

# **Controle Wifi**

Manual do programador

João Vianna (jvianna@gmail.com)

**Versão 0.82**

**03/05/2024**

## Referência do Arquivo main.c

Controle de Periféricos através do Wifi.

### Funções

void [app\\_main](#) (void)

*Função principal do aplicativo.*

---

### Descrição detalhada

Este aplicativo cria um protocolo para controle de um módulo esp32 utilizando uma conexão Wifi e um servidor HTTP.

#### Autor

Joao Vianna ([jvianna@gmail.com](mailto:jvianna@gmail.com))

#### Versão

0.82

Base do código - Exemplos da biblioteca esp-idf

#### Veja também

[app\\_web\\_server.c](#) Protocolo HTTP para controlar os periféricos.

[controle\\_gpio.c](#) Interface com o micro-controlador.

Informações sobre mDNS:

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mdns.html>

<https://docs.espressif.com/projects/esp-protocols/mdns/docs/latest/en/index.html>

#### Atividades futuras:

Telas HTML embutidas na memória flash, para testar e configurar.

Melhorar documentação de instalação, e desenvolvimento do servidor e cliente.

---

### Definições e Macros

```
#define USAR_MDNS 1
```

---

## Funções (detalhes)

### **void app\_main (void )**

Ativa os diversos módulos para: Configurar o micro-controlador; Ler a configuração de memória permanente; Iniciar a comunicação via Wifi no modo configurado; Iniciar o servidor HTTP.

## Referência do Arquivo `app_config.c`

Configuração do aplicativo em memória permanente.

### Funções

enum modo\_conexao [app\\_config\\_modulo\\_wifi](#) (void)  
*Indica se modo Wifi deve ser Access Point (AP) ou Station (STA)*

const char \* [app\\_config\\_wifi\\_ssid](#) (void)  
*Obtém ssid da conexão Wifi.*

const char \* [app\\_config\\_wifi\\_password](#) (void)  
*Obtém senha da conexão Wifi.*

const char \* [app\\_config\\_hostname](#) (void)  
*Obtém nome do servidor para mDNS.*

void [app\\_config\\_set\\_modulo\\_wifi](#) (const char \* modo)  
*Altera modo da conexão Wifi.*

void [app\\_config\\_set\\_wifi\\_ssid](#) (const char \*ssid)  
*Altera ssid da conexão Wifi.*

void [app\\_config\\_set\\_wifi\\_password](#) (const char \*pwd)  
*Altera a senha da conexão Wifi.*

void [app\\_config\\_set\\_hostname](#) (const char \*nome)  
*Altera o nome do servidor HTTP (mDNS).*

void [app\\_config\\_ler](#) (void)  
*Ler configuração do aplicativo da memória FLASH.*

esp\_err\_t [app\\_config\\_gravar](#) (void)  
*Gravar configuração do aplicativo na memória FLASH.*

---

## Descrição detalhada

Camada de abstração que esconde dos demais módulos a maneira como a configuração é armazenada.

Recupera e armazena informações de configuração gravadas na memória FLASH utilizando o sistema de arquivos LittleFS.

NOTA: Existe um pino no micro-controlador que força o retorno à configuração original. Se este pino estiver ativado (ON), a configuração armazenada é ignorada, e os valores 'de fábrica' são utilizados.

### Veja também

<https://github.com/littlefs-project/littlefs>

### Autor

Joao Vianna ([jvianna@gmail.com](mailto:jvianna@gmail.com))

### Versão

0.82 03/05/2024

Pino do micro-controlador força retorno às configurações de fábrica.

Adicionado campo hostname e pequenas mudanças nos nomes dos parâmetros.

Código de armazenamento derivado de:

.../esp-idf/examples/storage/littlefs/main/esp\_littlefs\_example.c

---

## Funções

### **esp\_err\_t app\_config\_gravar (void )**

Nota: Se a configuração não foi alterada, nada será gravado.

### **void app\_config\_ler (void )**

NOTA: Se o pino de voltar à configuração original estiver ligado (ON), ignora a configuração gravada e carrega um padrão 'de fábrica'.

### **void app\_config\_set\_hostname (const char \* *nome*)**

#### **Parâmetros**

<i>nome</i>	Nome do servidor
-------------	------------------

#### **Veja também**

[\*\*MAX\\_HOSTNAME\\_LEN\*\*](#)

### **void app\_config\_set\_modulo\_wifi (const char \* *modulo*)**

#### **Parâmetros**

<i>modulo</i>	"STA" para modo Station, ou "AP" para modo Access Point
---------------	---

**void app\_config\_set\_wifi\_password (const char \* *pwd*)**

**Parâmetros**

<i>pwd</i>	senha a utilizar
------------	------------------

**Veja também**

[MAX\\_PASSWORD\\_LEN](#)

NOTA: Se o tamanho do parâmetro exceder o limite, nada será alterado.

**void app\_config\_set\_wifi\_ssid (const char \* *ssid*)**

**Parâmetros**

<i>ssid</i>	ssid a utilizar
-------------	-----------------

**Veja também**

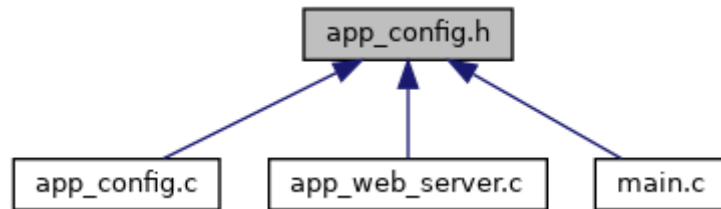
[MAX\\_SSID\\_LEN](#)

NOTA: Se o tamanho do parâmetro exceder o limite, nada será alterado.

## Referência do Arquivo `app_config.h`

Configuração do aplicativo

Arquivos direta ou indiretamente relacionados com esse arquivo:



### Definições e Macros

#define `MAX_SSID_LEN` 32  
*Tamanho máximo do ssid.*

#define `MAX_PASSWORD_LEN` 63  
*Tamanho máximo da senha.*

### Enumerações

enum `modo_conexao_wifi` { `MODO_WIFI_STA`, `MODO_WIFI_AP` }  
*Modo da conexão Wifi (STAtion ou Access Point).*

## Funções

void [app\\_config\\_ler](#) (void)

*Ler configuração do aplicativo da memória FLASH.*

esp\_err\_t [app\\_config\\_gravar](#) (void)

*Gravar configuração do aplicativo na memória FLASH.*

enum modo\_conexao [wifi\\_app\\_config\\_modulo\\_wifi](#) (void)

*Indica se modo Wifi deve ser Access Point (AP) ou Station (STA)*

const char \* [app\\_config\\_wifi\\_ssid](#) (void)

*Obtém ssid da conexão Wifi.*

const char \* [app\\_config\\_wifi\\_password](#) (void)

*Obtém senha da conexão Wifi.*

const char \* [app\\_config\\_hostname](#) (void)

*Obtém nome do servidor para mDNS.*

void [app\\_config\\_set\\_wifi\\_ssid](#) (const char \*ssid)

*Altera ssid da conexão Wifi.*

void [app\\_config\\_set\\_wifi\\_password](#) (const char \*pwd)

*Altera senha conexão Wifi.*

void [app\\_config\\_set\\_hostname](#) (const char \*nome)

*Altera o nome do servidor HTTP (mDNS).*



---

## Funções (detalhes)

### **esp\_err\_t app\_config\_gravar (void )**

Se a configuração não foi alterada, nada será gravado.

### **void app\_config\_ler (void )**

NOTA: Se o pino de voltar à configuração original estiver ligado (ON), ignora a configuração gravada e carrega um padrão 'de fábrica'.

### **void app\_config\_set\_hostname (const char \* *nome*)**

#### **Parâmetros**

<i>nome</i>	Nome do servidor
-------------	------------------

#### **Veja também**

[MAX\\_HOSTNAME\\_LEN](#)

### **void app\_config\_set\_modulo\_wifi (const char \* *modo*)**

#### **Parâmetros**

<i>modo</i>	"STA" para modo Station, ou "AP" para modo Access Point
-------------	---

### **void app\_config\_set\_wifi\_password (const char \* *pwd*)**

#### **Parâmetros**

<i>pwd</i>	senha a utilizar
------------	------------------

#### **Veja também**

[MAX\\_PASSWORD\\_LEN](#)

NOTA: Se o tamanho do parâmetro exceder o limite, nada será alterado.

### **void app\_config\_set\_wifi\_ssid (const char \* *ssid*)**

#### **Parâmetros**

<i>ssid</i>	ssid a utilizar
-------------	-----------------

#### **Veja também**

[MAX\\_SSID\\_LEN](#)

NOTA: Se o tamanho do parâmetro exceder o limite, nada será alterado.

## Referência do Arquivo `app_web_server.c`

Servidor HTTP

### Funções

`httpd_handle_t` [start\\_webserver](#) (void)

*Iniciar servidor http.*

`esp_err_t` [stop\\_webserver](#) (`httpd_handle_t` server)

*Terminar servidor http.*

---

### Descrição detalhada

Um aplicativo cliente controla um módulo contendo atuadores, sensores, etc, utilizando um protocolo HTTP com o servidor.

As solicitações seguem o protocolo:

GET /status

Para obter um texto indicando o status do controlador.

GET /sensor?id=(identificador)

Para ler o estado de um sensor (porta de entrada do módulo), Onde id indica o sensor para o qual se deseja obter o valor.

POST /atuador(id)

action=("off", "on", "toggle" ou "pulse")

[duration=(tempo em ms)]

Para alterar o estado de um atuador (porta de saída do módulo), Onde off desliga, on liga, toggle alterna o estado e pulse liga e desliga após duração de tempo determinada.

POST /config

ssid=(ssid do Wifi)

password=(senha do Wifi)

wifi\_mode=(STA/AP)

hostname=(nome do servidor HTTP)

Para alterar a configuração do módulo.

### Veja também

[app\\_config.h](#)

Base do código - Exemplos da biblioteca esp-idf

Derivado de Simple HTTPD Server Example e Example: GPIO

---

## Funções (detalhes)

**httpd\_handle\_t start\_webserver (void )**

Devolve o handle para o servidor, ou NULL, se houve erro.

**esp\_err\_t stop\_webserver (httpd\_handle\_t server)**

### Parâmetros

<i>server</i>	Handle para o servidor
---------------	------------------------

## Variáveis

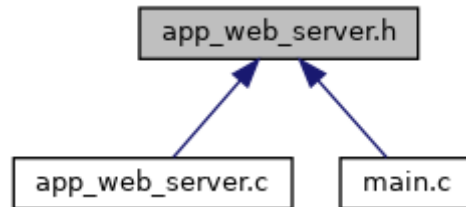
**const char\* acoes\_conhecidas[]**

Valor inicial:= {  
    "off",  
    "on",  
    "toggle",  
    "pulse",  
    "no action"  
}

## Referência do Arquivo `app_web_server.h`

Servidor HTTP

Arquivos direta ou indiretamente relacionados com esse arquivo:



### Funções

`httpd_handle_t` [start\\_webserver](#) (void)

*Iniciar servidor http.*

`esp_err_t` [stop\\_webserver](#) (`httpd_handle_t` server)

*Terminar servidor http.*

---

---

## Funções (detalhes)

**httpd\_handle\_t start\_webserver (void )**

Devolve o handle para o servidor, ou NULL, se houve erro.

**esp\_err\_t stop\_webserver (httpd\_handle\_t *server*)**

### Parâmetros

<i>server</i>	Handle para o servidor
---------------	------------------------

## Referência do Arquivo controle\_gpio.c

Controle dos atuadores e sensores.

### Funções

void [controle\\_gpio\\_iniciar](#) (void)

*Preparar a placa controladora para operar com o aplicativo.*

const char \* [controle\\_gpio\\_status](#) (void)

*Obter status da placa controladora.*

int [controle\\_gpio\\_ler\\_sensor](#) (int id)

*Ler valor de um sensor.*

void [controle\\_gpio\\_mudar\\_atuador](#) (int id, int valor)

*Mudar o valor de um atuador.*

void [controle\\_gpio\\_alternar\\_atuador](#) (int id)

*Alternar o valor de um atuador entre 0 e 1 (toggle)*

void [controle\\_gpio\\_pulsar\\_atuador](#) (int id, int duracao)

*Cria um pulso positivo no valor de um atuador com certa duração.*

bool [controle\\_gpio\\_reconfig](#) (void)

*Indica se o sensor de reconfiguração está ativado.*

---

## Descrição detalhada

Camada de abstração que esconde os detalhes das ligações das portas do micro-controlador.

Para o aplicativo, existem apenas atuadores, sensores, contadores... O código é específico para o ESP32, mas pode ser re-escrito para outros módulos, mantendo a mesma interface.

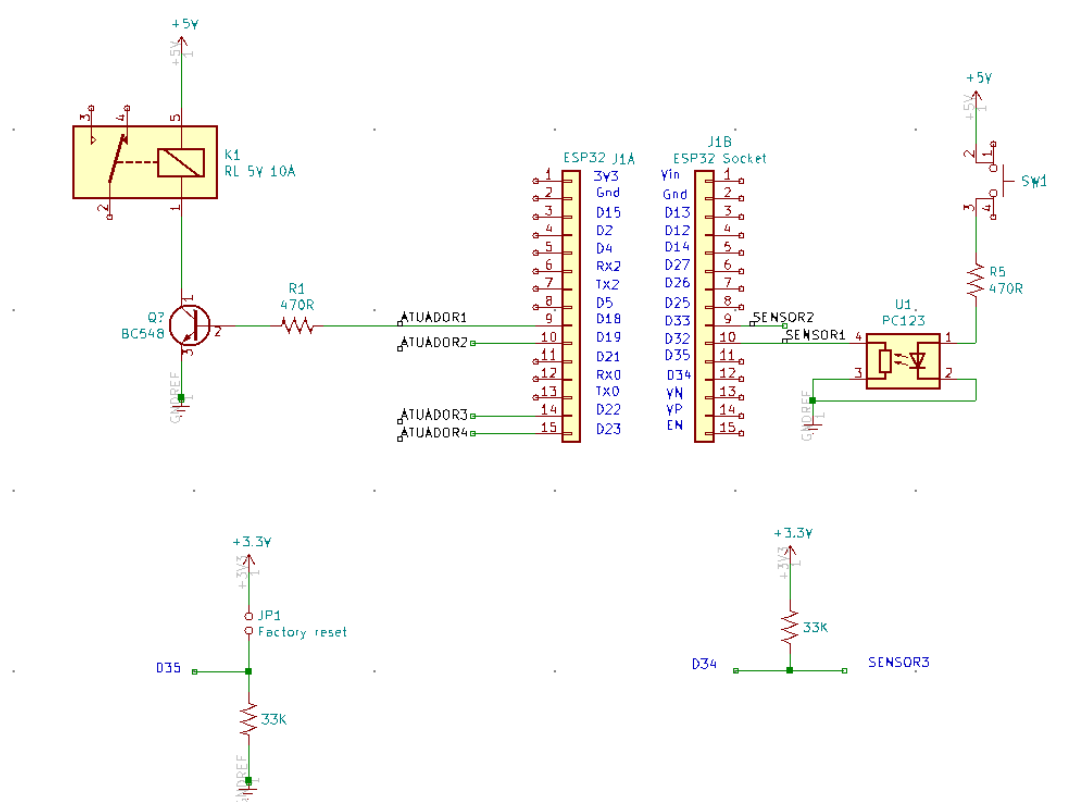
Na abstração, o identificador de cada pino vai de 1 até o limite da classe, independente do pino GPIO a que está conectado. Assim, temos atuador 1, atuador 2, ...; sensor 1, sensor 2, ...

Um pino especial oferece ao aplicativo um indicador de reconfiguração, para que o sistema possa ser retornado à configuração de fábrica.

Código criado a partir do exemplo:

`.../esp-idf/examples/peripherals/gpio/generic_gpio/main/gpio_example_main.c`

A implementação corrente é compatível com o circuito no diagrama abaixo:



Nota: É importante utilizar um diodo polarizado reversamente em paralelo com os relés (não representado no diagrama).

## Funções (detalhes)

### **void controle\_gpio\_iniciar (void )**

Deve ser ativada no início da lógica do aplicativo, antes da lógica que age sobre os diversos periféricos.

### **int controle\_gpio\_ler\_sensor (int *id*)**

#### **Parâmetros**

<i>id</i>	Identificador do sensor (de 1 a MAX_SENSORES)
-----------	---

Retorna 1 se o sensor está em nível ligado e 0 se está desligado.

### **void controle\_gpio\_alternar\_atuador (int *id*)**

#### **Parâmetros**

<i>id</i>	Identificador do atuador (de 1 a MAX_ATUADORES)
-----------	---

### **void controle\_gpio\_mudar\_atuador (int *id*, int *valor*)**

#### **Parâmetros**

<i>id</i>	Identificador do atuador (de 1 a MAX_ATUADORES)
<i>valor</i>	0 para desligado; 1 para ligado

### **void controle\_gpio\_pulsar\_atuador (int *id*, int *duracao*)**

O atuador é ligado por um tempo determinado, e depois desligado.

#### **Parâmetros**

<i>id</i>	Identificador do atuador (de 1 a MAX_ATUADORES)
<i>duracao</i>	Duração do pulso em milissegundos

#### **Atividades futuras:**

Completar implementação utilizando um temporizador.

### **const char\* controle\_gpio\_status (void )**

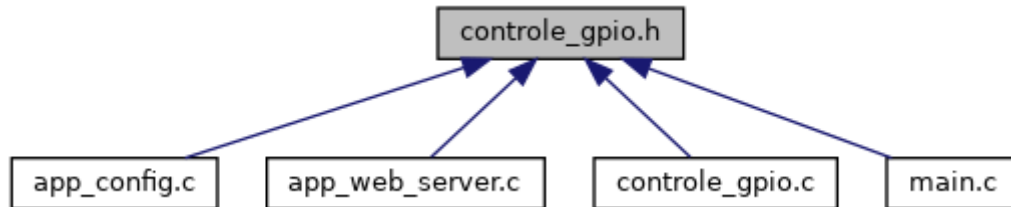
Devolve texto indicando o estado do módulo (número de dispositivos, etc.)



## Referência do Arquivo controle\_gpio.h

Controle dos atuadores e sensores.

Arquivos direta ou indiretamente relacionados com esse arquivo:



### Definições e Macros

#define [MAX\\_ALARMES](#) 1  
*Máximo de alarmes no módulo.*

#define [MAX\\_ATUADORES](#) 4  
*Máximo de atuadores no módulo.*

#define [MAX\\_SENSORES](#) 2  
*Máximo de sensores no módulo.*

### Enumerações

enum [periferico](#) { [PRF\\_NDEF](#), [PRF\\_ATUADOR](#), [PRF\\_ALARME](#),  
[PRF\\_SENSOR](#) }  
*Tipos de periféricos controlados.*

## Funções

void [controle\\_gpio\\_iniciar](#) (void)

*Preparar a placa controladora para operar com o aplicativo.*

const char \* [controle\\_gpio\\_status](#) (void)

*Obter status da placa controladora.*

int [controle\\_gpio\\_ler\\_sensor](#) (int id)

*Ler valor de um sensor.*

void [controle\\_gpio\\_mudar\\_atuador](#) (int id, int valor)

*Mudar o valor de um atuador.*

void [controle\\_gpio\\_alternar\\_atuador](#) (int id)

*Alternar o valor de um atuador entre 0 e 1 (toggle)*

void [controle\\_gpio\\_pulsar\\_atuador](#) (int id, int duracao)

*Cria um pulso positivo no valor de um atuador com certa duração.*

bool [controle\\_gpio\\_reconfig](#) (void)

*Indica se o sensor de reconfiguração está ativado.*

---

---

## Enumerações (detalhes)

enum [periferico](#)

### Enumeradores:

PRF_NDEF	Periférico não definido.
PRF_ATUADOR	Atuador binário (0 - Off, 1 - On)
PRF_ALARME	Alarme (contador de eventos)
PRF_SENSOR	Sensor binário (0 - Off, 1 - On)

---

## Funções (detalhes)

### void controle\_gpio\_iniciar (void )

Deve ser ativada no início da lógica do aplicativo, antes da lógica que age sobre os diversos periféricos.

### int controle\_gpio\_ler\_sensor (int *id*)

#### Parâmetros

<i>id</i>	Identificador do sensor (de 1 a MAX_SENSORES)
-----------	---

Retorna 1 se o sensor está em nível ligado e 0 se está desligado.

### void controle\_gpio\_alternar\_atuador (int *id*)

#### Parâmetros

<i>id</i>	Identificador do atuador (de 1 a MAX_ATUADORES)
-----------	---

### void controle\_gpio\_mudar\_atuador (int *id*, int *valor*)

#### Parâmetros

<i>id</i>	Identificador do atuador (de 1 a MAX_ATUADORES)
<i>valor</i>	0 para desligado; 1 para ligado

**void controle\_gpio\_pulsar\_atuador (int *id*, int *duracao*)**

O atuador é ligado por um tempo determinado, e depois desligado.

**Parâmetros**

<i>id</i>	Identificador do atuador (de 1 a MAX_ATUADORES)
<i>duracao</i>	Duração do pulso em milissegundos

**Atividades futuras:**

Completar implementação utilizando um temporizador.

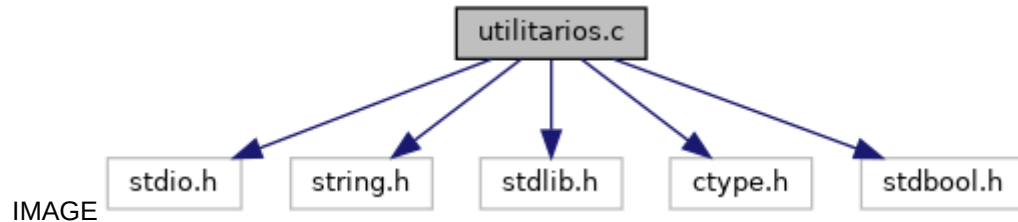
**const char\* controle\_gpio\_status (void )**

Devolve texto indicando o estado do módulo (número de dispositivos, etc.)

## Referência do Arquivo utilitarios.c

Diversos utilitários para lidar com strings, etc.

Dependência de inclusões para utilitarios.c:



## Funções

bool [str\\_startswith](#) (const char \*str, const char \*substr)

*Função auxiliar que verifica se um string se inicia com determinada sequência.*

void [urldecode2](#) (char \*dst, const char \*src)

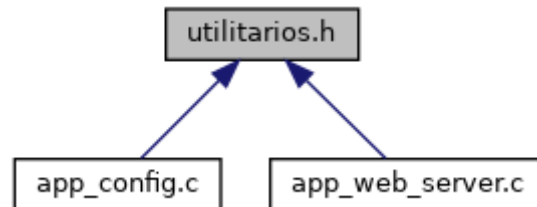
*Função auxiliar para decodificar uma URI.*

---

## Referência do Arquivo utilitarios.h

Diversos utilitários para lidar com strings, etc.

Arquivos direta ou indiretamente relacionados com esse arquivo:



### Funções

bool [str\\_startswith](#) (const char \*str, const char \*substr)

*Função auxiliar - Verifica se um string se inicia com determinada sequência.*

void [urldecode2](#) (char \*dst, const char \*src)

*Função auxiliar para decodificar uma URI.*

---

### Funções (detalhes)

bool **str\_startswith** (const char \* **str**, const char \* **substr**)

#### Parâmetros

<i>str</i>	string que vai ser verificada.
<i>substr</i>	substring que deveria aparecer no início de str.

#### Retorna

true se os strings são iguais até o fim do segundo.

void **urldecode2** (char \* **dst**, const char \* **src**)

#### Parâmetros

<i>dst</i>	Destino do resultado (pode ser o mesmo que a origem).
<i>src</i>	Começa com a URL original, a ser transformada;

Código obtido na Internet

#### Veja também

<https://gist.github.com/jmsaavedra/7964251>

## Referência do Arquivo wifi\_softap.c

Inicia Wifi no modo soft Access Point.

### Funções

void [wifi\\_init\\_softap](#) (const char \*ssid)

*Iniciar serviço de Wifi no modo Soft AP (Access Point)*

---

### Descrição detalhada

Código original em:

.../esp-idf/examples/wifi/getting\_started/softAP/

NOTA: Houve modificações do código original do exemplo conforme outros exemplos encontrados na Internet para permitir o uso juntamente com um servidor HTTP.

#### [Atividades futuras:](#)

Usar mDNS para publicar nome do servidor. Até esta versão, em modo AP, o IP do servidor é fixo (192.168.0.10).

---

### Funções (detalhes)

void wifi\_init\_softap (const char \* *ssid*)

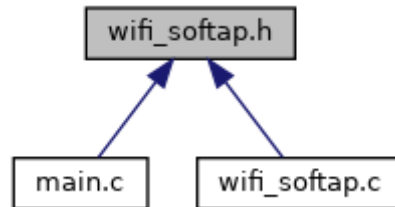
#### Parâmetros

<i>ssid</i>	Identificador do Access Point
-------------	-------------------------------

## Referência do Arquivo `wifi_softap.h`

Inicia Wifi no modo soft Access Point.

Arquivos direta ou indiretamente relacionados com esse arquivo:



## Funções

void [wifi\\_init\\_softap](#) (const char \*ssid)

*Iniciar serviço de Wifi no modo Soft AP (Access Point)*

---

## Funções (detalhes)

void `wifi_init_softap` (const char \* *ssid*)

### Parâmetros

<i>ssid</i>	Identificador do Access Point
-------------	-------------------------------



## Referência do Arquivo wifi\_station.c

Inicia Wifi no modo Station.

### Funções

void [wifi\\_init\\_sta](#) (const char \*ssid, const char \*pwd)  
*Iniciar serviço de Wifi no modo Station.*

---

### Descrição detalhada

Código original em:

.../esp-idf/examples/wifi/getting\_started/station/main/station\_example\_main.c

NOTA: Foram realizadas modificações mínimas, passando o ssid e senha como parâmetros para a função de iniciar.

---

### Funções (detalhes)

void wifi\_init\_sta (const char \* *ssid*, const char \* *pwd*)

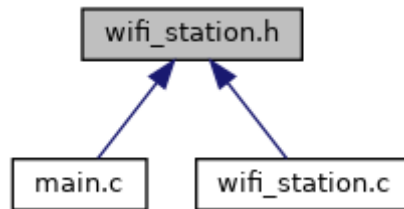
#### Parâmetros

<i>ssid</i>	Identificador do serviço Wifi com que conectar.
<i>pwd</i>	Senha a utilizar para se conectar.

## Referência do Arquivo wifi\_station.h

Inicia Wifi no modo Station.

Arquivos direta ou indiretamente relacionados com esse arquivo:



## Funções

void [wifi\\_init\\_sta](#) (const char \*ssid, const char \*pwd)

*Iniciar serviço de Wifi no modo Station.*

---

## Funções (detalhes)

void wifi\_init\_sta (const char \* *ssid*, const char \* *pwd*)

### Parâmetros

<i>ssid</i>	Identificador do serviço Wifi com que conectar.
<i>pwd</i>	Senha a utilizar para se conectar.

## Lista de atividades futuras

### Global [controle\\_gpio\\_ler\\_sensor](#) (int id)

Acrescentar sensores do tipo ADC, com valores inteiros.

### Global [controle\\_gpio\\_pulsar\\_atuador](#) (int id, int duracao)

Completar implementação utilizando um temporizador.

### Arquivo [main.c](#)

Telas HTML embutidas na memória flash, para testar e configurar.

Melhorar documentação de instalação, e desenvolvimento do servidor e cliente.

### Arquivo [wifi\\_softap.c](#)

Usar mDNS para publicar nome do servidor. Até esta versão, em modo AP, o IP do servidor é fixo (192.168.0.10).

## Índice de Arquivos Fonte

main.c.....	2
app_config.c.....	4
app_config.h.....	7
app_web_server.c.....	10
app_web_server.h.....	12
controle_gpio.c.....	14
controle_gpio.h.....	17
utilitarios.c.....	21
utilitarios.h.....	22
wifi_softap.c.....	23
wifi_softap.h.....	24
wifi_station.c.....	25
wifi_station.h.....	26
Lista de atividades futuras.....	27