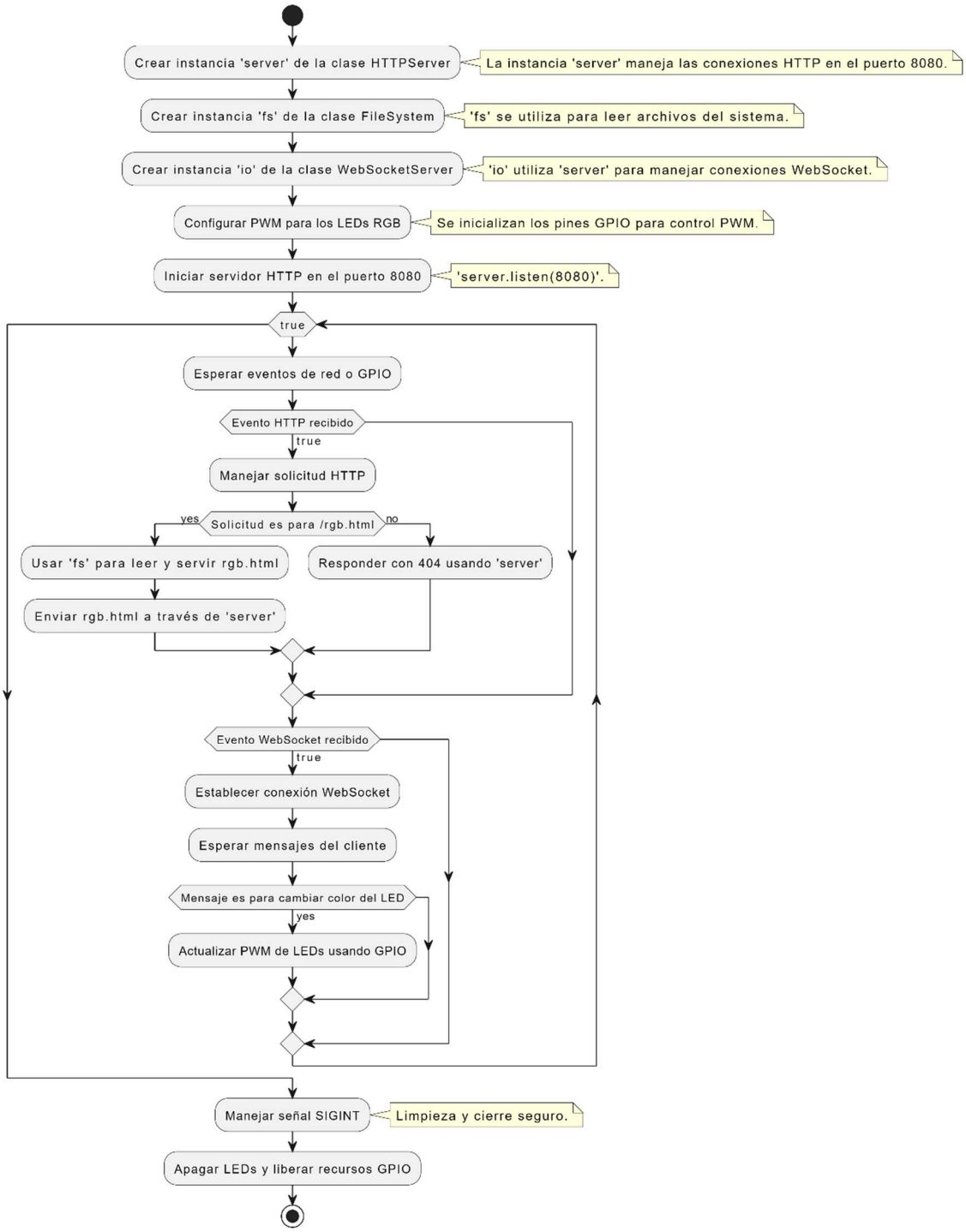
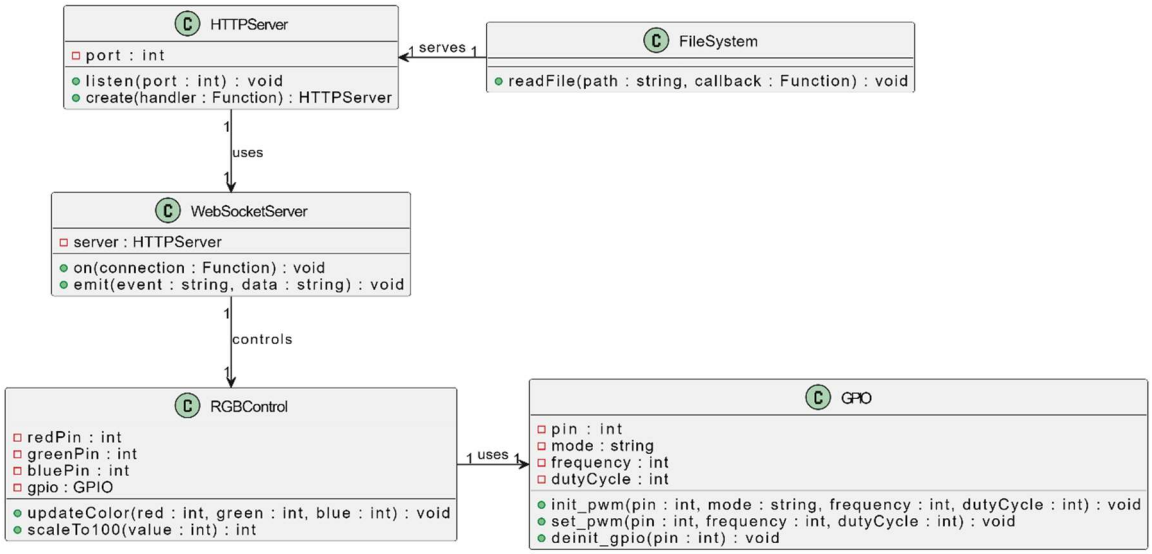
Rgbsocketio

serverAnode

```
// Crea un servidor HTTP que escucha en el puerto 8080. 'http' es una instancia de la clase HTTPServer.
// Esta instancia gestiona todas las solicitudes HTTP dirigidas al servidor en el puerto 8080.
const http = require('http').createServer(handler);

// 'fs' es una instancia de la clase FileSystem, usada para operaciones de lectura de archivos.
// Usado para acceder y leer archivos desde el disco, específicamente para servir páginas web a los clientes.
var fs = require('fs');

// 'io' es una instancia de la clase WebSocketServer, configurada para manejar comunicación en tiempo real sobre el servidor HTTP.
// Permite la comunicación en tiempo real con los clientes conectados, usada para recibir y enviar comandos de control del LED RGB.
var io = require('socket.io')(http);

// 'gpio' es una instancia de la clase GPIO, usada para controlar el hardware de los pines GPIO mediante PWM.
// Inicializa los pines GPIO designados para controlar un LED RGB mediante señales PWM.
const gpio = require("@iiot2k/gpiox");

// Configura los pines GPIO para cada color del LED RGB.
// Inicializa los pines como salidas PWM y configura la frecuencia y el ciclo de trabajo inicial.
const ledRed = 22; // Atributo del objeto GPIO para el pin del LED rojo
const ledGreen = 17; // Atributo del objeto GPIO para el pin del LED verde
const ledBlue = 27; // Atributo del objeto GPIO para el pin del LED azul
gpio.init_pwm(ledRed, gpio.GPIO_MODE_PWM, PWM_FREQ, PWM_DUTY_INIT);
gpio.init_pwm(ledGreen, gpio.GPIO_MODE_PWM, PWM_FREQ, PWM_DUTY_INIT);
gpio.init_pwm(ledBlue, gpio.GPIO_MODE_PWM, PWM_FREQ, PWM_DUTY_INIT);

// Activa el servidor para aceptar conexiones HTTP en el puerto especificado.
// Comienza a escuchar las solicitudes entrantes en el puerto 8080.
http.listen(8080);

// Función para manejar las solicitudes HTTP.
// Lee el archivo 'rgb.html' y lo envía al cliente o devuelve un error 404 si el archivo no se encuentra.
function handler(req, res) {
  fs.readFile(__dirname + '/public/rgb.html', function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'}); // Respuesta con estado 404
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'}); // Respuesta con estado 200
    res.write(data); // Envío del contenido del archivo
    return res.end();
  });
}

// Establece la gestión de eventos para la conexión WebSocket.
// Escucha los eventos de 'rgbLed' para recibir datos del cliente y ajustar el PWM de los LEDs según esos datos.
io.sockets.on('connection', function (socket) {
  socket.on('rgbLed', function(data) {
    console.log(data); // Muestra los datos recibidos de la interfaz web
    // Ajusta el color del LED RGB según los datos recibidos
```

```
    redRGB = 255 - parseInt(data.red);
    greenRGB = 255 - parseInt(data.green);
    blueRGB = 255 - parseInt(data.blue);

    let redScaled = scaleTo100(redRGB);
    let greenScaled = scaleTo100(greenRGB);
    let blueScaled = scaleTo100(blueRGB);

    gpio.set_pwm(ledRed, PWM_FREQ, redScaled);
    gpio.set_pwm(ledGreen, PWM_FREQ, greenScaled);
    gpio.set_pwm(ledBlue, PWM_FREQ, blueScaled);
  });
});

// Manejo de la señal SIGINT para una terminación segura y limpieza de los recursos GPIO.
// Desinicializa los pines GPIO y detiene el servidor antes de salir del proceso.
process.on('SIGINT', function () {
  gpio.deinit_gpio(ledRed);
  gpio.deinit_gpio(ledGreen);
  gpio.deinit_gpio(ledBlue);
  process.exit();
});
```

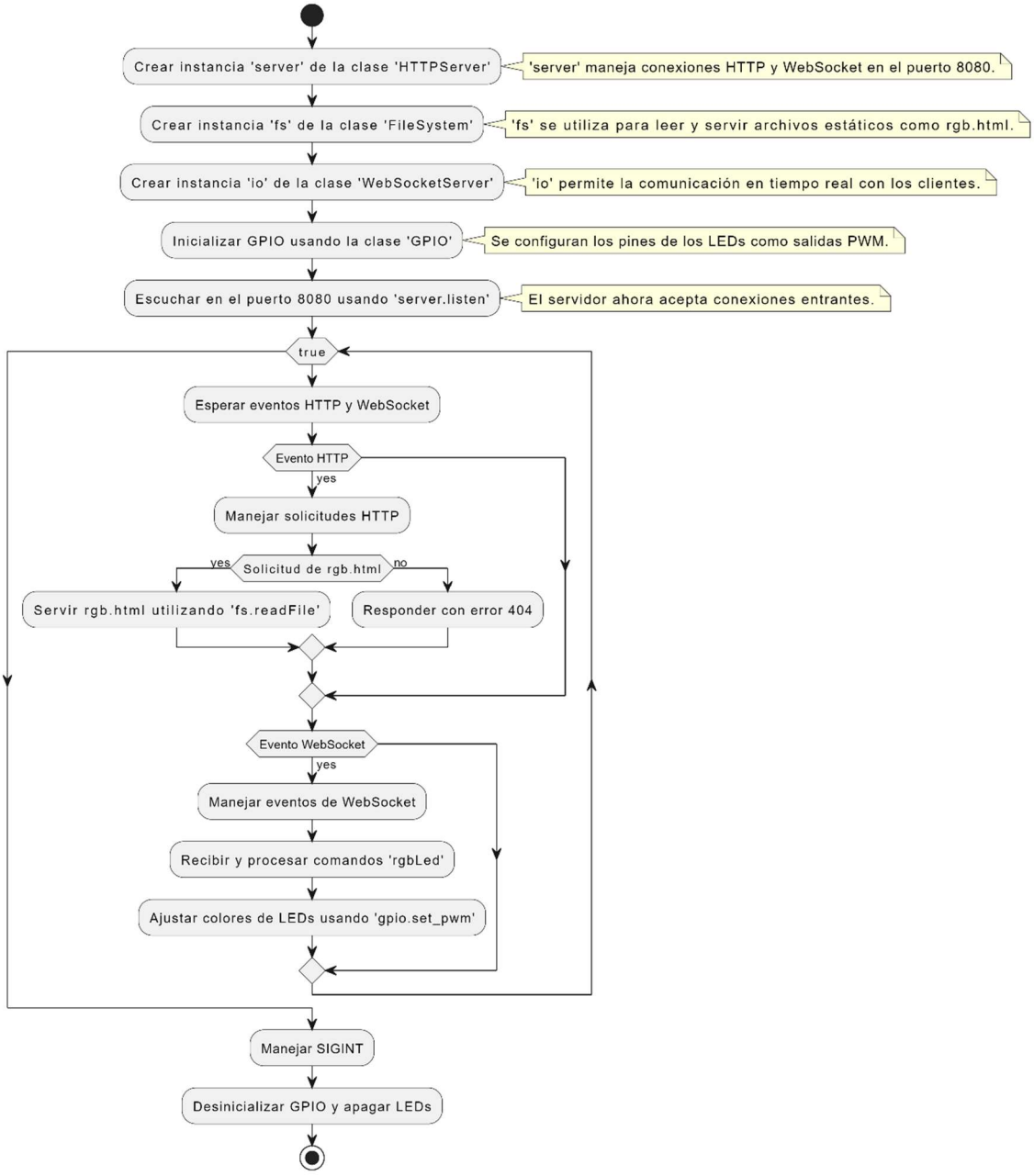

Crear instancia 'server' de la clase HTTPServer
La instancia 'server' maneja las conexiones HTTP en el puerto 8080.
Crear instancia 'fs' de la clase FileSystem
'fs' se utiliza para leer archivos del sistema.
Crear instancia 'io' de la clase WebSocketServer
'io' utiliza 'server' para manejar conexiones WebSocket.
Configurar PWM para los LEDs RGB
Se inicializan los pines GPIO para control PWM.
Iniciar servidor HTTP en el puerto 8080
'server.listen(8080)'.
true
Esperar eventos de red o GPIO
Evento HTTP recibido
true
Manejar solicitud HTTP
yes
Solicitud es para /rgb.html
no
Usar 'fs' para leer y servir rgb.html
Responder con 404 usando 'server'
Enviar rgb.html a través de 'server'
Evento WebSocket recibido
true
Establecer conexión WebSocket
Esperar mensajes del cliente
Mensaje es para cambiar color del LED
yes
Actualizar PWM de LEDs usando GPIO
Manejar señal SIGINT
Limpieza y cierre seguro.
Apagar LEDs y liberar recursos GPIO

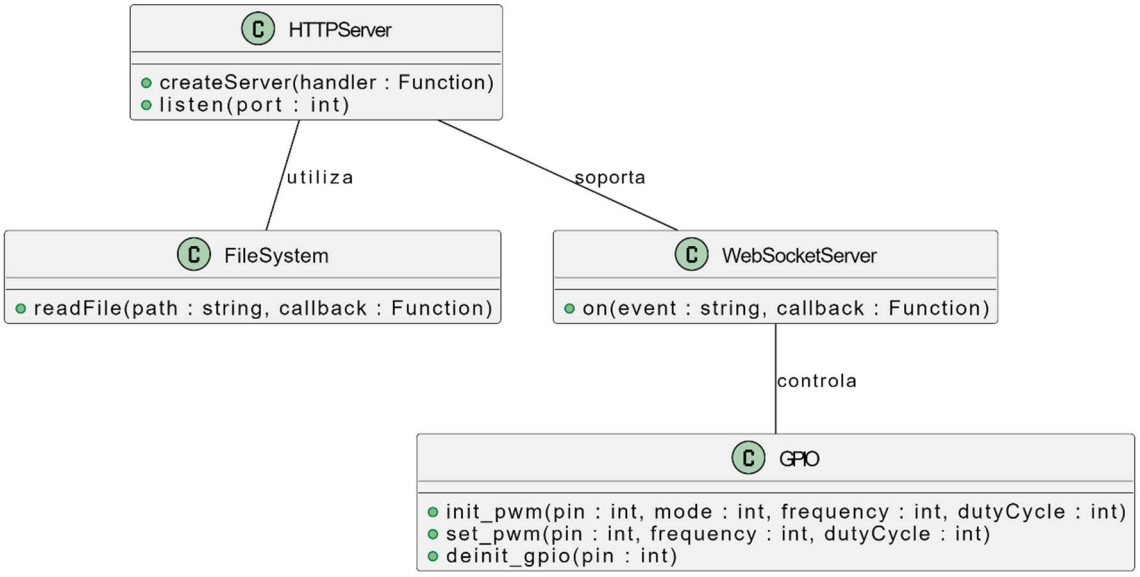

HTTPServer
port : int
listen(port : int) : void
create(handler : Function) : HTTPServer
serves
FileSystem
readFile(path : string, callback : Function) : void
uses
WebSocketServer
server : HTTPServer
on(connection : Function) : void
emit(event : string, data : string) : void
controls
RGBControl
redPin : int
greenPin : int
bluePin : int
gpio : GPIO
updateColor(red : int, green : int, blue : int) : void
scaleTo100(value : int) : int
uses
GPIO
pin : int
mode : string
frequency : int
dutyCycle : int
init_pwm(pin : int, mode : string, frequency : int, dutyCycle : int) : void
set_pwm(pin : int, frequency : int, dutyCycle : int) : void
deinit_gpio(pin : int) : void

serverCathode

```
// Crear un servidor HTTP que escucha en el puerto 8080. 'http' es una instancia de la clase HTTPServer.
// Esta instancia gestiona todas las solicitudes HTTP dirigidas al servidor en el puerto 8080.
const http = require('http').createServer(handler);

// 'fs' es una instancia de la clase FileSystem, usada para operaciones de lectura de archivos.
// Usado para acceder y leer archivos desde el disco, específicamente para servir páginas web a los clientes.
var fs = require('fs');

// 'io' es una instancia de la clase WebSocketServer, configurada para manejar comunicación en tiempo real sobre el servidor HTTP.
// Permite la comunicación en tiempo real con los clientes conectados, usada para recibir y enviar comandos de control del LED RGB.
var io = require('socket.io')(http);

// 'gpio' es una instancia de la clase GPIO, usada para controlar el hardware de los pines GPIO mediante PWM.
// Inicializa los pines GPIO designados para controlar un LED RGB mediante señales PWM.
const gpio = require("@iiot2k/gpiox");

// Configura los pines GPIO para cada color del LED RGB.
// Inicializa los pines como salidas PWM y configura la frecuencia y el ciclo de trabajo inicial.
const ledRed = 22; // Atributo del objeto GPIO para el pin del LED rojo
const ledGreen = 17; // Atributo del objeto GPIO para el pin del LED verde
const ledBlue = 27; // Atributo del objeto GPIO para el pin del LED azul
gpio.init_pwm(ledRed, gpio.GPIO_MODE_PWM, PWM_FREQ, PWM_DUTY_INIT);
gpio.init_pwm(ledGreen, gpio.GPIO_MODE_PWM, PWM_FREQ, PWM_DUTY_INIT);
gpio.init_pwm(ledBlue, gpio.GPIO_MODE_PWM, PWM_FREQ, PWM_DUTY_INIT);

// Activa el servidor para aceptar conexiones HTTP en el puerto especificado.
// Comienza a escuchar las solicitudes entrantes en el puerto 8080.
http.listen(8080);

// Función que maneja las solicitudes HTTP, enviando la página 'rgb.html' o devolviendo un error 404 si no se encuentra.
function handler(req, res) {
  fs.readFile(__dirname + '/public/rgb.html', function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'}); // Respuesta con estado 404
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'}); // Respuesta con estado 200
    res.write(data); // Envío del contenido del archivo
    return res.end();
  });
}

// Establece la gestión de eventos para la conexión WebSocket.
// Escucha los eventos de 'rgbLed' para recibir datos del cliente y ajustar el PWM de los LEDs según esos datos.
io.sockets.on('connection', function (socket) {
  socket.on('rgbLed', function(data) {
    console.log(data); // Muestra los datos recibidos de la interfaz web
    // Ajusta el color del LED RGB según los datos recibidos
    redRGB = parseInt(data.red);
    greenRGB = parseInt(data.green);
    blueRGB = parseInt(data.blue);
```

```
      let redScaled = scaleTo100(redRGB);
      let greenScaled = scaleTo100(greenRGB);
      let blueScaled = scaleTo100(blueRGB);

      gpio.set_pwm(ledRed, PWM_FREQ, redScaled);
      gpio.set_pwm(ledGreen, PWM_FREQ, greenScaled);
      gpio.set_pwm(ledBlue, PWM_FREQ, blueScaled);
    });
});

// Manejo de la señal SIGINT para una terminación segura y limpieza de los recursos GPIO.
// Desinicializa los pines GPIO y detiene el servidor antes de salir del proceso.
process.on('SIGINT', function () {
    gpio.deinit_gpio(ledRed);
    gpio.deinit_gpio(ledGreen);
    gpio.deinit_gpio(ledBlue);
    process.exit();
});
```


Crear instancia 'server' de la clase 'HTTPServer'
'server' maneja conexiones HTTP y WebSocket en el puerto 8080.
Crear instancia 'fs' de la clase 'FileSystem'
'fs' se utiliza para leer y servir archivos estáticos como rgb.html.
Crear instancia 'io' de la clase 'WebSocketServer'
'io' permite la comunicación en tiempo real con los clientes.
Inicializar GPIO usando la clase 'GPIO'
Se configuran los pines de los LEDs como salidas PWM.
Escuchar en el puerto 8080 usando 'server.listen'
El servidor ahora acepta conexiones entrantes.
true
Esperar eventos HTTP y WebSocket
Evento HTTP
yes
Manejar solicitudes HTTP
yes
Solicitud de rgb.html
no
Servir rgb.html utilizando 'fs.readFile'
Responder con error 404
Evento WebSocket
yes
Manejar eventos de WebSocket
Recibir y procesar comandos 'rgbLed'
Ajustar colores de LEDs usando 'gpio.set_pwm'
Manejar SIGINT
Desinicializar GPIO y apagar LEDs


HTTPServer
createServer(handler : Function)
listen(port : int)
utiliza
soporta
FileSystem
readFile(path : string, callback : Function)
WebSocketServer
on(event : string, callback : Function)
controla
GPIO
init_pwm(pin : int, mode : int, frequency : int, dutyCycle : int)
set_pwm(pin : int, frequency : int, dutyCycle : int)
deinit_gpio(pin : int)