ENVIRONMENTAL MONITORING

PHASE-04

SOURCE CODE:

```
#include <JSONVar.h>
#include <Arduino_JSON.h>
#include <JSON.h>
#include <Wire.h>
#include "Adafruit_SGP30.h"
#include "MutichannelGasSensor.h"
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>
#include "SdsDustSensor.h"
#include "ThingSpeak.h"
#include <Arduino.h>
#include "sensirion_common.h"
#include <Adafruit_Sensor.h>
#include "DHT.h"
// Use this file to store all of the private credentials
// and connection details
#define SECRET_CH_ID1 864649 // replace 0000000 with your channel
number
#define SECRET_WRITE_APIKEY1 "D6WO0I37GORJEIV9" // replace
XYZ with your channel write API Key
#define SECRET_CH_ID2 864650 // replace 0000000 with your channel
number
#define SECRET_WRITE_APIKEY2 "DDEGP9X1V4WEGEFH" //
replace XYZ with your channel write API Key
#define SECRET_CH_ID3 864651 // replace 0000000 with your channel
```

```
number
#define SECRET_WRITE_APIKEY3 "MC5M1BZI4U9422XF" // replace
XYZ with your channel write API Key
#define SECRET_CH_ID4 864652 // replace 0000000 with your channel
number
#define SECRET_WRITE_APIKEY4 "1T9T3FK7NR422DJP" // replace
XYZ with your channel write API Key
//#define SECRET_CH_ID1 906528 // replace 0000000 with your
channel number
//#define SECRET_WRITE_APIKEY1 "LL4J2EL6WCIW3SKD" //
replace XYZ with your channel write API Key
//
//#define SECRET_CH_ID2 907653 // replace 0000000 with your
channel number
//#define SECRET_WRITE_APIKEY2 "JIT20STHHFLPYTBD" // replace
XYZ with your channel write API Key
//
//#define SECRET_CH_ID3 907654 // replace 0000000 with your
channel number
//#define SECRET_WRITE_APIKEY3 "XSP8H1C1CD9VDQ2K" //
replace XYZ with your channel write API Key
//
//#define SECRET_CH_ID4 907655 // replace 0000000 with your
channel number
//#define SECRET_WRITE_APIKEY4 "Z5HI2QGCMDXMTGQ0" //
replace XYZ with your channel write API Key
int rxPin = 14;
int txPin = 15;
SdsDustSensor sds(rxPin, txPin);
```

```
uint32_t delayMS;
int x;
// ###################### Update the Wifi SSID, Password and IP adress
of the server ##########
// WIFI params
char* WIFI_SSID = "JioFi_20FDE31";
char* WIFI_PSWD = "n5v406hr5d";
//char* WIFI_SSID = "WPS unavailable";
//char* WIFI_PSWD = "no game no life";
String CSE_IP = "onem2m.iiit.ac.in";
// ########################################################
int WIFI_DELAY = 100; //ms
// oneM2M : CSE params
int CSE_HTTP_PORT = 80;
String CSE_NAME = "in-name";
String CSE_M2M_ORIGIN = "admin:admin";
// oneM2M : resources' params
String DESC_CNT_NAME = "DESCRIPTOR";
String DATA_CNT_NAME = "DATA";
String CMND_CNT_NAME = "COMMAND";
int TY_AE = 2;
int TY_CNT = 3;
int TY_CI = 4;
int TY_SUB = 23;
// HTTP constants
int LOCAL_PORT = 9999;
char* HTTP_CREATED = "HTTP/1.1 201 Created";
char* HTTP_OK = "HTTP/1.1 200 OK\r\n";
int REQUEST_TIME_OUT = 5000; //ms
```

```
//MISC
//int LED_PIN = D1;/
int SERIAL_SPEED = 9600;
#define DEBUG
////////////////////////////////////////
//sensor variables
#define DHTPIN 0 // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
DHT dht(DHTPIN, DHTTYPE);
float dht_val[2];
Adafruit_SGP30 sgp;
////////////////////////////////////////
// Global variables
WiFiServer server(LOCAL_PORT); // HTTP Server (over WiFi). Binded
to listen on LOCAL_PORT contant
WiFiClient client;
String context = "";
String command = ""; // The received command
unsigned long myChannelNumber1 = SECRET_CH_ID1;
const char * myWriteAPIKey1 = SECRET_WRITE_APIKEY1;
unsigned long myChannelNumber2 = SECRET_CH_ID2;
const char * myWriteAPIKey2 = SECRET_WRITE_APIKEY2;
unsigned long myChannelNumber3 = SECRET_CH_ID3;
const char * myWriteAPIKey3 = SECRET_WRITE_APIKEY3;
unsigned long myChannelNumber4 = SECRET_CH_ID4;
const char * myWriteAPIKey4 = SECRET_WRITE_APIKEY4;
String myStatus = "";
// Method for creating an HTTP POST with preconfigured oneM2M headers
// param : url --> the url path of the targted oneM2M resource on the remote
```

```
CSE
// param : ty --> content-type being sent over this POST request (2 for ae, 3
for cnt, etc.)
// param : rep --> the representaton of the resource in JSON format
String doPOST(String url, int ty, String rep) {
 String postRequest = String() + "POST " + url + " HTTP/1.1\r\n" +
 "Host: " + CSE_IP + ":" + CSE_HTTP_PORT + "\r\n" +
 "X-M2M-Origin: " + CSE_M2M_ORIGIN + "\r\n" +
 "Content-Type: application/json;ty=" + ty + "\r\n" +
 "Content-Length: " + rep.length() + "\r\n"
 "Connection: close\r\n\n" +
 rep;
 // Connect to the CSE address
Serial.println("connecting to " + CSE_IP + ":" + CSE_HTTP_PORT + "
...");
 // Get a client
 WiFiClient client;
 if (!client.connect(CSE_IP, CSE_HTTP_PORT)) {
 Serial.println("Connection failed !");
 return "error";
 }
 // if connection succeeds, we show the request to be send
#ifdef DEBUG
 Serial.println(postRequest);
#endif
 // Send the HTTP POST request
 client.print(postRequest);
 // Manage a timeout
 unsigned long startTime = millis();
```

```
 while (client.available() == 0) {
 if (millis() - startTime > REQUEST_TIME_OUT) {
 Serial.println("Client Timeout");
 client.stop();
 return "error";
 }
 }
 // If success, Read the HTTP response
 String result = "";
 if (client.available()) {
 result = client.readStringUntil('\r');
 // Serial.println(result);
 }
 while (client.available()) {
 String line = client.readStringUntil('\r');
 Serial.print(line);
 }
 Serial.println();
 Serial.println("closing connection...");
 return result;
}
// Method for creating an ApplicationEntity(AE) resource on the remote CSE
(this is done by sending a POST request)
// param : ae --> the AE name (should be unique under the remote CSE)
String createAE(String ae) {
 String aeRepresentation =
 "{\"m2m:ae\": {"
 "\"rn\":\"" + ae + "\","
 "\"api\":\"org.demo." + ae + "\","
```

```
 "\"rr\":\"true\","
 "\"poa\":[\"http://" + WiFi.localIP().toString() + ":" + LOCAL_PORT +
"/" + ae + "\"]"
 "}}";
#ifdef DEBUG
 Serial.println(aeRepresentation);
#endif
 return doPOST("/" + CSE_NAME, TY_AE, aeRepresentation);
}
// Method for creating an Container(CNT) resource on the remote CSE under
a specific AE (this is done by sending a POST request)
// param : ae --> the targeted AE name (should be unique under the remote
CSE)
// param : cnt --> the CNT name to be created under this AE (should be
unique under this AE)
String createCNT(String ae, String cnt) {
 String cntRepresentation =
 "{\"m2m:cnt\": {"
 "\"rn\":\"" + cnt + "\","
 "\"min\":\"" + -1 + "\""
 "}}";
 return doPOST("/" + CSE_NAME + "/" + ae, TY_CNT,
cntRepresentation);
}
// Method for creating an ContentInstance(CI) resource on the remote CSE
under a specific CNT (this is done by sending a POST request)
// param : ae --> the targted AE name (should be unique under the remote
CSE)
// param : cnt --> the targeted CNT name (should be unique under this AE)
```

```
// param : ciContent --> the CI content (not the name, we don't give a name
for ContentInstances)
String createCI(String ae, String cnt, String ciContent) {
 String ciRepresentation =
 "{\"m2m:cin\": {"
 "\"con\":\"" + ciContent + "\""
 "}}";
 return doPOST("/" + CSE_NAME + "/" + ae + "/" + cnt, TY_CI,
ciRepresentation);
}
// Method for creating an Subscription (SUB) resource on the remote CSE
(this is done by sending a POST request)
// param : ae --> The AE name under which the SUB will be created .(should
be unique under the remote CSE)
// The SUB resource will be created under the COMMAND container
more precisely.
String createSUB(String ae) {
 String subRepresentation =
 "{\"m2m:sub\": {"
 "\"rn\":\"SUB_" + ae + "\","
 "\"nu\":[\"" + CSE_NAME + "/" + ae + "\"], "
 "\"nct\":1"
 "}}";
 return doPOST("/" + CSE_NAME + "/" + ae + "/" + CMND_CNT_NAME,
TY_SUB, subRepresentation);
}
// Method to register a module (i.e. sensor or actuator) on a remote oneM2M
CSE
void registerModule(String module, bool isActuator, String
```

```
intialDescription, String initialData) {
 if (WiFi.status() == WL_CONNECTED) {
 String result;
 // 1. Create the ApplicationEntity (AE) for this sensor
 result = createAE(module);
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("AE " + module + " created !");
#endif
 // 2. Create a first container (CNT) to store the description(s) of the
sensor
 result = createCNT(module, DESC_CNT_NAME);
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("CNT " + module + "/" + DESC_CNT_NAME + "
created !");
#endif
 // Create a first description under this container in the form of a
ContentInstance (CI)
 result = createCI(module, DESC_CNT_NAME, intialDescription);
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("CI " + module + "/" + DESC_CNT_NAME +
"/{initial_description} created !");
#endif
 }
 }
 // 3. Create a second container (CNT) to store the data of the sensor
 result = createCNT(module, DATA_CNT_NAME);
```

```
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("CNT " + module + "/" + DATA_CNT_NAME + "
created !");
#endif
 // Create a first data value under this container in the form of a
ContentInstance (CI)
 result = createCI(module, DATA_CNT_NAME, initialData);
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("CI " + module + "/" + DATA_CNT_NAME +
"/{initial_aata} created !");
#endif
 }
 }
 // 3. if the module is an actuator, create a third container (CNT) to store
the received commands
 if (isActuator) {
 result = createCNT(module, CMND_CNT_NAME);
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("CNT " + module + "/" + CMND_CNT_NAME + "
created !");
#endif
 // subscribe to any ne command put in this container
 result = createSUB(module);
 if (result == HTTP_CREATED) {
#ifdef DEBUG
 Serial.println("SUB " + module + "/" + CMND_CNT_NAME +
```

```
"/SUB_" + module + " created !");
#endif
 }
 }
 }
 }
 }
}
void init_WiFi() {
 Serial.println("Connecting to " + String(WIFI_SSID) + " ...");
 WiFi.persistent(false);
 WiFi.begin(WIFI_SSID, WIFI_PSWD);
 // wait until the device is connected to the wifi network
 while (WiFi.status() != WL_CONNECTED) {
 delay(WIFI_DELAY);
 Serial.print(".");
 }
 // Connected, show the obtained ip address
 Serial.println("WiFi Connected ==> IP Address = " +
WiFi.localIP().toString());
}
void init_HTTPServer() {
 server.begin();
 Serial.println("Local HTTP Server started !");
}
void task_HTTPServer() {
 // Check if a client is connected
 client = server.available();
 if (!client)
```

```
 return;
 // Wait until the client sends some data
 Serial.println("New client connected. Receiving request... ");
 while (!client.available()) {
#ifdef DEBUG_MODE
 Serial.print(".");
#endif
 delay(5);
 }
```

SOURCE CODE OUTPUT DIAGRAM: