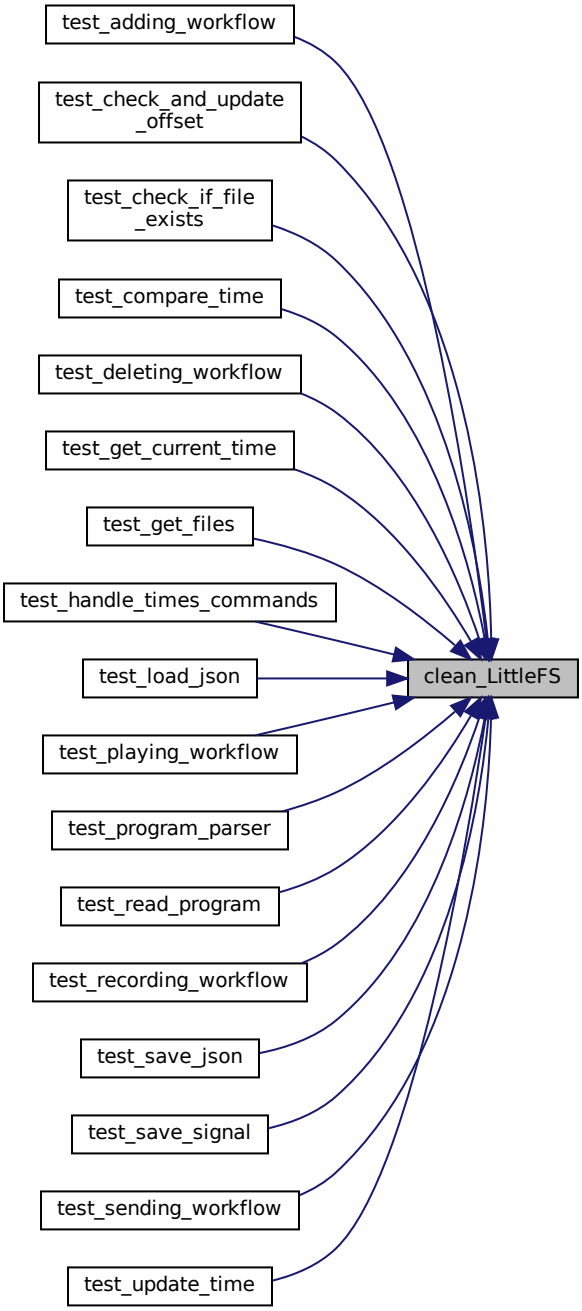
3.3.2 Function Documentation

3.3.2.1 clean_LittleFS()

`void clean_LittleFS ( )`

Deletes all files in "/", "/signals" and "/programs" in the LittleFS.

This function does not call any other function. Here is the caller graph for this function:



3.3.2.2 empirical_test_get_NTP_time()

```
boolean empirical_test_get_NTP_time ( )
```

Tests empirically the function get_NTP_time()

Returns

boolean - true if the test passed, false if the test failed

- Setup: This test connects to the mobile hotspot of a phone. (didn't want to write my wifi credentials here)

1. request NTP time and check if the init_offset is correct (error should be less than 1000ms)
2. request NTP time 100 times after random time intervals and check if the time is equal to the expected time (error should be less than 1000ms due to rounding errors)

3.3.2.3 run_all_empirical_tests()

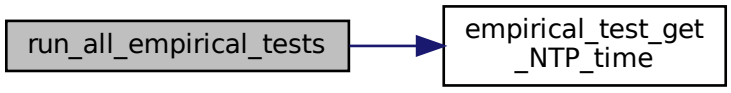
```
void run_all_empirical_tests (
            boolean stop_on_error )
```

runs all empirical tests

Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

Here is the call graph for this function:



3.3.2.4 run_all_filesystem_tests()

```
boolean run_all_filesystem_tests (
            boolean stop_on_error )
```

runs all tests for filesystem.cpp

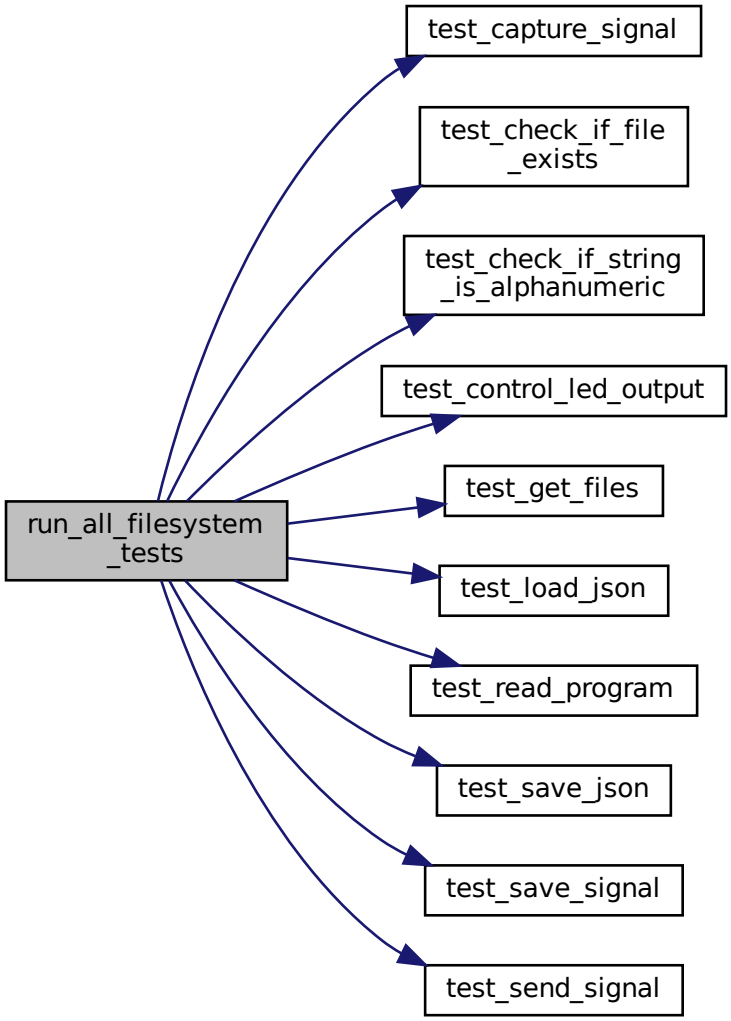
Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

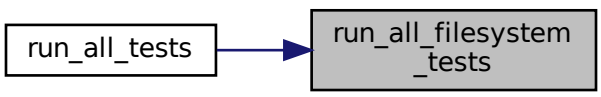
Returns

boolean - true if all tests passed, false if at least one test failed

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.2.5 run_all_tests()

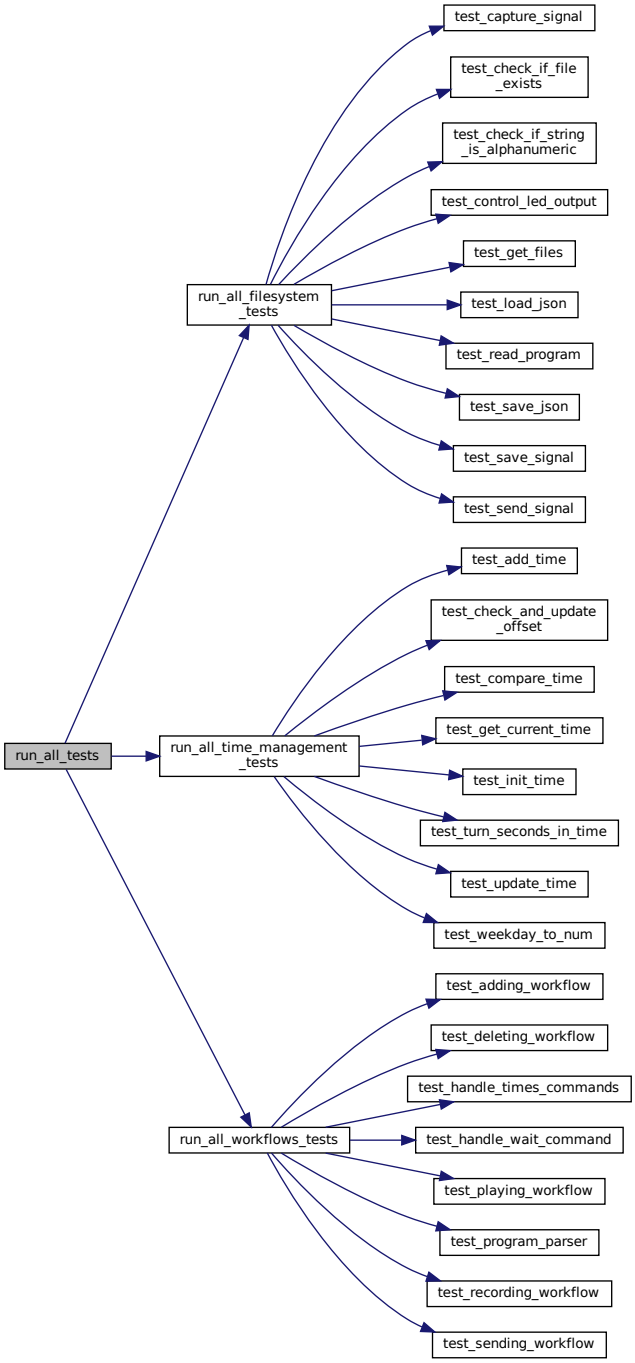
```
void run_all_tests (
            boolean stop_on_error )
```

runs all tests for all files

Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

Here is the call graph for this function:



3.3.2.6 run_all_time_management_tests()

```
boolean run_all_time_management_tests (
            boolean stop_on_error )
```

runs all tests for time_management.cpp

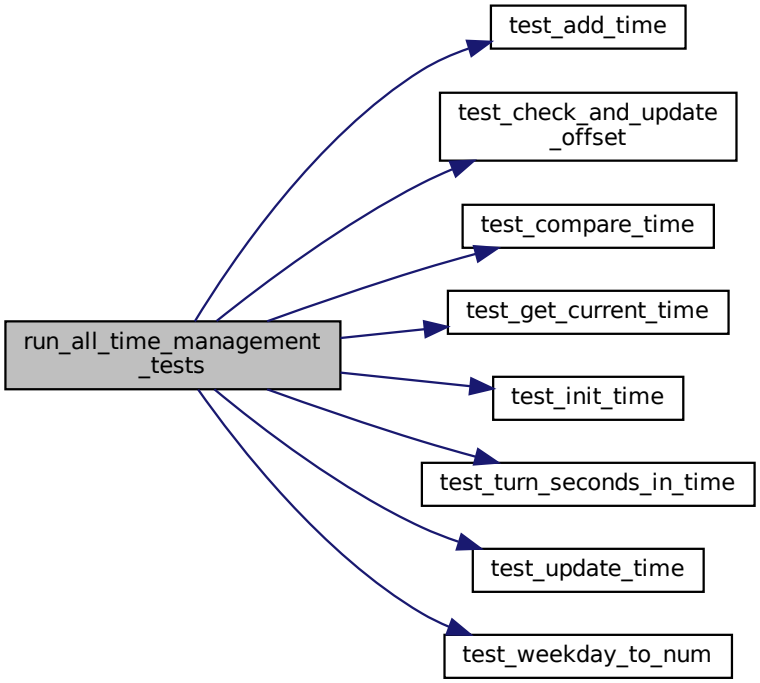
Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

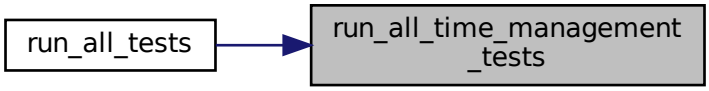
Returns

boolean - true if all tests passed, false if at least one test failed

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.2.7 run_all_workflows_tests()

```
boolean run_all_workflows_tests (
            boolean stop_on_error )
```

runs all tests for workflows.cpp

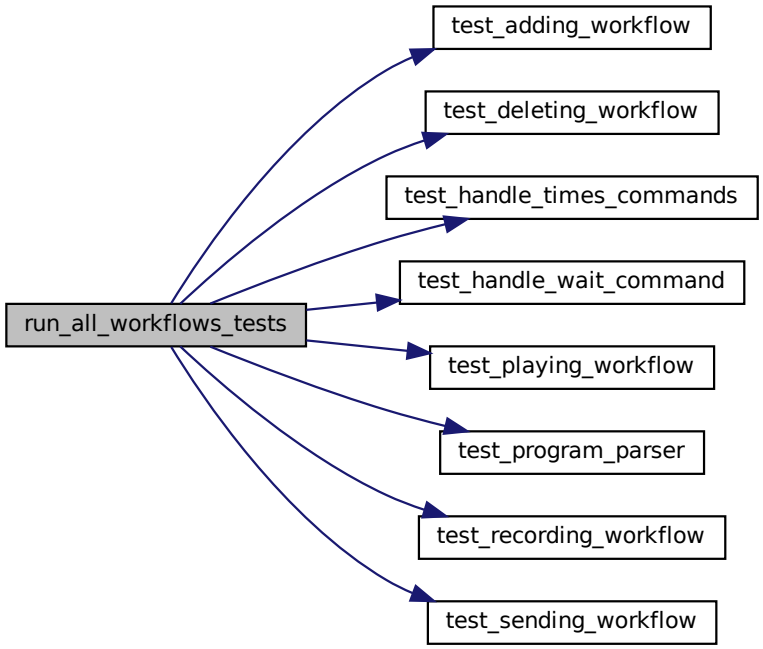
Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

Returns

boolean - true if all tests passed, false if at least one test failed

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.2.8 test_add_time()

```
boolean test_add_time ( )
```

Unit test for the function "add_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks samples of timestamps

See also

add_time

3.3.2.9 test_adding_workflow()

```
boolean test_adding_workflow ( )
```

Unit test for the function "adding_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS

1. check if functions return value is correct
2. check if program code is correctly written to file

See also

adding_workflow

3.3.2.10 test_capture_signal()

```
boolean test_capture_signal ( )
```

Unit test for the function "capture_signal".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. check if return value is correct (no random noise signal is received)
2. check if execution time is normal (1s more is acceptable)

See also

capture_signal

3.3.2.11 test_check_and_update_offset()

```
boolean test_check_and_update_offset ( )
```

Unit test for the function "check_and_update_offset".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and initialize time.json with test data

1. check if the offset is updated correctly

See also

check_and_update_offset

3.3.2.12 test_check_if_file_exists()

```
boolean test_check_if_file_exists ( )
```

Unit test for the function "check_if_file_exists".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and create test file in LittleFS

1. checks existing file is found
2. checks non-existing file is not found

See also

check_if_file_exists

3.3.2.13 test_check_if_string_is_alphanumeric()

```
boolean test_check_if_string_is_alphanumeric ( )
```

Unit test for the function "test_check_if_string_is_alphanumeric".

Returns

boolean - true if the test passed, false if the test failed

-Setup: -

1. checks if sample Strings are recognized correctly

See also

test_check_if_string_is_alphanumeric

3.3.2.14 test_compare_time()

```
boolean test_compare_time ( )
```

Unit test for the function "compare_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and initialize time.json with test data

1. checks with loop if true is returned when times match
2. checks if false is returned when times do not match

See also

compare_time

3.3.2.15 test_control_led_output()

```
boolean test_control_led_output ( )
```

Unit test for the function "test_control_led_output".

Returns

boolean - true if the test passed, false if the test failed

The functionality of this function is manually tested.

See also

test_control_led_output

3.3.2.16 test_deleting_workflow()

```
boolean test_deleting_workflow ( )
```

Unit test for the function "deleting_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create test signal file

1. check if function returns correct error message when file does not exist
2. check if no other file is deleted
3. check if file is deleted correctly
4. check again if no other file is deleted

See also

deleting_workflow

3.3.2.17 test_get_current_time()

```
boolean test_get_current_time ( )
```

Unit test for the function "get_current_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and initialize time.json with test data

1. checks twice if time is returned correctly

See also

get_current_time

3.3.2.18 test_get_files()

```
boolean test_get_files ( )
```

Unit test for the function "get_files".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and create test files in LittleFS

1. checks if files are returned correctly
2. checks if no files are returned if no files exist

See also

get_files

3.3.2.19 test_handle_times_commands()

```
boolean test_handle_times_commands ( )
```

Unit test for the function "handle_times_commands".

Returns

boolean - true if the test passed, false if the test failed

- Setup: create test signal in LittleFS

1. check error message if weekday is invalid
2. check if invalid commands are caught

See also

handle_times_commands

3.3.2.20 test_handle_wait_command()

```
boolean test_handle_wait_command ( )
```

Unit test for the function "handle_wait_command".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. check if return value is "success"
2. check if set amount of time is waited
3. check if function waited too long

See also

handle_wait_command

3.3.2.21 test_init_time()

```
boolean test_init_time ( )
```

Unit test for the function "get_NTP_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks if the document is empty (since Wifi is not available)
2. functionality is tested empirically

See also

get_NTP_time

3.3.2.22 test_load_json()

```
boolean test_load_json ( )
```

Unit test for the function "load_json".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS

1. checks if data is correctly loaded from file
2. checks if empthy JSON Doc is returned if file does not exist
3. checks if empthy JSON Doc is returned if file is empthy
4. checks if empthy JSON Doc is returned if file is not in JSON format

See also

load_json

3.3.2.23 test_playing_workflow()

```
boolean test_playing_workflow ( )
```

Unit test for the function "playing_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create 2 test programs in LittleFS

1. check if functions return value is correct when file does not exist
2. check if program can be correctly executed
3. check if error messages of program_parser are shown correctly

See also

playing_workflow

3.3.2.24 test_program_parser()

```
boolean test_program_parser ( )
```

Unit test for the function "program_parser".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create 2 test signals in LittleFS

1. check if program can be executed correctly
2. if error message is correct when program is faulty
3. if error message is correct when signal does not exist

See also

program_parser

3.3.2.25 test_read_program()

```
boolean test_read_program ( )
```

Unit test for the function "read_program".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and create test program in LittleFS

1. checks if program is read correctly
2. checks if empthy string is returned if program does not exist

See also

read_program

3.3.2.26 test_recording_workflow()

```
boolean test_recording_workflow ( )
```

Unit test for the function "recording_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS

1. check if error message is correct when nothing was recorded
2. check if no file is written when nothing was recorded

See also

recording_workflow

3.3.2.27 test_save_json()

```
boolean test_save_json ( )
```

Unit test for the function "save_json".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS

1. checks if file is created
2. checks if JSON-Document is correctly written to file
3. checks if file is overwritten if it already exists

- (no check if filename or data is correct (this is checked by higher level))

See also

save_json

3.3.2.28 test_save_signal()

```
boolean test_save_signal ( )
```

Unit test for the function "save_signal".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS

1. checks if formatting of name is considered
2. checks if formatting of result_string is considered

See also

save_signal

3.3.2.29 test_send_signal()

```
boolean test_send_signal ( )
```

Unit test for the function "send_signal".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks if signal is sent correctly
2. checks if JSON Doc with invalid length is not accepted
3. checks if JSON Doc with without sequence is not accepted
4. checks if JSON Doc with without length is not accepted

See also

send_signal

3.3.2.30 test_sending_workflow()

```
boolean test_sending_workflow ( )
```

Unit test for the function "sending_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create 2 test signals in LittleFS

1. check if error message is correct when file does not exist
2. check if sequence can be correctly sent
3. check if error messages of send_signal are shown correctly (1 example)

See also

sending_workflow

3.3.2.31 test_turn_seconds_in_time()

```
boolean test_turn_seconds_in_time ( )
```

Unit test for the function "turn_seconds_in_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks samples of conversions

See also

turn_seconds_in_time

3.3.2.32 test_update_time()

```
boolean test_update_time ( )
```

Unit test for the function "update_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and initialize time.json with test data

1. checks if the time is updated correctly in Station mode
2. checks if the time is updated correctly in AP mode

See also

update_time

3.3.2.33 test_weekday_to_num()

```
boolean test_weekday_to_num ( )
```

Unit test for the function "weekday_to_num".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks if the correct number for each weekday is returned
2. checks if "error" is returned if the weekday is not valid

See also

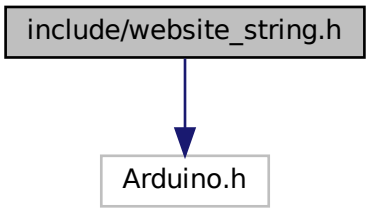
weekday_to_num

3.4 include/website_string.h File Reference

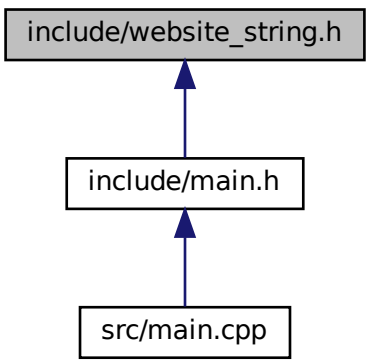
Header file that holds the website as a string.

```
#include <Arduino.h>
```

Include dependency graph for website_string.h:



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



Variables

- const char index_html[ ] PROGMEM

  The website as a string.

3.4.1 Detailed Description

Header file that holds the website as a string.

Author

Marc Ubbelohde

This file holds the website as a string. It is used to reduce the amount of code in the main.h file. Additionally to this file, you can find the website in html format in website.hmtl.

3.4.2 Variable Documentation

3.4.2.1 PROGMEM

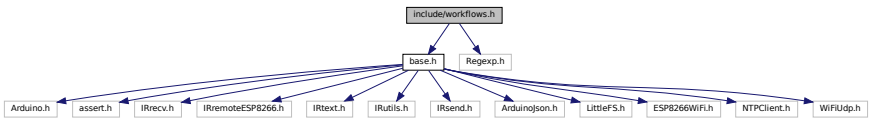
`const char index_html [] PROGMEM`
The website as a string.

3.5 include/workflows.h File Reference

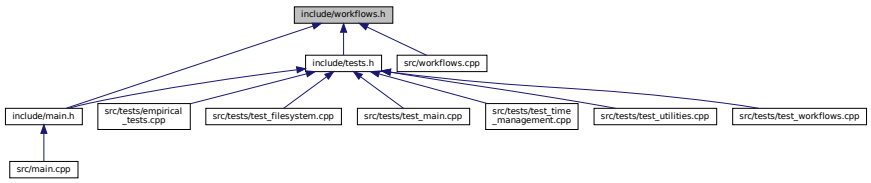
Header file for workflows.cpp.

```
#include "base.h"
#include "Regexp.h"
```

Include dependency graph for workflows.h:



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



Functions

- String deleting_workflow (String directory, String command_name)

  This function deletes a file from the LittleFS filesystem.
- String recording_workflow (String command_name)

  This function records and saves a signal.
- String sending_workflow (String command_name)

This function loads a signal from a file and sends it.

- String adding_workflow (String program_name, String program_code)

  This function creates a file with the programs name and writes the code to it.
- String playing_workflow (String program_name)

  This function loads a program from a file and hands it to the program_parser.
- String program_parser (String code)

  This function parses the code of a program line by line and executes the commands.
- String handle_wait_command (unsigned long waiting_time)

  This function waits a certain amount of time.
- String handle_times_commands (String command, boolean day_included)

  This function executes timed commands.

3.5.1 Detailed Description

Header file for workflows.cpp.

Author

Marc Ubbelohde

This file includes the dependencies for the workflows.cpp file which include the Regexp.h file and the base.h file.

3.5.2 Function Documentation

3.5.2.1 adding_workflow()

```
String adding_workflow (
            String program_name,
            String program_code )
```

This function creates a file with the programs name and writes the code to it.

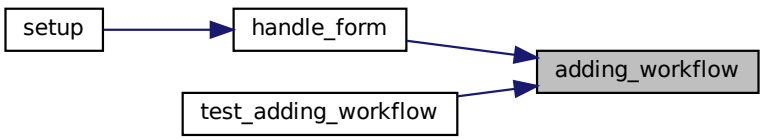
Parameters

program_name	- name of the program to be added
program_code	- code of the program to be added

Returns

String - message that will be displayed on the webpage:
"success message" - if file was created and code was written
"error message" - if file could not be created

Here is the caller graph for this function:



3.5.2.2 deleting_workflow()

```
String deleting_workflow (
            String directory,
            String name )
```

This function deletes a file from the LittleFS filesystem.

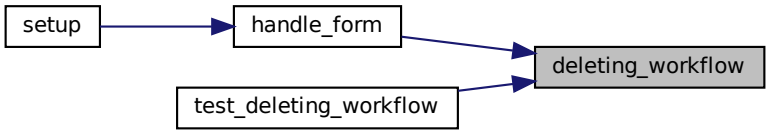
Parameters

directory	- "signals" or "programs"
name	- name of the sequence or program to be deleted

Returns

String - message that will be displayed on the webpage:
"success message" - if file was deleted
"error message" - if file could not be found

This function is used to delete signals and programs.
This function does not call other functions. Here is the caller graph for this function:



3.5.2.3 handle_times_commands()

```
String handle_times_commands (
            String command,
            boolean day_included )
```

This function executes timed commands.

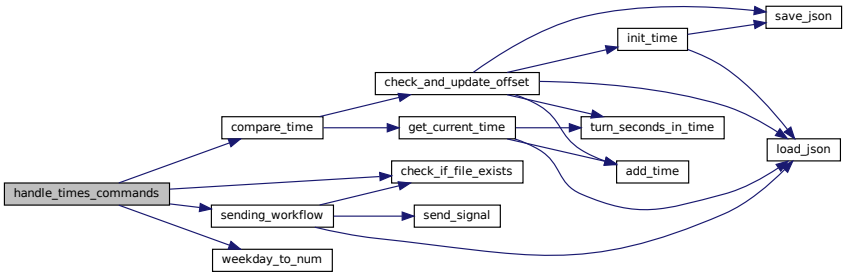
Parameters

command	- String command: "weekday hh:mm:ss signal_name" - if day_included is true "hh:mm:ss signal_name" - if day_included is false
day_included	- true if day is included in command, false if not

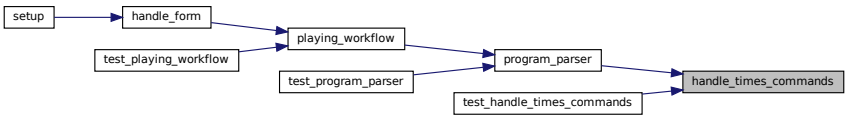
Returns

String - message that will be displayed on the webpage:
"success message" - if command was executed successfully

This function waits until a certain day and/or time is reached and then executes the given signal. Here is the call graph for this function:



Here is the caller graph for this function:



3.5.2.4 handle_wait_command()

```
String handle_wait_command (
            unsigned long waiting_time )
```

This function waits a certain amount of time.

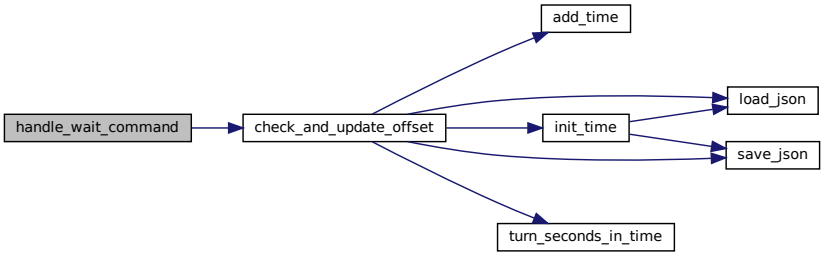
Parameters

waiting_time	- time to wait in milliseconds

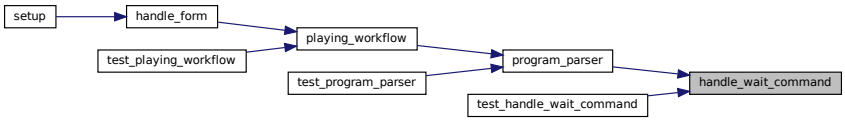
Returns

String - message that will be displayed on the webpage:
"success message" - if command was executed successfully
"error message" - if command was interrupted by the user

This function waits a certain amount of time. It is used for the wait and skip command. It is necessary to check beforehand if a millis() overflow will occur during the waiting time. If an overflow will occur, the function will first calculate the time it will have to wait after the overflow occurs, then it will wait until the overflow occurs and waits the remaining time. The function also checks if the user pressed the interrupt button. Here is the call graph for this function:



Here is the caller graph for this function:



3.5.2.5 playing_workflow()

```
String playing_workflow (
            String program_name )
```

This function loads a program from a file and hands it to the program_parser.

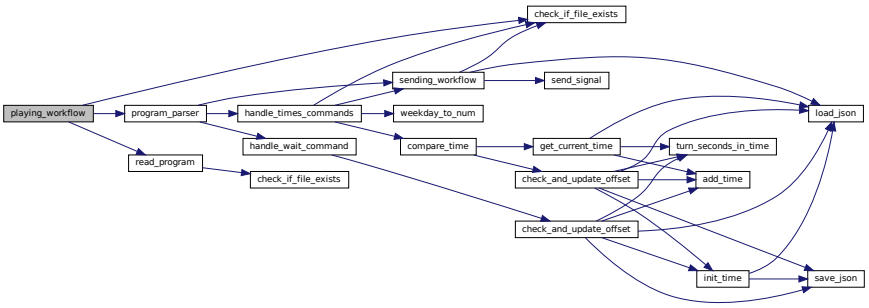
Parameters

program_name	- name of the program to be played

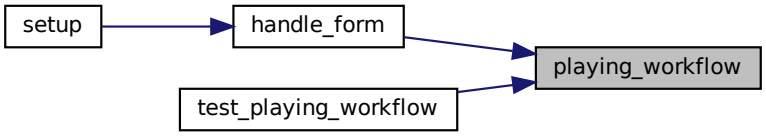
Returns

String - message that will be displayed on the webpage:
"success message" - if file was found and program was played successfully
"error message" - if file could not be found or if in one of the commands an error occured (error message gets passed by program_parser)

This function loads a program from a file and hands it to the program_parser. The program_parser then sends the commands and returns a message when the execution of the program finished. Here is the call graph for this function:



Here is the caller graph for this function:



3.5.2.6 program_parser()

```
String program_parser (
            String code )
```

This function parses the code of a program line by line and executes the commands.

Parameters

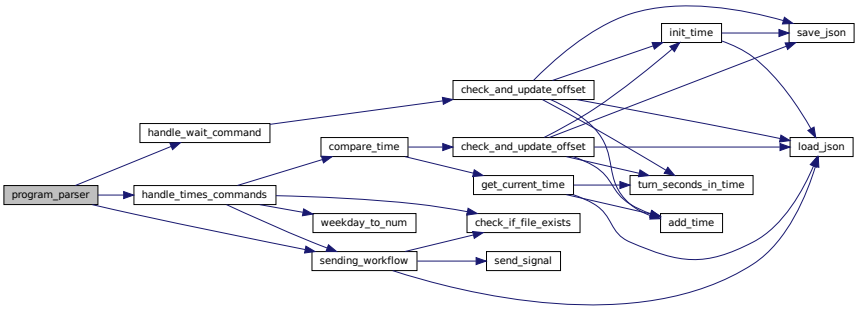
code	- code of the program to be parsed

Returns

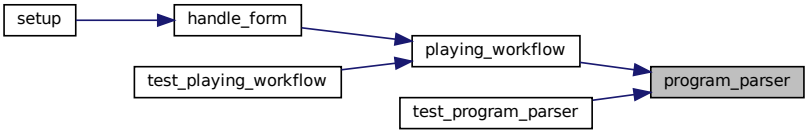
String - message that will be displayed on the webpage:
"success message" - if program was played successfully
"error message" - if in one of the commands an error occured an command specific error message is returned

This function parses the code of a program line by line and executes the commands. It was necessary to split the parser from the playing_workflow function to be able to call it recursively (for loops). Each line is searched for command specific keywords and the corresponding command handler is called. Here is the call graph for this

function:



Here is the caller graph for this function:



3.5.2.7 recording_workflow()

```
String recording_workflow (
            String signal_name )
```

This function records and saves a signal.

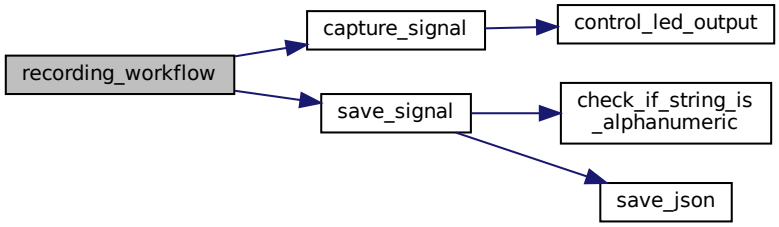
Parameters

signal_name	- name of the sequence to be recorded

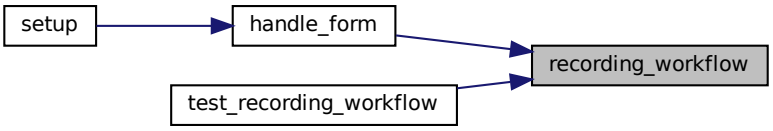
Returns

String - message that will be displayed on the webpage:
"success message" - if signal was saved
"error message" - no signal was captured

This function records a signal and saves it to a file in the LitteFS with the signals name. (spaces at the end of the signal name will be removed) Here is the call graph for this function:



Here is the caller graph for this function:



3.5.2.8 sending_workflow()

```
String sending_workflow (
            String signal_name )
```

This function loads a signal from a file and sends it.

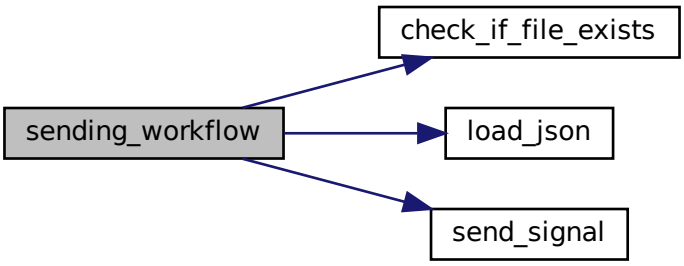
Parameters

signal_name	- name of the sequence to be sent

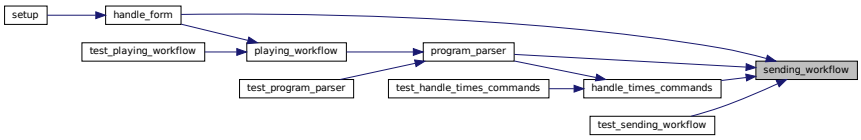
Returns

String - message that will be displayed on the webpage:
"success message" - if file was found and command was sent
"error message" - if file could not be found

Here is the call graph for this function:

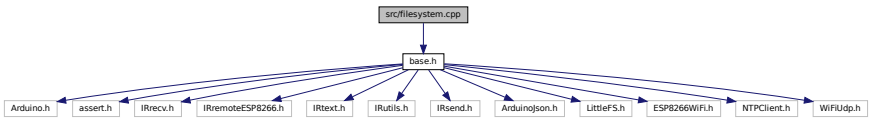


Here is the caller graph for this function:



3.6 src/filesystem.cpp File Reference

In this file, all functions related to the filesystem are defined.
`#include "base.h"`
Include dependency graph for filesystem.cpp:



Functions

- String capture_signal ()

  This function captures a signal and returns it as a String.
- String save_signal (String result_string, String name)

  This function saves a captured signal.
- void save_json (String filename, DynamicJsonDocument doc)

  This function saves a JSON document to a specified file.
- DynamicJsonDocument load_json (String filename)

This function loads a JSON document from a specified file.

- String send_signal (DynamicJsonDocument doc)

  This function sends a signal provided in JSON format.

- String get_files (String folder_signals, String folder_programs)

  Returns List of saved signals and programs.

- boolean check_if_file_exists (String filename)

  Checks if a file exists.

- String read_program (String program_name)

  Reads a program file and returns its content as a String.

- void control_led_output (String signal)

  Controls the LED output.

- boolean check_if_string_is_alphanumeric (String word)

  Checks if a String is alphanumeric.

3.6.1 Detailed Description

In this file, all functions related to the filesystem are defined.

Author

Marc Ubbelohde

The functions in this file are used to save and load data from the filesystem, provide the frontend with the data it needs to display necessary information, to handle signal capture and sending and to control the LED output. This file is the foundation of the project and the functions are used by almost all other files.

3.6.2 Function Documentation

3.6.2.1 capture_signal()

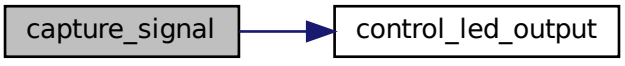
```
String capture_signal ( )
```

This function captures a signal and returns it as a String.

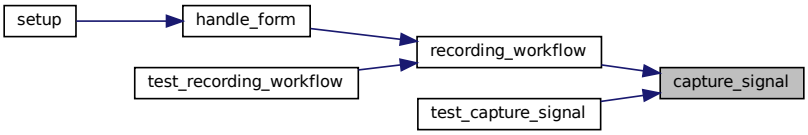
Returns

String - String containing the captured signal in the format:
"uint16_t rawData[67] = {1234, 5678, ...};" - if signal was receiver, format defined by resultToSourceCode() in IRremoteESP8266/src/IRutils.cpp
"no_signal" - if no signal was captured

This function uses the IRremoteESP8266 library and is based on the IRrecvDumpV2 example from the library. Here is the call graph for this function:



Here is the caller graph for this function:



3.6.2.2 check_if_file_exists()

```
boolean check_if_file_exists (
            String filename )
```

Checks if a file exists.

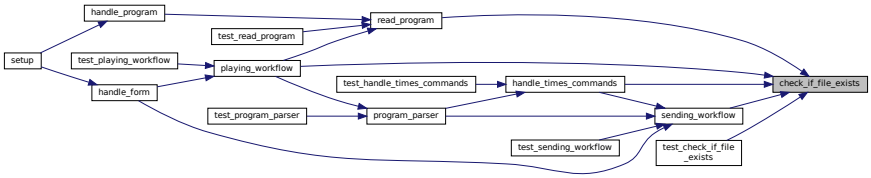
Parameters

filename	- name of the file to be checked

Returns

boolean - true if the file exists, false if not

This function checks if a file exists in the LittleFS.
This function does not call any other function. Here is the caller graph for this function:



3.6.2.3 check_if_string_is_alphanumeric()

```
boolean check_if_string_is_alphanumeric (
            String word )
```

Checks if a String is alphanumeric.

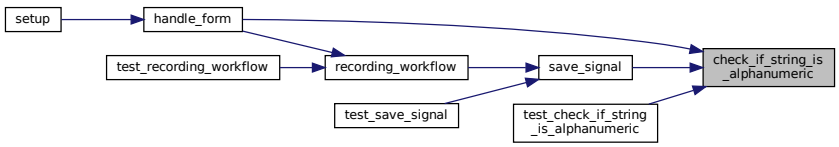
Parameters

word	- String to be checked

Returns

boolean - true if the String is alphanumeric, false if not

This function checks if a String is alphanumeric. Spaces, dashes and underscores are also allowed.
This function does not call any other function. Here is the caller graph for this function:



3.6.2.4 control_led_output()

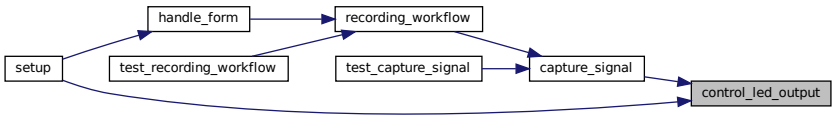
```
void control_led_output (
            String signal )
```

Controls the LED output.

Parameters

signal	- String containing the signal to be send: "no_signal" - LED blinks 3 times "no_mDNS" - LED blinks 3 times "signal_received" - LED blinks once

This function controls the LED output via codewords to specify the kind of signal to be send. The LED is connected to GPIO 5 (D1) on the ESP8266 and is ment as a way to communicate errors to the user and singal when the ESP is ready to receive a signal.
This function does not call any other function. Here is the caller graph for this function:



3.6.2.5 get_files()

```
String get_files (
            String folder_signals,
            String folder_programs )
```

Returns List of saved signals and programs.

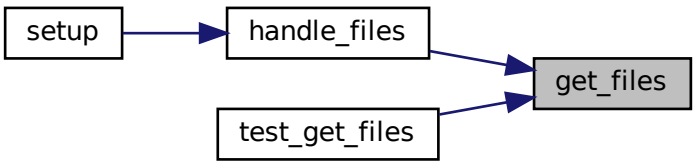
Parameters

folder_signals	- name of the folder containing the signals
folder_programs	- name of the folder containing the programs

Returns

String - String containing all files in the specified folders, separated by a semicolon:
"signal1, signal2, ...;program1, program2, ..."

This function scans the signal and program folders for files and returns a String containing all files. It uses LittleFS.
This function does not call any other function. Here is the caller graph for this function:



3.6.2.6 load_json()

```
DynamicJsonDocument load_json (
            String filename )
```

This function loads a JSON document from a specified file.

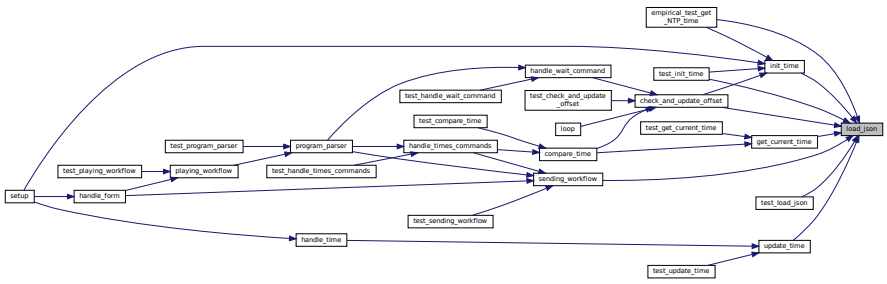
Parameters

filename	- name of the file

Returns

DynamicJsonDocument - JSON document containing unspecified data

This function uses LittleFS and the ArduinoJson library.
This function does not call any other function. Here is the caller graph for this function:



3.6.2.7 read_program()

```
String read_program (
            String program_name )
```

Reads a program file and returns its content as a String.

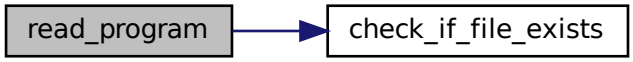
Parameters

program_name	- name of the program to be read

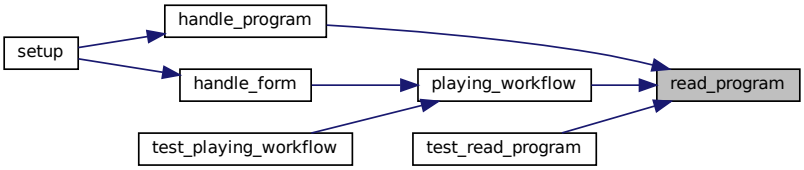
Returns

String - String containing the program code

This function has its reason of existence next to load_json because in the frontend, after pressing the "edit" button, the program code of the specified program is displayed in the textarea as a string.
This function does not call any other function. Here is the call graph for this function:



Here is the caller graph for this function:



3.6.2.8 save_json()

```
void save_json (
            String filename,
            DynamicJsonDocument doc )
```

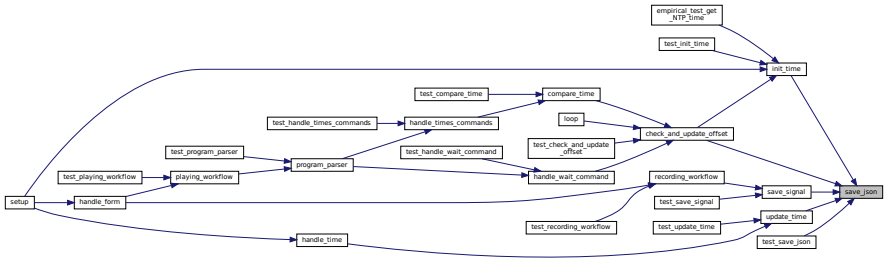
This function saves a JSON document to a specified file.

Parameters

filename	- name of the file
doc	- JSON document containing unspecified data

This function uses LittleFS and the ArduinoJson library.

This function does not call any other function. Here is the caller graph for this function:



3.6.2.9 save_signal()

```
String save_signal (
            String result_string,
            String name )
```

This function saves a captured signal.

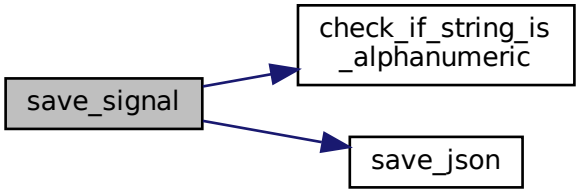
Parameters

result_string	- String containing the captured signal in the format: "uint16_t rawData[67] = {1234, 5678, ...};" (format defined by resultToSourceCode() in IRremoteESP8266/src/IRutils.cpp)
name	- name of the signal

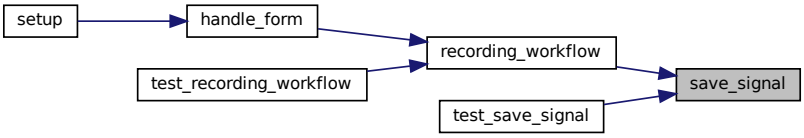
Returns

String - "success" - if signal was saved successfully
"Error: ..." - if an error occurred

This function saves a captured signal in json format to a file. It uses the ArduinoJson library. Here is the call graph for this function:



Here is the caller graph for this function:



3.6.2.10 send_signal()

```
String send_signal (
            DynamicJsonDocument doc )
```

This function sends a signal provided in JSON format.

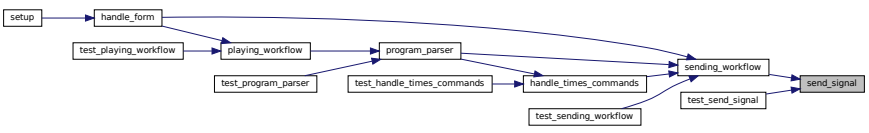
Parameters

doc	- JSON document containing the signal to be sent:
	{
	"name": "name",
	"length": 67,
	"sequence": "1234, 5678, ..."
	}

Returns

String - "success" if sending was successful
"Error: ..." if sending failed

This function sends a signal provided in JSON format. It uses the IRremoteESP8266 library and is based on the example code provided by the library.
This function does not call any other function. Here is the caller graph for this function:

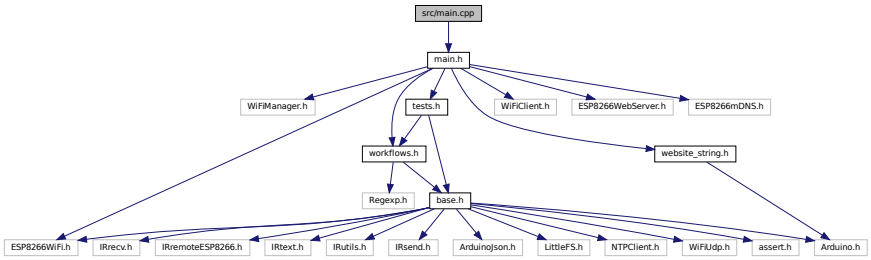


3.7 src/main.cpp File Reference

Main file of the program.

```
#include "main.h"
```

Include dependency graph for main.cpp:



Functions

- void setup ()

  Arduino Setup function.
- void loop ()

  Arduino Loop function.
- void handle_root ()

  Handler function for the root page.
- void handle_not_found ()

  Handler function for any page that is not found.
- void handle_program ()

  Handler function to display the program the user wants to edit.
- void handle_error ()

  Handler function to display the error message.
- void handle_files ()

  Handler function to send a list of all signals and programs to the forntend.
- void handle_time ()

  Handler funciton to synchronize the time and/or timezone provided by the user.
- void handle_credentials ()

  Handler function to erase the wifi credentials saved on the ESP.
- void handle_apmode ()

  Handler function to switch between AP mode and normal mode.
- void handle_apinfo ()

  Handler function to send the current AP mode setting to the frontend.
- void handle_password ()

  Handler function that receives new password entries.
- void handle_form ()

  Handler function that receives GET requests from all form elements related to signals and programs.

3.7.1 Detailed Description

Main file of the program.

Author

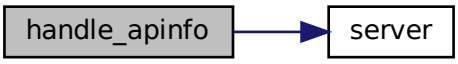
Marc Ubbelohde

In this file, you find the main procedures of the program and the handler functions for the webserver.

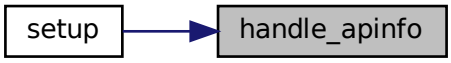
3.7.2 Function Documentation

3.7.2.1 handle_apinfo()

`void handle_apinfo ( )`
Handler function to send the current AP mode setting to the frontend.
Sends the current AP mode setting to the frontend. This is used to display the correct setting at the top the website.
This function is called on a GET request to /apinfo. Here is the call graph for this function:

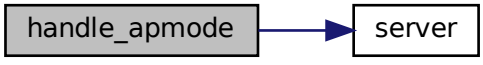


Here is the caller graph for this function:

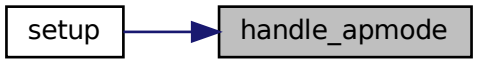


3.7.2.2 handle_apmode()

`void handle_apmode ( )`
Handler function to switch between AP mode and normal mode.
Checks the config file and switches between AP mode and normal mode. The possibel reconfiguration of the wifi credentails is done after restart.
This function is called on a GET request to /apmode. Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.3 handle_credentials()

```
void handle_credentials ( )
```

Handler function to erase the wifi credentials saved on the ESP.
Erases the wifi credentials saved on the ESP. This is useful if the user wants to connect to a different wifi network. This function is called on a GET request to /credentials. Here is the call graph for this function:

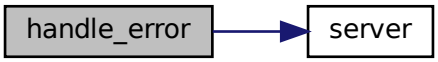
Here is the caller graph for this function:

3.7.2.4 handle_error()

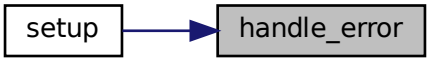
```
void handle_error ( )
```

Handler function to display the error message.
Similarly to handle_program(), the website will send a get request on /error to update the error message on reload. The MESSAGE is then written again by handle_form().

This function is called on a GET request to /error. Here is the call graph for this function:



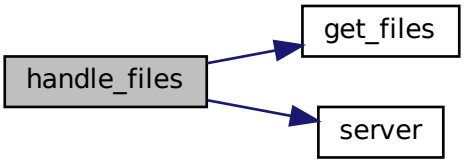
Here is the caller graph for this function:



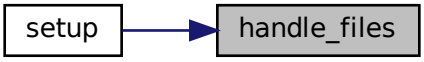
3.7.2.5 handle_files()

```
void handle_files ( )
```

Handler function to send a list of all signals and programs to the forntend.
Sends a list of all files in /signals and /programs on reload to be displayed on the website.
This function is called on a GET request to /files. Here is the call graph for this function:



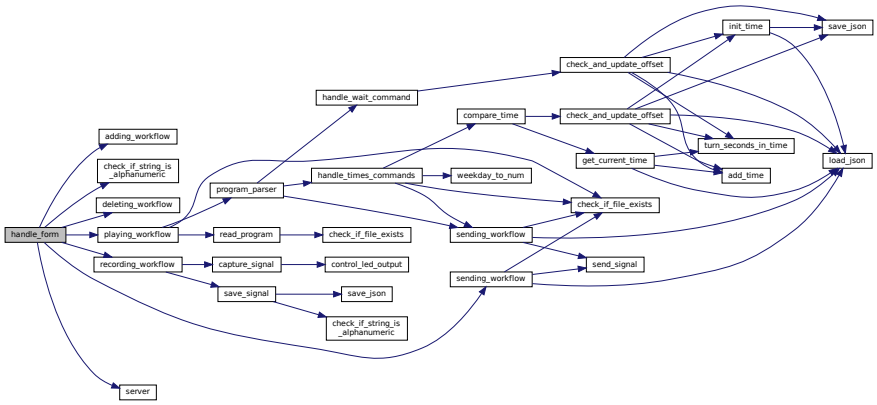
Here is the caller graph for this function:



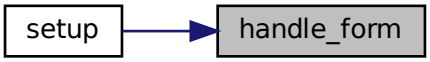
3.7.2.6 handle_form()

`void handle_form ( )`

Handler function that receives GET requests from all form elements related to signals and programs.
Handles all form elements on the website (signals and programs) also updates the error message and PROGRA↩
MNAME. All the user interaction with the website is handled here and the functions from workflows.h are called.
This function is called on a GET request to /form. Here is the call graph for this function:



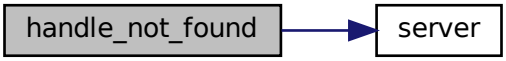
Here is the caller graph for this function:



3.7.2.7 handle_not_found()

`void handle_not_found ( )`

Handler function for any page that is not found.
This function is called when a page is requested that is not found. It serves a 404 error to the client.
This function is called on a GET request to a page that is not found. Here is the call graph for this function:

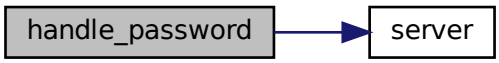


Here is the caller graph for this function:

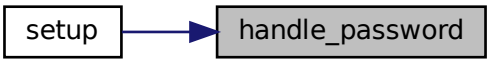
3.7.2.8 handle_password()

```
void handle_password ( )
```

Handler function that receives new password entries.
Receives new password entries from frontend and changes password if entries are the same (to prevent typos).
This function is called on a GET request to /password. Here is the call graph for this function:



Here is the caller graph for this function:

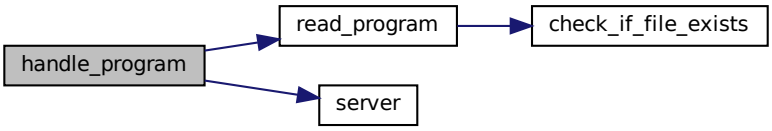


3.7.2.9 handle_program()

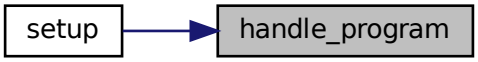
```
void handle_program ( )
```

Handler function to display the program the user wants to edit.
This function is called when the user wants to edit a program. It reads the program from the file system and sends it back to the website. The website is designed with form elements that trigger forwarding to /form. This is a problem because we can only communicate on that channel and we already have to communicate a back to the website to update the page. So in order to still be able to send data back to the website, on every reload the website will send a get request on /program which triggers this function. The data that is sent back is the name of the currently selected program and the code of that program. It is only sent back if the "edit" button was pressed. (if not, the variables "PROGRAMNAME" and "PROGRAMCODE" will be empty)

This function is called on a GET request to /program. Here is the call graph for this function:



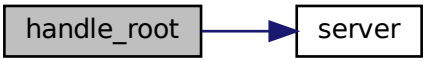
Here is the caller graph for this function:



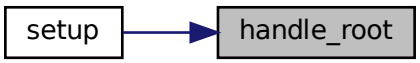
3.7.2.10 handle_root()

```
void handle_root ( )
```

Handler function for the root page.
This function is called when the root page is requested. It serves the content of the index_html string from the website_string.h file to the client.
This function is called on a GET request to the root page. Here is the call graph for this function:



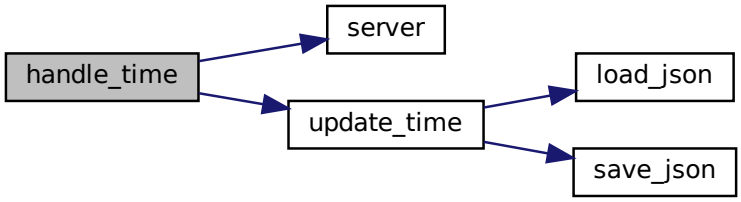
Here is the caller graph for this function:



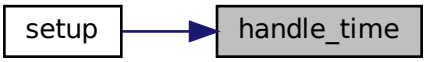
3.7.2.11 handle_time()

```
void handle_time ( )
```

Handler funciton to synchronize the time and/or timezone provided by the user.
Gets the time from the client via Date() and saves it together with the millis() offset to the LittleFS. The offset is important because only with millis() we can calculate the time that has passed between the synchronisation and the time of program execution. This enabled the ESP to execute timed programs even if the wifi connection is lost.
This function is called on a GET request to /time. Here is the call graph for this function:



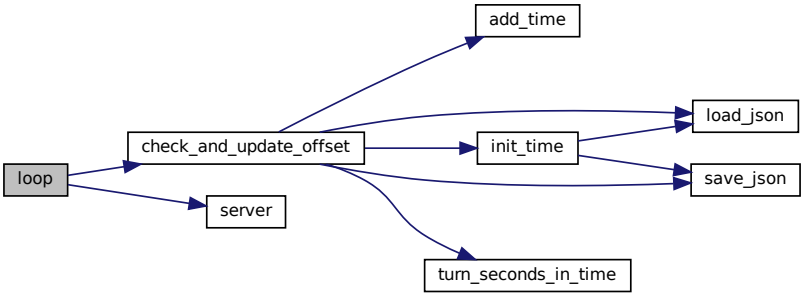
Here is the caller graph for this function:



3.7.2.12 loop()

```
void loop ( )
```

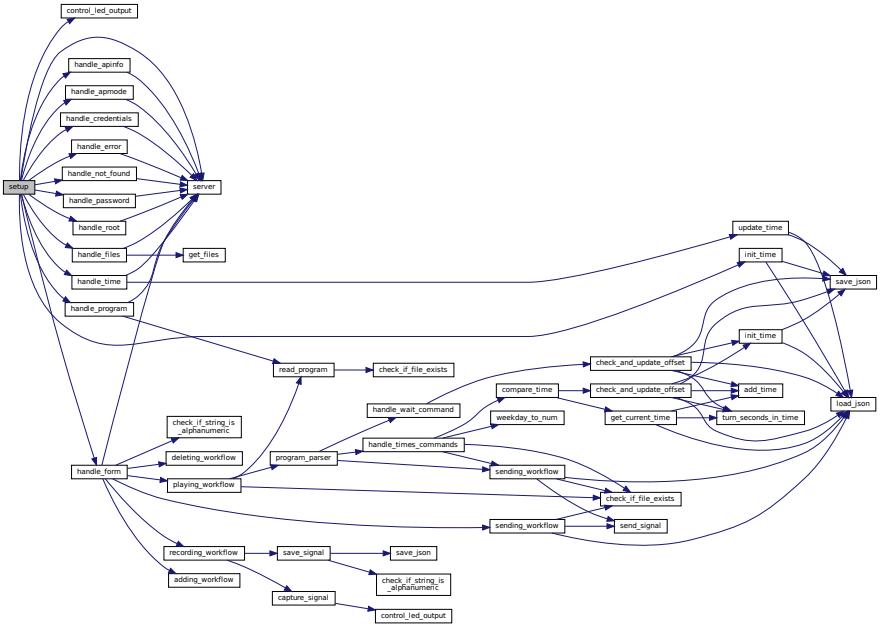
Arduino Loop function.
This function is called repeatedly. It updates the mDNS, handles clients and checks for a millis() overflow every 5 minutes.
This function is called by the Arduino framework. Here is the call graph for this function:



3.7.2.13 setup()

```
void setup ( )
```

Arduino Setup function.
This function is called once at the start of the program. It checks the configuration file and start either the Access Point or the WFiManager. It also starts the webserver and initializes the time file. Optionally, unit tests can be run.
This function is called by the Arduino framework. Here is the call graph for this function:

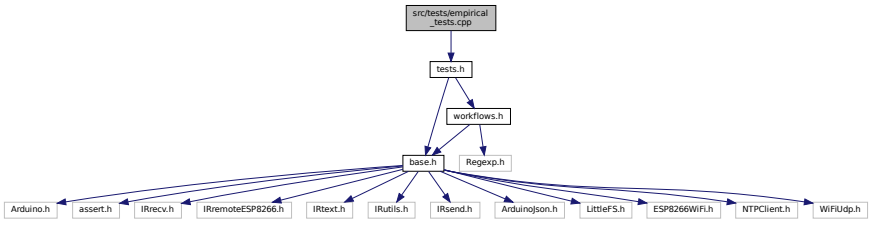


3.8 src/tests/empirical_tests.cpp File Reference

collection of empirical tests

```
#include "tests.h"
```

Include dependency graph for empirical_tests.cpp:



Functions

- boolean empirical_test_get_NTP_time ()

  Tests empirically the function get_NTP_time()

3.8.1 Detailed Description

collection of empirical tests

Author

Marc Ubbelohde

In this file you can find all emprical tests. This tests are used to varify the functionality of parts of the code that are not easily testable with common unit tests because they rely on internet connection or other external factors.

3.8.2 Function Documentation

3.8.2.1 empirical_test_get_NTP_time()

```
boolean empirical_test_get_NTP_time ( )
```

Tests empirically the function get_NTP_time()

Returns

boolean - true if the test passed, false if the test failed

- Setup: This test connects to the mobile hotspot of a phone. (didn't want to write my wifi credentials here)

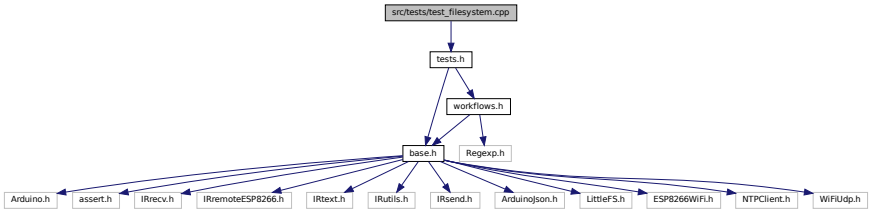
1. request NTP time and check if the init_offset is correct (error should be less than 1000ms)
2. request NTP time 100 times after random time intervals and check if the time is equal to the expected time (error should be less than 1000ms due to rounding errors)

3.9 src/tests/test_filesystem.cpp File Reference

This file contains unit tests for all functions from the filesystem.cpp.

```
#include "tests.h"
```

Include dependency graph for test_filesystem.cpp:



Functions

- boolean test_capture_signal ()

  Unit test for the function "capture_signal".
- boolean test_save_signal ()

  Unit test for the function "save_signal".
- boolean test_save_json ()

  Unit test for the function "save_json".
- boolean test_load_json ()

  Unit test for the function "load_json".
- boolean test_send_signal ()

  Unit test for the function "send_signal".
- boolean test_get_files ()

  Unit test for the function "get_files".

- boolean test_check_if_file_exists ()

  Unit test for the function "check_if_file_exists".
- boolean test_read_program ()

  Unit test for the function "read_program".
- boolean test_control_led_output ()

  Unit test for the function "test_control_led_output".
- boolean test_check_if_string_is_alphanumeric ()

  Unit test for the function "test_check_if_string_is_alphanumeric".

3.9.1 Detailed Description

This file contains unit tests for all functions from the filesystem.cpp.

Author

Marc Ubbelohde

3.9.2 Function Documentation

3.9.2.1 test_capture_signal()

```
boolean test_capture_signal ( )
```

Unit test for the function "capture_signal".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. check if return value is correct (no random noise signal is received)
2. check if execution time is normal (1s more is acceptable)

See also

capture_signal

3.9.2.2 test_check_if_file_exists()

```
boolean test_check_if_file_exists ( )
```

Unit test for the function "check_if_file_exists".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and create test file in LittleFS

1. checks existing file is found
2. checks non-existing file is not found

See also

check_if_file_exists

3.9.2.3 test_check_if_string_is_alphanumeric()

```
boolean test_check_if_string_is_alphanumeric ( )
```

Unit test for the function "test_check_if_string_is_alphanumeric".

Returns

boolean - true if the test passed, false if the test failed

-Setup: -

1. checks if sample Strings are recognized correctly

See also

test_check_if_string_is_alphanumeric

3.9.2.4 test_control_led_output()

```
boolean test_control_led_output ( )
```

Unit test for the function "test_control_led_output".

Returns

boolean - true if the test passed, false if the test failed

The functionality of this function is manually tested.

See also

test_control_led_output

3.9.2.5 test_get_files()

```
boolean test_get_files ( )
```

Unit test for the function "get_files".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and create test files in LittleFS

1. checks if files are returned correctly
2. checks if no files are returned if no files exist

See also

get_files

3.9.2.6 test_load_json()

```
boolean test_load_json ( )
```

Unit test for the function "load_json".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS

1. checks if data is correctly loaded from file
2. checks if empthy JSON Doc is returned if file does not exist
3. checks if empthy JSON Doc is returned if file is empthy
4. checks if empthy JSON Doc is returned if file is not in JSON format

See also

load_json

3.9.2.7 test_read_program()

```
boolean test_read_program ( )
```

Unit test for the function "read_program".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and create test program in LittleFS

1. checks if program is read correctly
2. checks if empthy string is returned if program does not exist

See also

read_program

3.9.2.8 test_save_json()

```
boolean test_save_json ( )
```

Unit test for the function "save_json".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS

1. checks if file is created
2. checks if JSON-Document is correctly written to file
3. checks if file is overwritten if it already exists

- (no check if filename or data is correct (this is checked by higher level))

See also

save_json

3.9.2.9 test_save_signal()

```
boolean test_save_signal ( )
```

Unit test for the function "save_signal".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS

1. checks if formatting of name is considered
2. checks if formatting of result_string is considered

See also

save_signal

3.9.2.10 test_send_signal()

```
boolean test_send_signal ( )
```

Unit test for the function "send_signal".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks if signal is sent correctly
2. checks if JSON Doc with invalid length is not accepted
3. checks if JSON Doc with without sequence is not accepted
4. checks if JSON Doc with without length is not accepted

See also

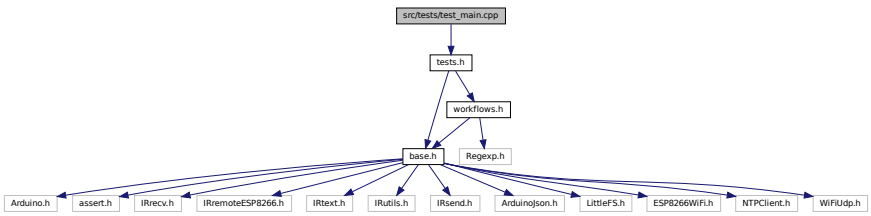
send_signal

3.10 src/tests/test_main.cpp File Reference

main file for all tests

```
#include "tests.h"
```

Include dependency graph for test_main.cpp:



Functions

- boolean run_all_filesystem_tests (boolean stop_on_error)

  runs all tests for filesystem.cpp
- boolean run_all_time_management_tests (boolean stop_on_error)

  runs all tests for time_management.cpp
- boolean run_all_workflows_tests (boolean stop_on_error)

  runs all tests for workflows.cpp
- void run_all_tests (boolean stop_on_error)

  runs all tests for all files
- void run_all_empirical_tests (boolean stop_on_error)

  runs all empirical tests

3.10.1 Detailed Description

main file for all tests

Author

Marc Ubbelohde

Here you find the functions that execute all tests for each file and finally a function that executes all tests for all files.

3.10.2 Function Documentation

3.10.2.1 run_all_empirical_tests()

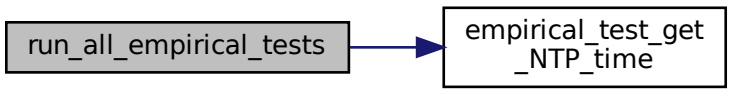
```
void run_all_empirical_tests (
            boolean stop_on_error )
```

runs all empirical tests

Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

Here is the call graph for this function:



3.10.2.2 run_all_filesystem_tests()

```
boolean run_all_filesystem_tests (
            boolean stop_on_error )
```

runs all tests for filesystem.cpp

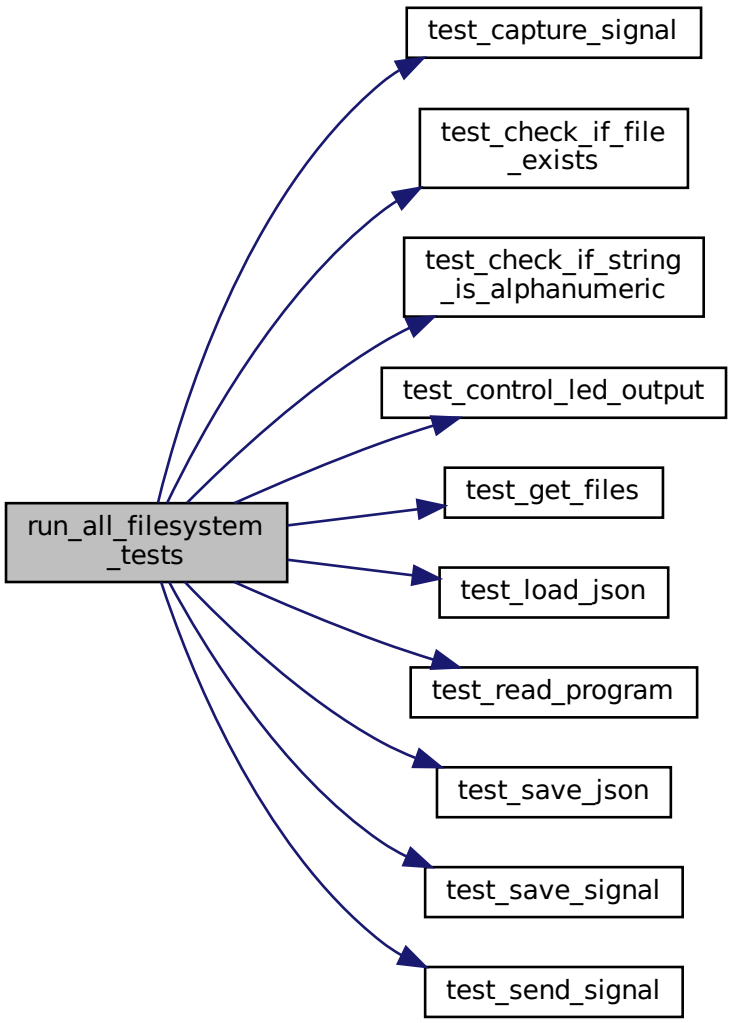
Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

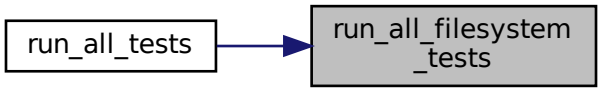
Returns

boolean - true if all tests passed, false if at least one test failed

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.2.3 run_all_tests()

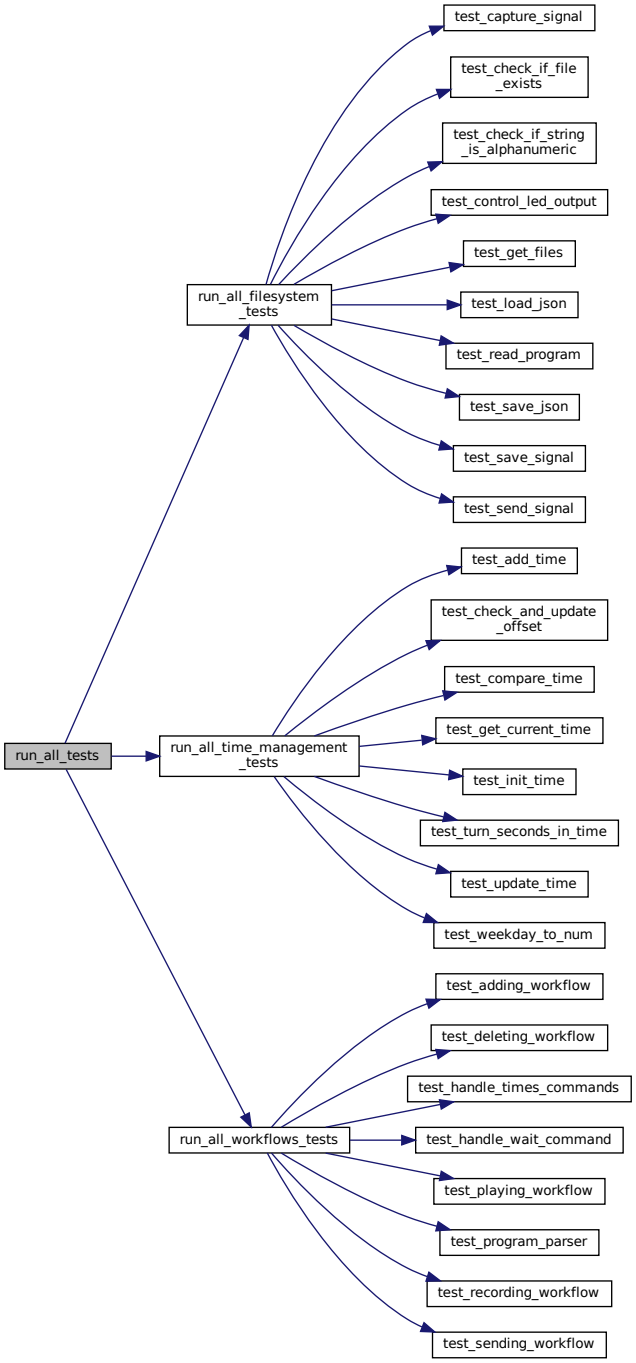
```
void run_all_tests (
            boolean stop_on_error )
```

runs all tests for all files

Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

Here is the call graph for this function:



3.10.2.4 run_all_time_management_tests()

```
boolean run_all_time_management_tests (
            boolean stop_on_error )
```

runs all tests for time_management.cpp

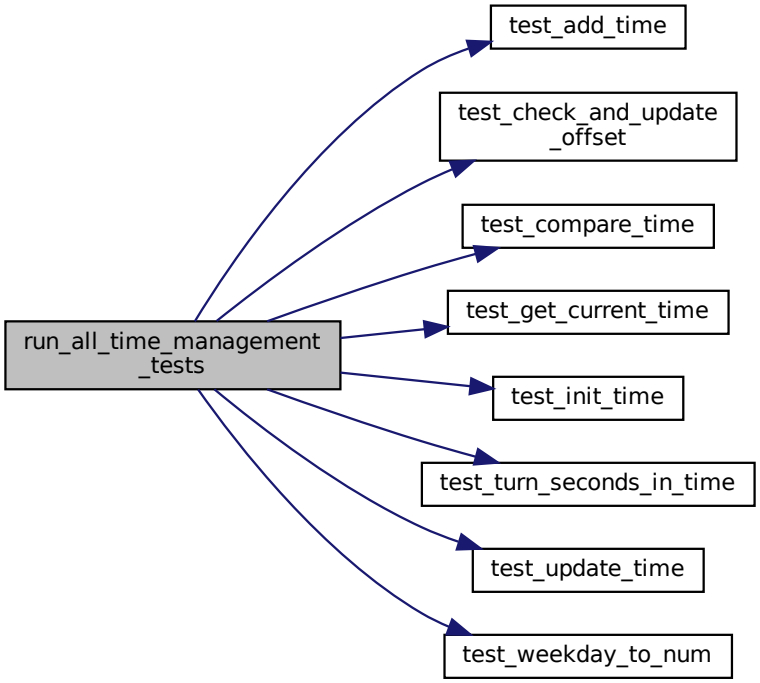
Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

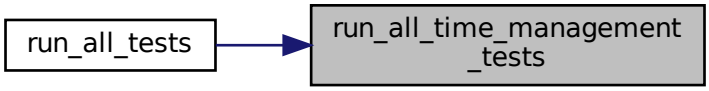
Returns

boolean - true if all tests passed, false if at least one test failed

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.2.5 run_all_workflows_tests()

```
boolean run_all_workflows_tests (
            boolean stop_on_error )
```

runs all tests for workflows.cpp

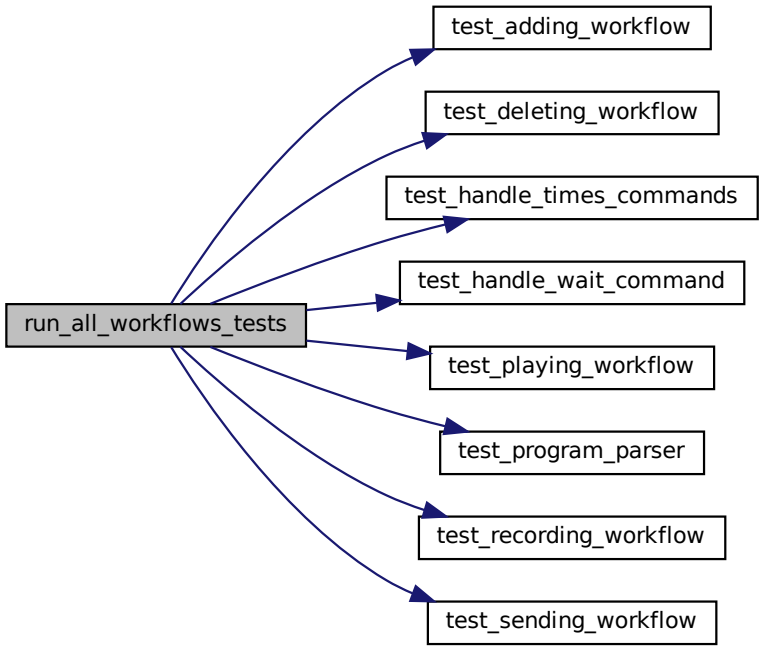
Parameters

stop_on_error	- if true, the function stops after the first failed test if false, the function continues to run all following tests

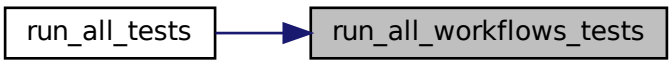
Returns

boolean - true if all tests passed, false if at least one test failed

Here is the call graph for this function:



Here is the caller graph for this function:

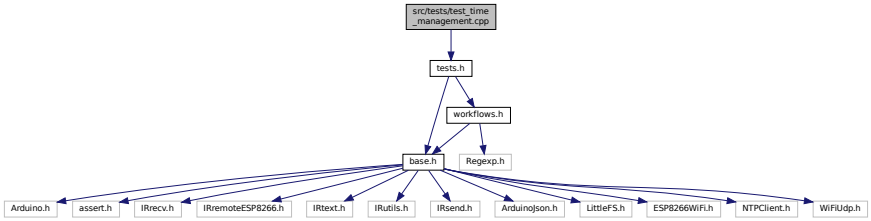


3.11 src/tests/test_time_management.cpp File Reference

This file contains unit tests for all functions from the time_management.cpp.

```
#include "tests.h"
```

Include dependency graph for test_time_management.cpp:



Functions

- boolean test_weekday_to_num ()

  Unit test for the function "weekday_to_num".
- boolean test_compare_time ()

  Unit test for the function "compare_time".
- boolean test_update_time ()

  Unit test for the function "update_time".
- boolean test_get_current_time ()

  Unit test for the function "get_current_time".
- boolean test_turn_seconds_in_time ()

  Unit test for the function "turn_seconds_in_time".
- boolean test_add_time ()

  Unit test for the function "add_time".
- boolean test_init_time ()

  Unit test for the function "get_NTP_time".
- boolean test_check_and_update_offset ()

  Unit test for the function "check_and_update_offset".

3.11.1 Detailed Description

This file contains unit tests for all functions from the time_management.cpp.

Author

Marc Ubbelohde

3.11.2 Function Documentation

3.11.2.1 test_add_time()

```
boolean test_add_time ( )
```

Unit test for the function "add_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks samples of timestamps

See also

add_time

3.11.2.2 test_check_and_update_offset()

```
boolean test_check_and_update_offset ( )
```

Unit test for the function "check_and_update_offset".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and initialize time.json with test data

1. check if the offset is updated correctly

See also

check_and_update_offset

3.11.2.3 test_compare_time()

```
boolean test_compare_time ( )
```

Unit test for the function "compare_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and initialize time.json with test data

1. checks with loop if true is returned when times match
2. checks if false is returned when times do not match

See also

compare_time

3.11.2.4 test_get_current_time()

```
boolean test_get_current_time ( )
```

Unit test for the function "get_current_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and initialize time.json with test data

1. checks twice if time is returned correctly

See also

get_current_time

3.11.2.5 test_init_time()

```
boolean test_init_time ( )
```

Unit test for the function "get_NTP_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks if the document is empty (since Wifi is not available)
2. functionality is tested empirically

See also

get_NTP_time

3.11.2.6 test_turn_seconds_in_time()

```
boolean test_turn_seconds_in_time ( )
```

Unit test for the function "turn_seconds_in_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks samples of conversions

See also

turn_seconds_in_time

3.11.2.7 test_update_time()

```
boolean test_update_time ( )
```

Unit test for the function "update_time".

Returns

boolean - true if the test passed, false if the test failed

- Setup: Clean LittleFS and initialize time.json with test data

1. checks if the time is updated correctly in Station mode
2. checks if the time is updated correctly in AP mode

See also

update_time

3.11.2.8 test_weekday_to_num()

```
boolean test_weekday_to_num ( )
```

Unit test for the function "weekday_to_num".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. checks if the correct number for each weekday is returned
2. checks if "error" is returned if the weekday is not valid

See also

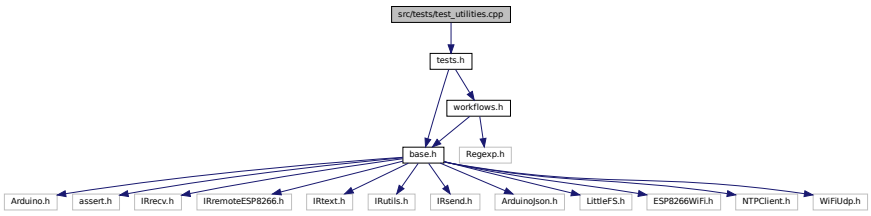
weekday_to_num

3.12 src/tests/test_utilities.cpp File Reference

This file contains functions that are used in multiple tests.

```
#include "tests.h"
```

Include dependency graph for test_utilities.cpp:



Functions

- void clean_LittleFS ()

  Deletes all files in "/", "/signals" and "/programs" in the LittleFS.

3.12.1 Detailed Description

This file contains functions that are used in multiple tests.

Author

Marc Ubbelohde

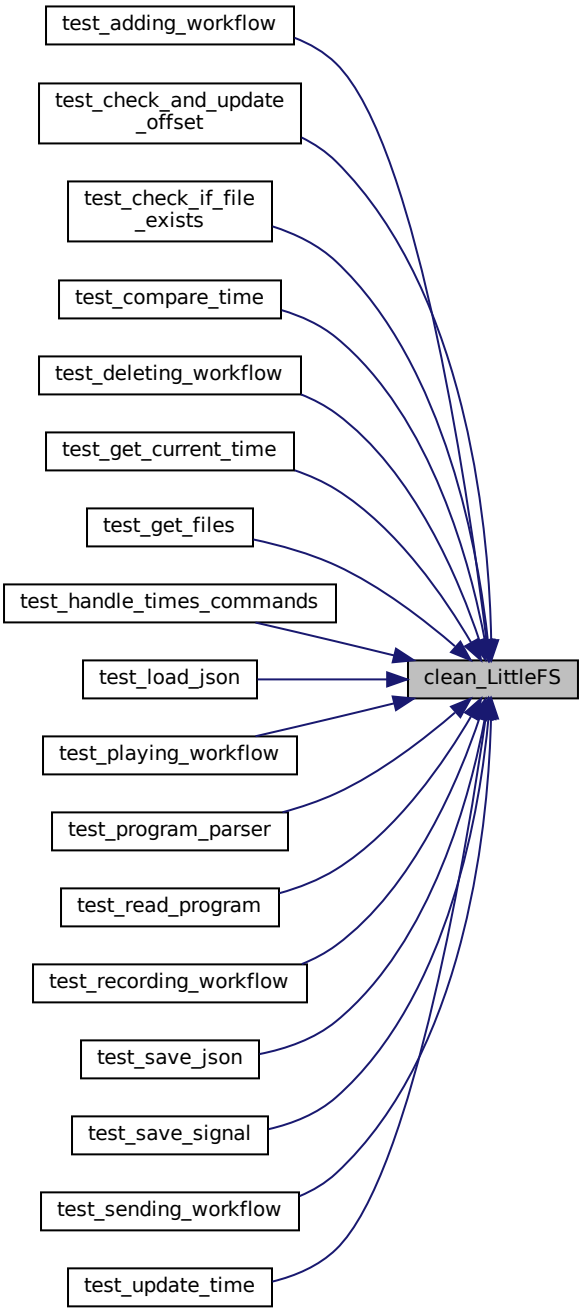
3.12.2 Function Documentation

3.12.2.1 clean_LittleFS()

```
void clean_LittleFS ( )
```

Deletes all files in "/", "/signals" and "/programs" in the LittleFS.

This function does not call any other function. Here is the caller graph for this function:

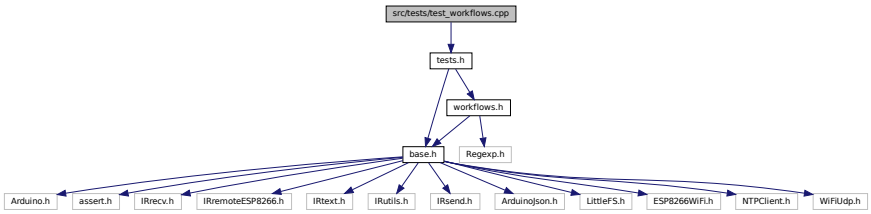


3.13 src/tests/test_workflows.cpp File Reference

This file contains unit tests for all functions from the workflows.cpp.

```
#include "tests.h"
```

Include dependency graph for test_workflows.cpp:



Functions

- boolean test_deleting_workflow ()

  Unit test for the function "deleting_workflow".
- boolean test_recording_workflow ()

  Unit test for the function "recording_workflow".
- boolean test_sending_workflow ()

  Unit test for the function "sending_workflow".
- boolean test_adding_workflow ()

  Unit test for the function "adding_workflow".
- boolean test_playing_workflow ()

  Unit test for the function "playing_workflow".
- boolean test_program_parser ()

  Unit test for the function "program_parser".
- boolean test_handle_wait_command ()

  Unit test for the function "handle_wait_command".
- boolean test_handle_times_commands ()

  Unit test for the function "handle_times_commands".

3.13.1 Detailed Description

This file contains unit tests for all functions from the workflows.cpp.

Author

Marc Ubbelohde

3.13.2 Function Documentation

3.13.2.1 test_adding_workflow()

```
boolean test_adding_workflow ( )
```

Unit test for the function "adding_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS

1. check if functions return value is correct
2. check if program code is correctly written to file

See also

adding_workflow

3.13.2.2 test_deleting_workflow()

```
boolean test_deleting_workflow ( )
```

Unit test for the function "deleting_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create test signal file

1. check if function returns correct error message when file does not exist
2. check if no other file is deleted
3. check if file is deleted correctly
4. check again if no other file is deleted

See also

deleting_workflow

3.13.2.3 test_handle_times_commands()

```
boolean test_handle_times_commands ( )
```

Unit test for the function "handle_times_commands".

Returns

boolean - true if the test passed, false if the test failed

- Setup: create test signal in LittleFS

1. check error message if weekday is invalid
2. check if invalid commands are caught

See also

handle_times_commands

3.13.2.4 test_handle_wait_command()

```
boolean test_handle_wait_command ( )
```

Unit test for the function "handle_wait_command".

Returns

boolean - true if the test passed, false if the test failed

- Setup: -

1. check if return value is "success"
2. check if set amount of time is waited
3. check if function waited too long

See also

handle_wait_command

3.13.2.5 test_playing_workflow()

```
boolean test_playing_workflow ( )
```

Unit test for the function "playing_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create 2 test programs in LittleFS

1. check if functions return value is correct when file does not exist
2. check if program can be correctly executed
3. check if error messages of program_parser are shown correctly

See also

playing_workflow

3.13.2.6 test_program_parser()

```
boolean test_program_parser ( )
```

Unit test for the function "program_parser".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create 2 test signals in LittleFS

1. check if program can be executed correctly
2. if error message is correct when program is faulty
3. if error message is correct when signal does not exist

See also

program_parser

3.13.2.7 test_recording_workflow()

```
boolean test_recording_workflow ( )
```

Unit test for the function "recording_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS

1. check if error message is correct when nothing was recorded
2. check if no file is written when nothing was recorded

See also

recording_workflow

3.13.2.8 test_sending_workflow()

```
boolean test_sending_workflow ( )
```

Unit test for the function "sending_workflow".

Returns

boolean - true if the test passed, false if the test failed

- Setup: clean LittleFS and create 2 test signals in LittleFS

1. check if error message is correct when file does not exist
2. check if sequence can be correctly sent
3. check if error messages of send_signal are shown correctly (1 example)

See also

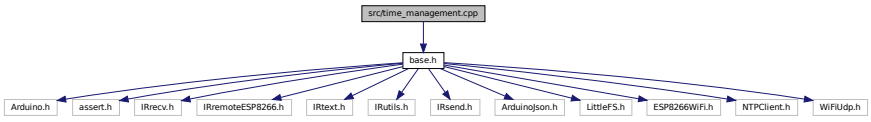
sending_workflow

3.14 src/time_management.cpp File Reference

This file contains the functions to manage the time.

```
#include "base.h"
```

Include dependency graph for time_management.cpp:



Functions

- String weekday_to_num (String weekday)

  Converts a weekday String to a weekday number.
- boolean compare_time (String time, boolean weekday_included)

  Compare specified time with current time.
- void update_time (String time, boolean AP_mode)

  Updates the time in the LittleFS.
- String get_current_time ()

  Loads the current time from LittleFS.
- String turn_seconds_in_time (unsigned long input_seconds)

  This function converts seconds to time format.
- String add_time (String time, String offset_time)

  adds two times together
- void init_time ()

  Gets time from NTP server.
- void check_and_update_offset ()

  Checks if millis() overflowed and updates time if necessary.

3.14.1 Detailed Description

This file contains the functions to manage the time.

Author

Marc Ubbdelohde

The time functions are exlusively used in the timed programs. The complexity of some of the functions is due to the fact that the device does not use an external RTC and that the millis() function overflows after about 49 days. The different functions utilize functions from the filesystem.cpp file to load and save time information to the LittleFS. They provide functionality to each other and the higher level functions in workflows.cpp and main.cpp where fronend functionalities are implemented.

3.14.2 Function Documentation

3.14.2.1 add_time()

```
String add_time (
            String time,
            String offset_time )
```

adds two times together

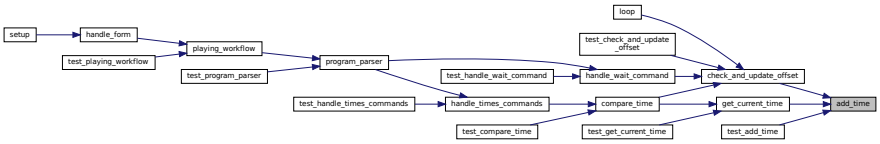
Parameters

time	- time to add to in format "hh:mm:ss weekday"
offset_time	- time to add in format "hh:mm:ss"

Returns

String - time in format "hh:mm:ss weekday"

This function adds two times together. The order of the parameters is important. The first parameter contains the weekday, the second parameter does not.
This function does not call other functions. Here is the caller graph for this function:

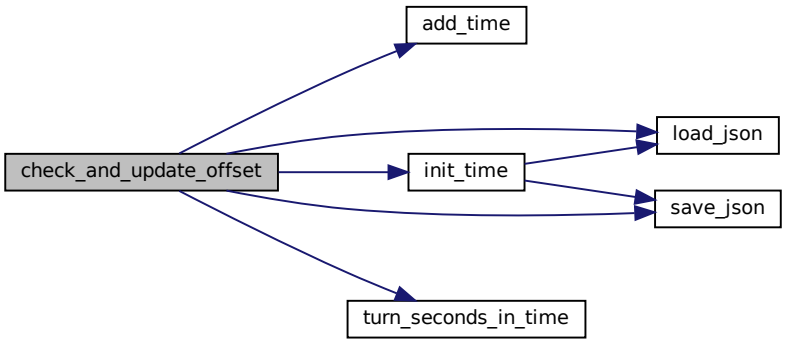


3.14.2.2 check_and_update_offset()

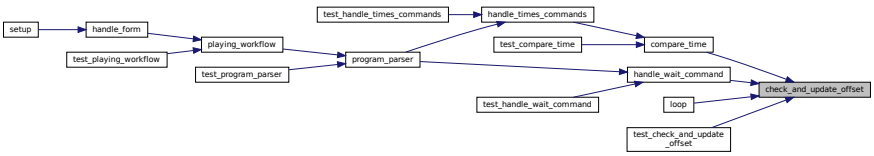
```
void check_and_update_offset ( )
```

Checks if millis() overflowed and updates time if necessary.
Since millis() overflows after 49.7 days, this function checks if an overflow occured and updates the time saved in "time.json" every time it happens to still be able to calculate teh current time. This function is used whenever long waiting times are expected (e.g. in timed programs, wait/skip command or user inactivity). It is important to note that this function has to be able to work offline (no NTP call) since it should be possible to run programs without

internet connection. Here is the call graph for this function:



Here is the caller graph for this function:



3.14.2.3 compare_time()

```
boolean compare_time (
            String time,
            boolean weekday_included )
```

Compare specified time with current time.

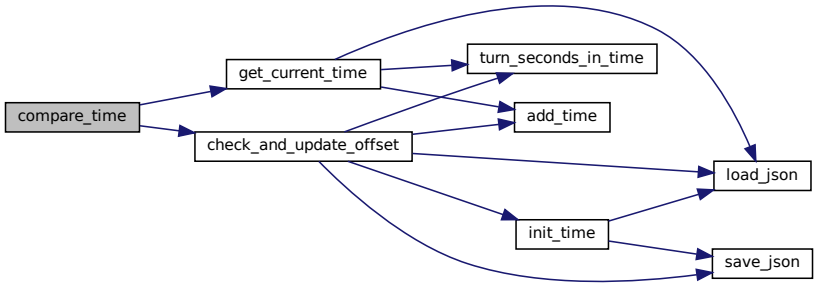
Parameters

time	- String in format "weekday hh:mm:ss timezone"
weekday_included	- true if weekday is included in time, false if not

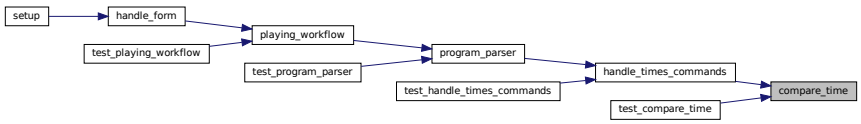
Returns

boolean - true if time is equal to current time, false if not

This elementary function checks if the current time is equal to the time in the program. It is used in timed programs and handles millis() overflow. The function has a delay of 500ms to reduce the number of operations inside the while(true) loop. Here is the call graph for this function:



Here is the caller graph for this function:



3.14.2.4 get_current_time()

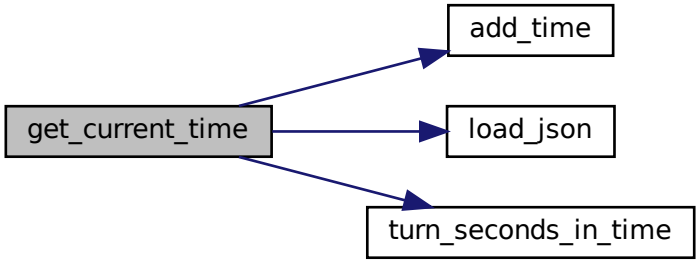
`String get_current_time ( )`

Loads the current time from LittleFS.

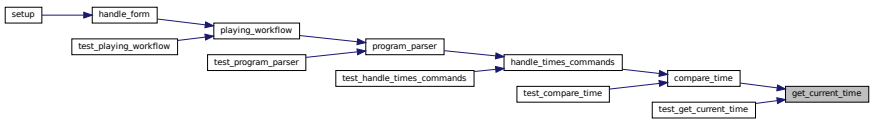
Returns

String - current time in format "hh:mm:ss weekday"

This function loads the time from the LittleFS, adds the relative offset between the offset of initialization and the current offset and returns the current time. Here is the call graph for this function:



Here is the caller graph for this function:

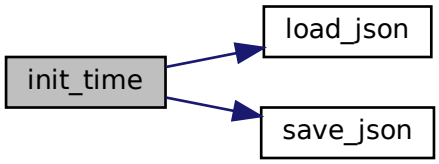


3.14.2.5 init_time()

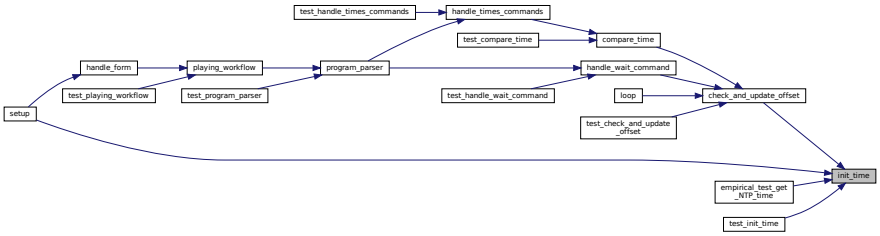
```
void init_time ( )
```

Gets time from NTP server.

This function initiates the time by getting the time from the NTP server. It then saves it ot the LittlfeFS. It also passes the saved timezone to the NTP server or 0 if no timezone is saved. Here is the call graph for this function:



Here is the caller graph for this function:



3.14.2.6 turn_seconds_in_time()

```
String turn_seconds_in_time (
            unsigned long input_seconds )
```

This function converts seconds to time format.

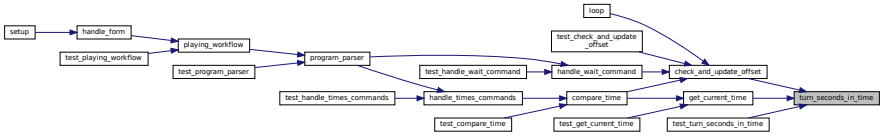
Parameters

input_seconds	- seconds to convert

Returns

String - time in format "hh:mm:ss"

This function converts seconds to time format. It is used in get_current_time() to prepare millis() offset for comparison with saved time.
This function does not call other functions. Here is the caller graph for this function:



3.14.2.7 update_time()

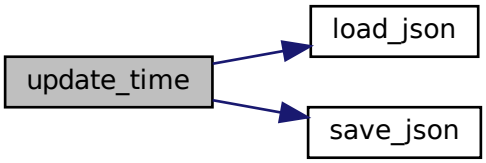
```
void update_time (
            String time,
            boolean AP_mode )
```

Updates the time in the LittleFS.

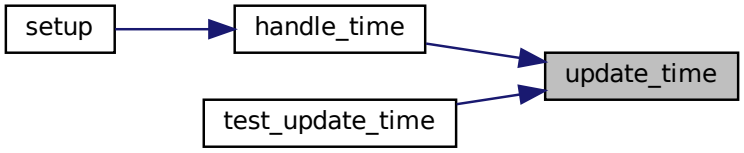
Parameters

time	- String in format "weekday hh:mm:ss timezone"
AP_mode	- true if the device is in AP mode, false if not

This function is called when the user presses the "sync" button on the website. It updates only the timezone to the LittleFS since the time from the NTP request is more precise than the time from the user. Here is the call graph for this function:



Here is the caller graph for this function:



3.14.2.8 weekday_to_num()

```
String weekday_to_num (
            String weekday )
```

Converts a weekday String to a weekday number.

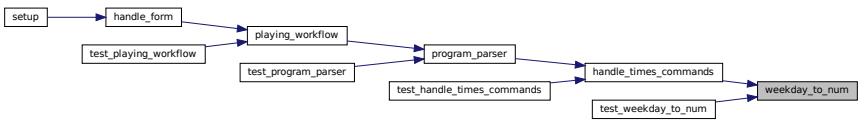
Parameters

weekday	- weekday as a String

Returns

String - weekday as a number:
"Monday" - "1"
"Tuesday" - "2"
"Wednesday" - "3"
"Thursday" - "4"
"Friday" - "5"
"Saturday" - "6"
"Sunday" - "0"
error - "error"

This function converts a weekday String to a weekday number.
This function does not call any function. Here is the caller graph for this function:

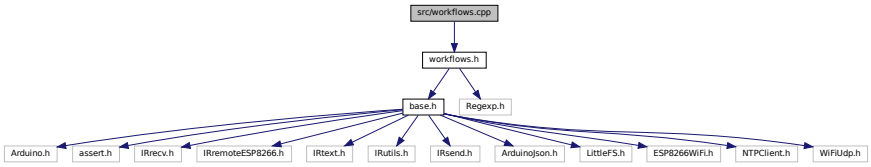


3.15 src/workflows.cpp File Reference

This file contains high level functions.

```
#include "workflows.h"
```

Include dependency graph for workflows.cpp:



Functions

- String deleting_workflow (String directory, String name)

  This function deletes a file from the LittleFS filesystem.
- String recording_workflow (String signal_name)

  This function records and saves a signal.
- String sending_workflow (String signal_name)

  This function loads a signal from a file and sends it.
- String adding_workflow (String program_name, String program_code)

  This function creates a file with the programs name and writes the code to it.
- String playing_workflow (String program_name)

  This function loads a program from a file and hands it to the program_parser.
- String program_parser (String code)

  This function parses the code of a program line by line and executes the commands.
- String handle_wait_command (unsigned long waiting_time)

  This function waits a certain amount of time.
- String handle_times_commands (String command, boolean day_included)

  This function executes timed commands.

3.15.1 Detailed Description

This file contains high level functions.

Author

Marc Ubbelohde

The functions in this file implement workflows which define the main functionalities of the device such as recording signals and sending and deleting signals and programs. Some functions are written here and not in the filesystem.cpp file because they are called directly from the webpage or use other functions from this file.

3.15.2 Function Documentation

3.15.2.1 adding_workflow()

```
String adding_workflow (
            String program_name,
            String program_code )
```

This function creates a file with the programs name and writes the code to it.

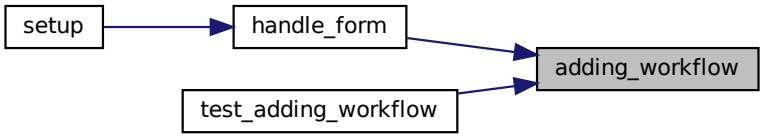
Parameters

program_name	- name of the program to be added
program_code	- code of the program to be added

Returns

String - message that will be displayed on the webpage:
"success message" - if file was created and code was written
"error message" - if file could not be created

Here is the caller graph for this function:



3.15.2.2 deleting_workflow()

```
String deleting_workflow (
            String directory,
            String name )
```

This function deletes a file from the LittleFS filesystem.

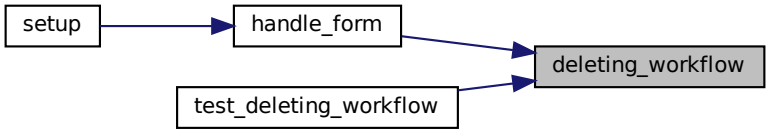
Parameters

directory	- "signals" or "programs"
name	- name of the sequence or program to be deleted

Returns

String - message that will be displayed on the webpage:
"success message" - if file was deleted
"error message" - if file could not be found

This function is used to delete signals and programs.
This function does not call other functions. Here is the caller graph for this function:



3.15.2.3 handle_times_commands()

```
String handle_times_commands (
            String command,
            boolean day_included )
```

This function executes timed commands.

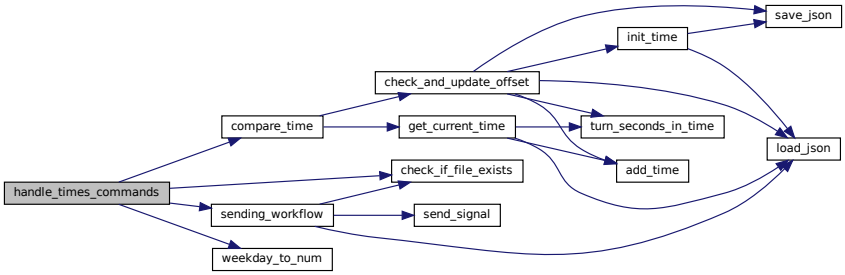
Parameters

command	- String command: "weekday hh:mm:ss signal_name" - if day_included is true "hh:mm:ss signal_name" - if day_included is false
day_included	- true if day is included in command, false if not

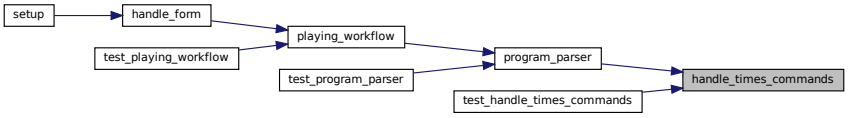
Returns

String - message that will be displayed on the webpage:
"success message" - if command was executed successfully

This function waits until a certain day and/or time is reached and then executes the given signal. Here is the call graph for this function:



Here is the caller graph for this function:



3.15.2.4 handle_wait_command()

```
String handle_wait_command (
            unsigned long waiting_time )
```

This function waits a certain amount of time.

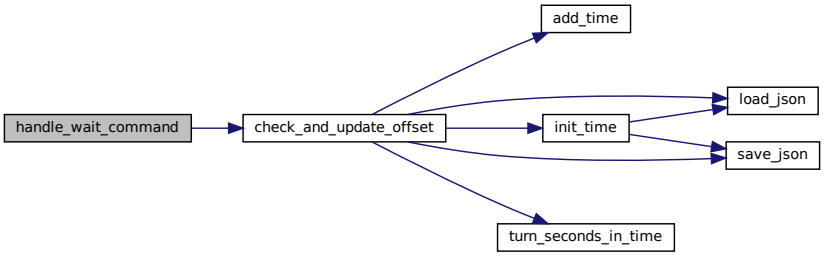
Parameters

waiting_time	- time to wait in milliseconds

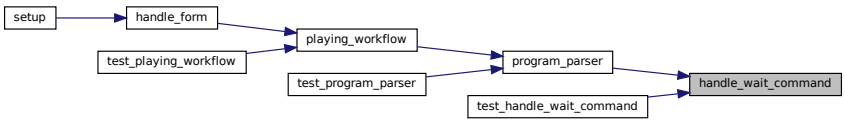
Returns

String - message that will be displayed on the webpage:
"success message" - if command was executed successfully
"error message" - if command was interrupted by the user

This function waits a certain amount of time. It is used for the wait and skip command. It is necessary to check beforehand if a millis() overflow will occur during the waiting time. If an overflow will occur, the function will first calculate the time it will have to wait after the overflow occurs, then it will wait until the overflow occurs and waits the remaining time. The function also checks if the user pressed the interrupt button. Here is the call graph for this function:



Here is the caller graph for this function:



3.15.2.5 playing_workflow()

```
String playing_workflow (
            String program_name )
```

This function loads a program from a file and hands it to the program_parser.

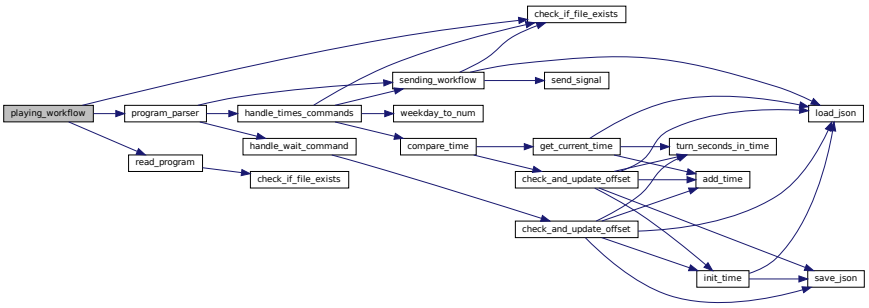
Parameters

program_name	- name of the program to be played

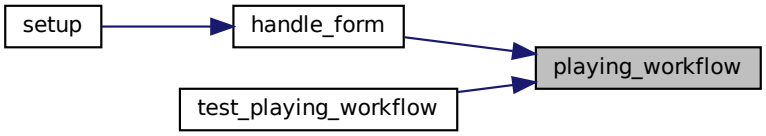
Returns

String - message that will be displayed on the webpage:
"success message" - if file was found and program was played successfully
"error message" - if file could not be found or if in one of the commands an error occured (error message gets passed by program_parser)

This function loads a program from a file and hands it to the program_parser. The program_parser then sends the commands and returns a message when the execution of the program finished. Here is the call graph for this function:



Here is the caller graph for this function:



3.15.2.6 program_parser()

```
String program_parser (
            String code )
```

This function parses the code of a program line by line and executes the commands.

Parameters

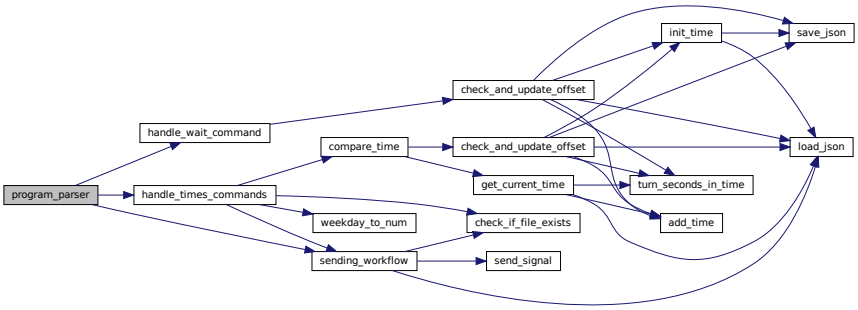
code	- code of the program to be parsed

Returns

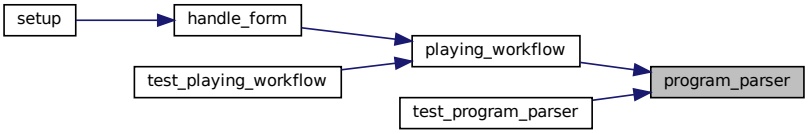
String - message that will be displayed on the webpage:
"success message" - if program was played successfully
"error message" - if in one of the commands an error occured an command specific error message is returned

This function parses the code of a program line by line and executes the commands. It was necessary to split the parser from the playing_workflow function to be able to call it recursively (for loops). Each line is searched for command specific keywords and the corresponding command handler is called. Here is the call graph for this

function:



Here is the caller graph for this function:



3.15.2.7 recording_workflow()

```
String recording_workflow (
            String signal_name )
```

This function records and saves a signal.

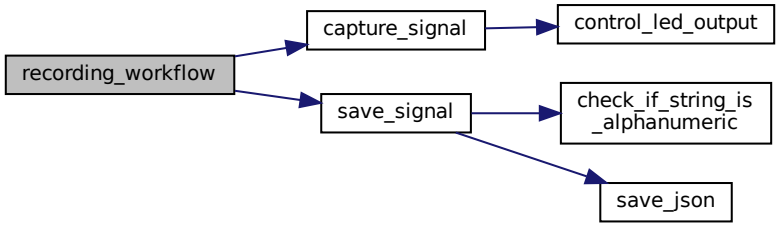
Parameters

signal_name	- name of the sequence to be recorded

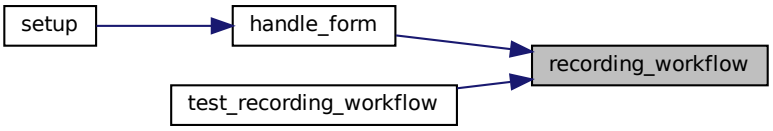
Returns

String - message that will be displayed on the webpage:
"success message" - if signal was saved
"error message" - no signal was captured

This function records a signal and saves it to a file in the LitteFS with the signals name. (spaces at the end of the signal name will be removed) Here is the call graph for this function:



Here is the caller graph for this function:



3.15.2.8 sending_workflow()

```
String sending_workflow (
            String signal_name )
```

This function loads a signal from a file and sends it.

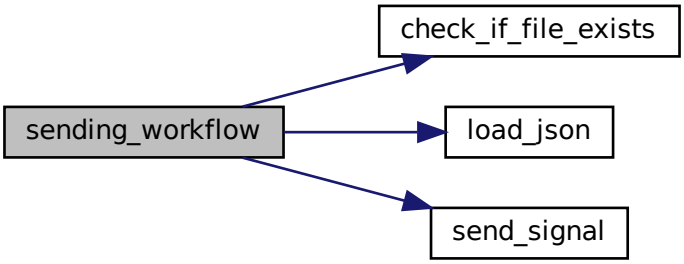
Parameters

signal_name	- name of the sequence to be sent

Returns

String - message that will be displayed on the webpage:
"success message" - if file was found and command was sent
"error message" - if file could not be found

Here is the call graph for this function:



Here is the caller graph for this function: