CLOUD PLC DIGITAL INPUT WITH SNMP IMPLEMENTATION

SNMP (Simple Network Management Protocol)

SNMP (Simple Network Management Protocol) is a standard protocol used for managing and monitoring devices on IP networks. It enables network administrators to manage network performance, find and solve network problems, and plan for network growth. SNMP is widely used in network management systems to monitor network-attached devices such as routers, switches, servers, workstations, printers, and more.

This user manual provides comprehensive guidance for setting up and using the SNMP manager and SNMP Agent program on an ESP32/ESP8266 Device. The program is designed to query SNMP agent on a network and retrieve information such as the device name, uptime, IO status.

System Requirement

Hardware: ESP32 or ESP8266 microcontroller
Software: Arduino IDE (version 1.8.13 or higher)
Libraries:
SNMP.h :**https://drive.google.com/drive/folders/13vvRUq90Ef12RryM2d4bf_nnIdmYgB4D?usp=sharing,**
SNMP_ Manager.h : **https://drive.google.com/drive/folders/1k1CzKECl9iiWrAwne7puH4cTSpgIOfCI?usp=sharing,**
SNMP_Agent.h : https://drive.google.com/drive/folders/1ktdFpn-KRiVwhGf8ro5JliIw_FLpb55s?usp=sharing
Network: Wi-Fi network with access to the SNMP agents.

Configuration

Wi-Fi settings:

Set your Wi-Fi SSID and Password in the Following lines:

```
const char *ssid = "YOUR_SSID";

const char *password = "YOUR_PASSWORD";
```

SNMP Settings:

Set the SNMP community string and version:

```
const char *community = "public";

const int snmpVersion = 1; // 1 for SNMPv1, 2 for SNMPv2c
```

Firewall and Network Configuration

To ensure proper communication between the SNMP Manager and SNMP Agents, you may need to configure your network's firewall and security settings. Below are some key considerations and steps for setting up your firewall and network:

- Firewall Settings

Allow SNMP Traffic:

- SNMP uses the User Datagram Protocol (UDP) and commonly operates on port 161 for SNMP messages and port 162 for SNMP traps. Ensure that these ports are open on both the SNMP Manager and Agent devices.

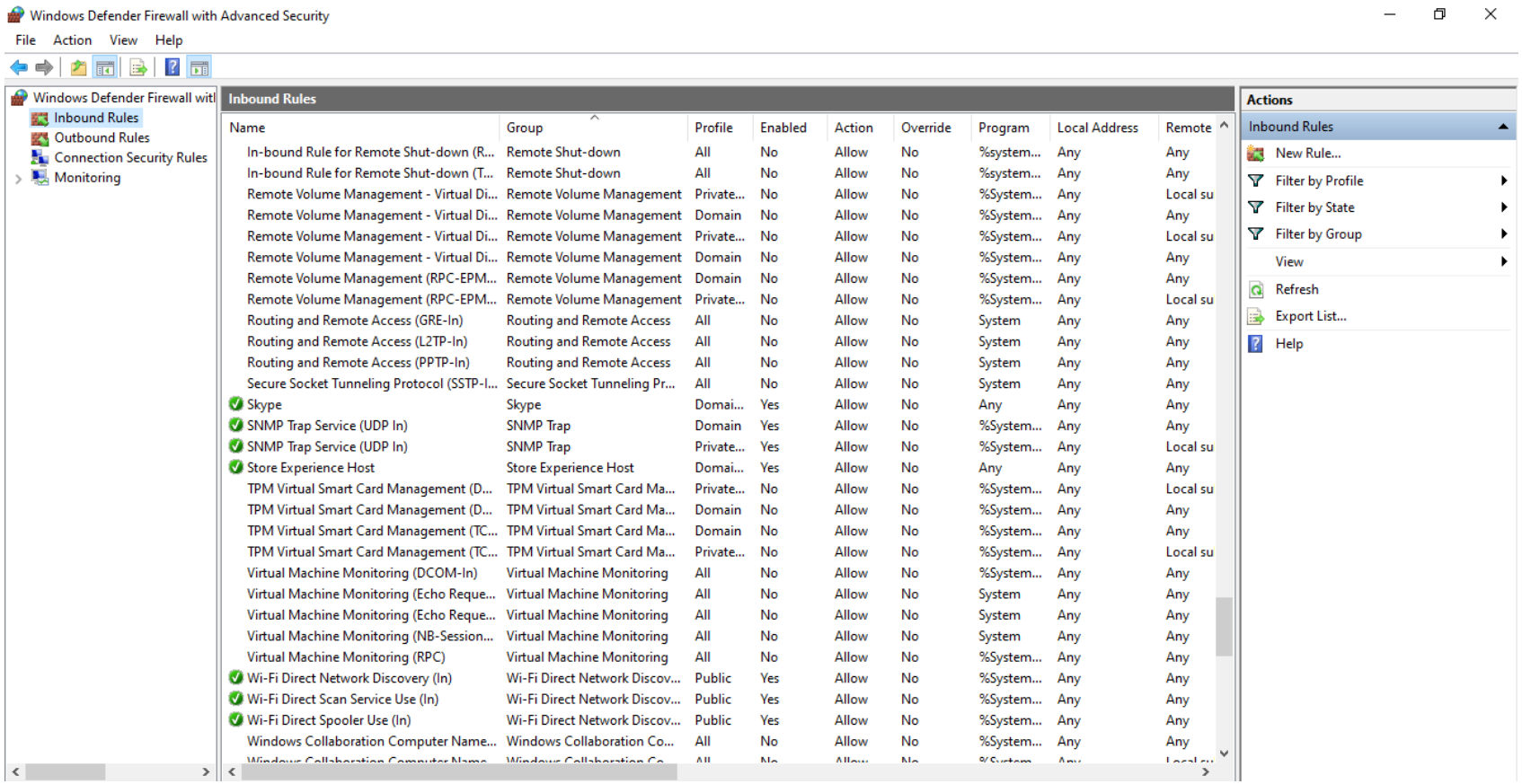
Inbound Rules:

- Allow incoming UDP traffic on port 161 to the SNMP Manager.

  If traps are being used, allow incoming UDP traffic on port 162.

Outbound Rules:

- Allow outgoing UDP traffic on port 161 from the SNMP Manager to the SNMP Agents.
- Allow outgoing UDP traffic on port 162 if the SNMP Manager is sending traps to a management system.


Windows Defender Firewall with Advanced Security
File Action View Help
Windows Defender Firewall witl
Inbound Rules
Outbound Rules
Connection Security Rules
Monitoring
Inbound Rules
Name | Group | Profile | Enabled | Action | Override | Program | Local Address | Remote
In-bound Rule for Remote Shut-down (R... | Remote Shut-down | All | No | Allow | No | %system... | Any | Any
In-bound Rule for Remote Shut-down (T... | Remote Shut-down | All | No | Allow | No | %system... | Any | Any
Remote Volume Management - Virtual Di... | Remote Volume Management | Private... | No | Allow | No | %System... | Any | Local su
Remote Volume Management - Virtual Di... | Remote Volume Management | Domain | No | Allow | No | %System... | Any | Any
Remote Volume Management - Virtual Di... | Remote Volume Management | Private... | No | Allow | No | %System... | Any | Local su
Remote Volume Management - Virtual Di... | Remote Volume Management | Domain | No | Allow | No | %System... | Any | Any
Remote Volume Management (RPC-EPM... | Remote Volume Management | Domain | No | Allow | No | %System... | Any | Any
Remote Volume Management (RPC-EPM... | Remote Volume Management | Private... | No | Allow | No | %System... | Any | Local su
Routing and Remote Access (GRE-In) | Routing and Remote Access | All | No | Allow | No | System | Any | Any
Routing and Remote Access (L2TP-In) | Routing and Remote Access | All | No | Allow | No | System | Any | Any
Routing and Remote Access (PPTP-In) | Routing and Remote Access | All | No | Allow | No | System | Any | Any
Secure Socket Tunneling Protocol (SSTP-I... | Secure Socket Tunneling Pr... | All | No | Allow | No | System | Any | Any
Skype | Skype | Domai... | Yes | Allow | No | Any | Any | Any
SNMP Trap Service (UDP In) | SNMP Trap | Domain | Yes | Allow | No | %System... | Any | Any
SNMP Trap Service (UDP In) | SNMP Trap | Private... | Yes | Allow | No | %System... | Any | Local su
Store Experience Host | Store Experience Host | Domai... | Yes | Allow | No | Any | Any | Any
TPM Virtual Smart Card Management (D... | TPM Virtual Smart Card Ma... | Private... | No | Allow | No | %System... | Any | Local su
TPM Virtual Smart Card Management (D... | TPM Virtual Smart Card Ma... | Domain | No | Allow | No | %System... | Any | Any
TPM Virtual Smart Card Management (TC... | TPM Virtual Smart Card Ma... | Domain | No | Allow | No | %System... | Any | Any
TPM Virtual Smart Card Management (TC... | TPM Virtual Smart Card Ma... | Private... | No | Allow | No | %System... | Any | Local su
Virtual Machine Monitoring (DCOM-In) | Virtual Machine Monitoring | All | No | Allow | No | %System... | Any | Any
Virtual Machine Monitoring (Echo Reque... | Virtual Machine Monitoring | All | No | Allow | No | System | Any | Any
Virtual Machine Monitoring (Echo Reque... | Virtual Machine Monitoring | All | No | Allow | No | System | Any | Any
Virtual Machine Monitoring (NB-Session... | Virtual Machine Monitoring | All | No | Allow | No | System | Any | Any
Virtual Machine Monitoring (RPC) | Virtual Machine Monitoring | All | No | Allow | No | %System... | Any | Any
Wi-Fi Direct Network Discovery (In) | Wi-Fi Direct Network Discov... | Public | Yes | Allow | No | %System... | Any | Any
Wi-Fi Direct Scan Service Use (In) | Wi-Fi Direct Network Discov... | Public | Yes | Allow | No | %System... | Any | Any
Wi-Fi Direct Spooler Use (In) | Wi-Fi Direct Network Discov... | Public | Yes | Allow | No | %System... | Any | Any
Windows Collaboration Computer Name... | Windows Collaboration Co... | All | No | Allow | No | %System... | Any | Any
Actions
Inbound Rules
New Rule...
Filter by Profile
Filter by State
Filter by Group
View
Refresh
Export List...
Help

SNMP Manager:

- This code implements an SNMP Manager on an ESP32 or ESP8266 device.
- The manager connects to a WiFi network and periodically sends SNMP Get requests to a range of IP addresses within a specified subnet to retrieve information such as device '**name, uptime, and IO status"**. The SNMP responses are processed and stored in a device records array, which is then printed to the serial monitor.

Key Components:

1. WiFi Connection: The device connects to a specified WiFi network using provided SSID and password.
2. SNMP Manager: It initializes the SNMP manager and sets up a callback mechanism for handling SNMP responses.
3. Device Polling: It sends SNMP requests to devices within a specified IP range to fetch OID values such as system name, uptime, and IO status.
4. Data Storage and Display: The received data is stored in a structured format and displayed on the serial monitor.

Program:

```
#if defined(ESP8266)
#include <ESP8266WiFi.h> // ESP8266 Core WiFi Library
#else
#include <WiFi.h> // ESP32 Core WiFi Library
#endif

#include <WiFiUdp.h>
#include <Arduino_SNMP_Manager.h>

//************************************
//* Your WiFi info                   *
//************************************
const char *ssid = "your ssid";
const char *password = "your password";
//************************************

//************************************
//* SNMP Device Info                 *
//************************************
const char *community = "public"; // SNMP Community string
const int snmpVersion = 1;        // Enum, SNMP Version 1 = 0, SNMP Version 2 = 1
// OIDs
const char *oidSysName = ".1.3.6.1.2.1.1.5.0"; // OctetString SysName
const char *oidUptime = ".1.3.6.1.2.1.1.3.0";  // TimeTicks uptime (hundredths of seconds)
const char *oidIOStatus = ".1.3.6.1.4.1.5.13"; // Custom OID for IO status
//************************************

//************************************
//* Settings                         *
//************************************
int devicePollInterval = 100;    // delay in milliseconds
```

```
int lastDeviceWaitPeriod = 5000; // delay in milliseconds
#define LOWEROCTETLIMIT 190      // Set the lowest IP address to 1, .0 typically isn't used and isn't well
supported.
#define UPPEROCTETLIMIT 199      // Set the upper limit of the range of IPs to query
//**********************************

//**********************************
//* Initialise                     *
//**********************************
// Structures
struct device
{
  IPAddress address;
  char name[50];
  char *sysName = name; // StringHandler needs pointer to char*
  unsigned int uptime;
  int ioStatus;
}; // Structure for the device records

// Global Variables
struct device deviceRecords[UPPEROCTETLIMIT + 1]; // Array of device records. _1 as we're not using the 0 index in
the array.
int lastOctet = LOWEROCTETLIMIT;                  // Initialise last octet to lowest IP
bool allDevicesPolled = false;                    // Flag to indicate all devices have been sent SNMP requests. Note
responses may not yet have arrived.
// Initialise variables used for timer counters to zero.
unsigned long devicePollStart = 0;
unsigned long intervalBetweenDevicePolls = 0;
unsigned long intervalBetweenLastDeviceReadyPolls = 0;
unsigned long deviceReadyStart = 0;

// SNMP Objects
WiFiUDP udp;                                          // UDP object used to send and receive packets
SNMPManager snmp = SNMPManager(community);            // Starts an SNMPManager to listen to replies to get-requests
SNMPGet snmpRequest = SNMPGet(community, snmpVersion); // Starts an SNMPGet instance to send requests
ValueCallback *callbackSysName;                       // Callback pointer for each OID
ValueCallback *callbackUptime;                        // Callback pointer for each OID
ValueCallback *callbackIOStatus;                      // Callback pointer for IO status
//**********************************
```

```
//************************************
//* Function declarations            *
//************************************
void sendSNMPRequest(IPAddress, struct device *deviceRecord);
int getNextOctet(int current);
void printVariableValues();
//************************************

void setup()
{
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.printf("\nConnected to SSID: %s - IP Address: ", ssid);
  Serial.println(WiFi.localIP());

  snmp.setUDP(&udp); // give snmp a pointer to the UDP object
  snmp.begin();      // start the SNMP Manager
}

void loop()
{
  snmp.loop();                                                    // Needs to be called frequently to process
incoming SNMP responses.
  intervalBetweenDevicePolls = millis() - devicePollStart;        // Timer for triggering per device Polls
  intervalBetweenLastDeviceReadyPolls = millis() - deviceReadyStart; // Timer to trigger end of polling all devices
(allowing some time for final packets to be processed)
  if (allDevicesPolled)
  {
    if (intervalBetweenLastDeviceReadyPolls > lastDeviceWaitPeriod) // Waiting time after all devices polled
complete?
    {
```

```
      deviceReadyStart += lastDeviceWaitPeriod; // This prevents drift in the delays
      printVariableValues();                    // Print the values to the serial console
      allDevicesPolled = false;                 // Reset the flag
    }
  }
  else
  {
    if (intervalBetweenDevicePolls >= devicePollInterval) // Time to poll the next device?
    {
      devicePollStart += devicePollInterval;                // This prevents drift in the delays
      IPAddress deviceIP(192, 168, 137, lastOctet);         // Set IP address to be queried. Note: This simple
example will only work with the last Octet of the address changing as this is used as a simple index for the device
records.
      sendSNMPRequest(deviceIP, &deviceRecords[lastOctet]); // Function call to send SNMP requests to specified
device. Values will be stored in the deviceRecords array when they are returned (async)
      lastOctet = getNextOctet(lastOctet);                  // Update to the next IP address to be queried
    }
  }
}

void sendSNMPRequest(IPAddress target, struct device *deviceRecord)
{
  Serial.print("sendSNMPRequest - target: ");
  Serial.println(target);
  deviceRecord->address = target;
  // Get callbacks from creating a handler for each of the OID
  callbackSysName = snmp.addStringHandler(target, oidSysName, &deviceRecord->sysName);
  callbackUptime = snmp.addTimestampHandler(target, oidUptime, &deviceRecord->uptime);
  callbackIOStatus = snmp.addIntegerHandler(target, oidIOStatus, &deviceRecord->ioStatus);

  // Build a SNMP get-request add each OID to the request
  snmpRequest.addOIDPointer(callbackSysName);
  snmpRequest.addOIDPointer(callbackUptime);
  snmpRequest.addOIDPointer(callbackIOStatus);

  snmpRequest.setIP(WiFi.localIP()); // IP of the listening MCU
  snmpRequest.setUDP(&udp);
  snmpRequest.setRequestID(rand() % 5555);
  snmpRequest.sendTo(target);
```

```
  snmpRequest.clearOIDList();
}

int getNextOctet(int current)
{
  if (current == UPPEROCTETLIMIT)
  {
    allDevicesPolled = true;
    return LOWEROCTETLIMIT;
  }
  return current + 1;
}

void printVariableValues()
{
  int i;
  for (i = LOWEROCTETLIMIT; i <= UPPEROCTETLIMIT; i++)
  {
    Serial.print("Address: ");
    Serial.print(deviceRecords[i].address);
    Serial.print(" - Name: ");
    Serial.print(deviceRecords[i].name);
    Serial.print(" - Uptime: ");
    Serial.print(deviceRecords[i].uptime);
    Serial.print(" - IO Status: ");
    Serial.println(deviceRecords[i].ioStatus == 1 ? "HIGH" : "LOW");
  }
}
```

SNMP Agent:

- This code sets up an SNMP agent on an ESP32 or ESP8266 device, allowing it to respond to SNMP Get requests and send SNMP Traps.
- It connects to a specified WiFi network and configures SNMP to manage various OIDs, including the device name, uptime, and digital input status.
- The agent can also send SNMP traps to notify a manager of changes, such as when a monitored integer value changes.

```
#if defined(ESP8266)
#include <ESP8266WiFi.h>        // ESP8266 Core WiFi Library
#else
#include <WiFi.h>               // ESP32 Core WiFi Library
#endif

#include <WiFiUdp.h>
#include <SNMP_Agent.h>
#include <SNMPTrap.h>

const char* ssid = "your ssid";
const char* password = "your password";

WiFiUDP udp;
// Starts an SNMPAgent instance with the read-only community string 'public', and read-write community string
'private'
SNMPAgent snmp = SNMPAgent("public", "private");

// Numbers used to respond to Get requests
int changingNumber = 1;
int settableNumber = 0;
uint32_t tensOfMillisCounter = 0;

// Arbitrary data will be stored here to act as an OPAQUE data-type
uint8_t* stuff = 0;

// If we want to change the functionality of an OID callback later, store them here
ValueCallback* changingNumberOID;
```

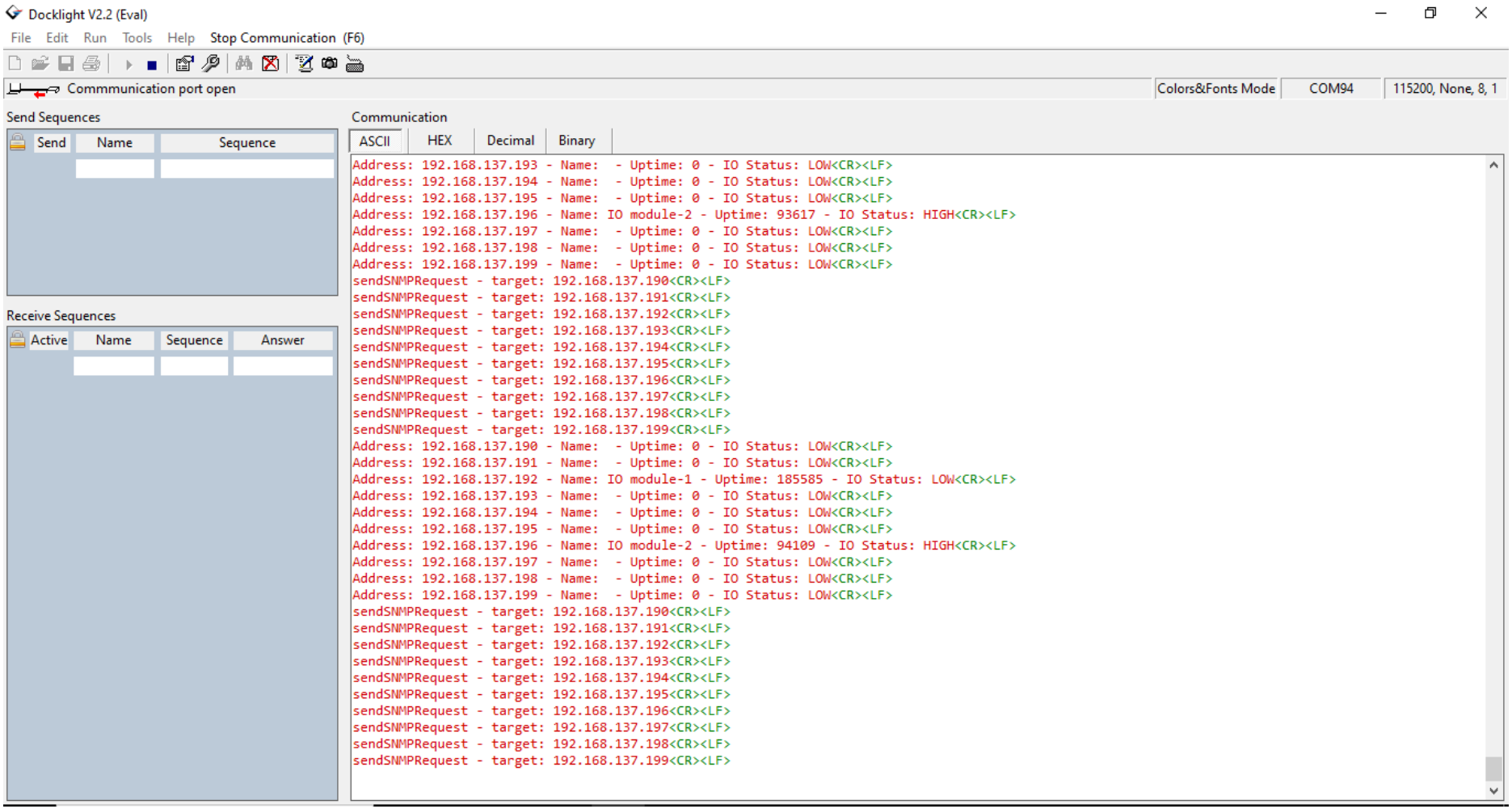
```
ValueCallback* settableNumberOID;
TimestampCallback* timestampCallbackOID;

// Static string
std::string staticString = "This value will never change";

// Setup an SNMPTrap for later use
SNMPTrap* settableNumberTrap = new SNMPTrap("public", SNMP_VERSION_2C);
char* changingString;

const char* deviceName = "IO module-2";
uint32_t uptimeSeconds = 0;
const int ioPin = 36;// digital  input pin
int ioStatus = 0; // 0 for LOW, 1 for HIGH

void setup() {
  Serial.begin(115200);
  IPAddress local_IP(192, 168, 137, 196);
  IPAddress subnet(255, 255, 255, 0);
  IPAddress gateway(192, 168, 137, 1);
  IPAddress primaryDNS(192, 168, 137, 1); // optional
  IPAddress secondaryDNS(192, 168, 137, 1);

  // Connect to WiFi using the configured IP
  delay(500);
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA Failed to configure");
  }
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
```

```
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Give SNMP a pointer to the UDP object
snmp.setUDP(&udp);
snmp.begin();

// Setup our OPAQUE data-type
stuff = (uint8_t*)malloc(4);
stuff[0] = 1;
stuff[1] = 2;
stuff[2] = 24;
stuff[3] = 67;

// Add callback for the standard sysName OID
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.2.1.1.5.0", deviceName);

// Add callback for custom OIDs
changingNumberOID = snmp.addIntegerHandler(".1.3.6.1.4.1.5.0", &changingNumber);
settableNumberOID = snmp.addIntegerHandler(".1.3.6.1.4.1.5.1", &settableNumber, true);
snmp.addIntegerHandler(".1.3.6.1.4.1.4.0", &changingNumber);
snmp.addOpaqueHandler(".1.3.6.1.4.1.5.9", stuff, 4, true);
snmp.addReadOnlyStaticStringHandler(".1.3.6.1.4.1.5.11", staticString);

// Add handler for the digital input pin status
snmp.addIntegerHandler(".1.3.6.1.4.1.5.13", &ioStatus);

// Setup read/write string
changingString = (char*)malloc(25 * sizeof(char));
snprintf(changingString, 25, "This is changeable");
snmp.addReadWriteStringHandler(".1.3.6.1.4.1.5.12", &changingString, 25, true);

// Setup SNMP TRAP
timestampCallbackOID = (TimestampCallback*)snmp.addTimestampHandler(".1.3.6.1.2.1.1.3.0", &tensOfMillisCounter);
settableNumberTrap->setUDP(&udp); // Give a pointer to our UDP object
settableNumberTrap->setTrapOID(new OIDType(".1.3.6.1.2.1.33.2")); // OID of the trap
settableNumberTrap->setSpecificTrap(1);

// Set the uptime counter to use in the trap (required)
```

```
  settableNumberTrap->setUptimeCallback(timestampCallbackOID);

  // Set some previously set OID Callbacks to send these values with the trap (optional)
  settableNumberTrap->addOIDPointer(changingNumberOID);
  settableNumberTrap->addOIDPointer(settableNumberOID);

  settableNumberTrap->setIP(WiFi.localIP()); // Set our Source IP

  // Ensure to sortHandlers after adding/removing and OID callbacks - this makes snmpwalk work
  snmp.sortHandlers();

  // Initialize IO pin
  pinMode(ioPin, INPUT);
}

void loop() {
  snmp.loop(); // Must be called as often as possible

  // Update IO status (0 for LOW, 1 for HIGH)
  ioStatus = digitalRead(ioPin);

  // Update uptime
  uptimeSeconds = millis() / 1000;

  if (settableNumberOID->setOccurred) {
    Serial.printf("Number has been set to value: %i\n", settableNumber);
    if (settableNumber % 2 == 0) {
      // Sending an SNMPv2 INFORM (trap will be kept and re-sent until it is acknowledged by the IP address it was
sent to)
      settableNumberTrap->setVersion(SNMP_VERSION_2C);
      settableNumberTrap->setInform(true); // Set this to false and send using `settableNumberTrap->sendTo` to send
it without the INFORM request
    } else {
      // Sending regular SNMPv1 trap
      settableNumberTrap->setVersion(SNMP_VERSION_1);
      settableNumberTrap->setInform(false);
    }
    settableNumberOID->resetSetOccurred();
```

```
    // Send the trap to the specified IP address
    IPAddress destinationIP = IPAddress(192, 168, 173, 71);
    if (snmp.sendTrapTo(settableNumberTrap, destinationIP, true, 2, 5000) != INVALID_SNMP_REQUEST_ID) {
      Serial.println("Sent SNMP Trap");
    } else {
      Serial.println("Couldn't send SNMP Trap");
    }
  }
  changingNumber++;
  tensOfMillisCounter = millis() / 10;
}
```

SNMP Monitoring on Serial Monitor:



In this program, SNMP (Simple Network Management Protocol) is used to monitor and manage devices within a network. Here's a brief explanation of how SNMP is utilized:

1. **SNMP Manager Initialization**:
    - An SNMP Manager (SNMPManager snmp) is created to send and receive SNMP messages. It is initialized with a community string ("public") and the specified SNMP version (v1).
2. **Polling Devices**:
    - The program continuously sends SNMP Get requests to devices within a specified IP range. This is done using the sendSNMPRequest function, which targets each device's IP address and queries specific Object Identifiers (OIDs) for information.
    - The OIDs being queried include:
        - oidSysName: The device's system name (a static identifier).
        - oidUptime: The device's uptime (in hundredths of seconds since the device was last rebooted).
        - oidIOStatus: A custom OID representing the status of a digital IO pin (0 for LOW, 1 for HIGH).
3. **Handling Responses**:
    - The program registers callback functions for each OID, allowing it to handle incoming SNMP responses asynchronously. These callbacks update the corresponding fields in the deviceRecords array with the data received from each device.
4. **Data Collection and Display**:
    - The collected data (device name, uptime, and IO status) is stored in the deviceRecords array. After polling all devices, the program waits for any final responses and then prints the collected information to the serial monitor.

Practical Role of SNMP in the Program:

- **Device Monitoring**: The SNMP manager collects vital information (like uptime and IO status) from various devices, allowing network administrators to monitor the status and performance of these devices.
- **Network Management**: The program can be used to identify issues, such as devices that have been restarted (from uptime) or changes in device status (from IO status), enabling proactive maintenance and troubleshooting.
- **Centralized Data Gathering**: By querying multiple devices and storing the results, the SNMP manager provides a centralized view of the network's status, simplifying management and analysis.