Part 18
-
SNMP

Version: 2021-04-07

What is SNMP?

SNMP stands for Simple Network Monitoring Protocol. It is a protocol for management information transfer in networks
Its usefulness in network administration comes from the fact that it allows information about network-connected devices to be collected in a standardized way across a large variety of hardware and software types.
Hardly any network admin renounces SNMP. Rather, most of them confidently rely on it because nearly all kinds of devices from many different manufacturers support SNMP, which helps them achieve comprehensive monitoring thanks to the SNMP technology.

Versions

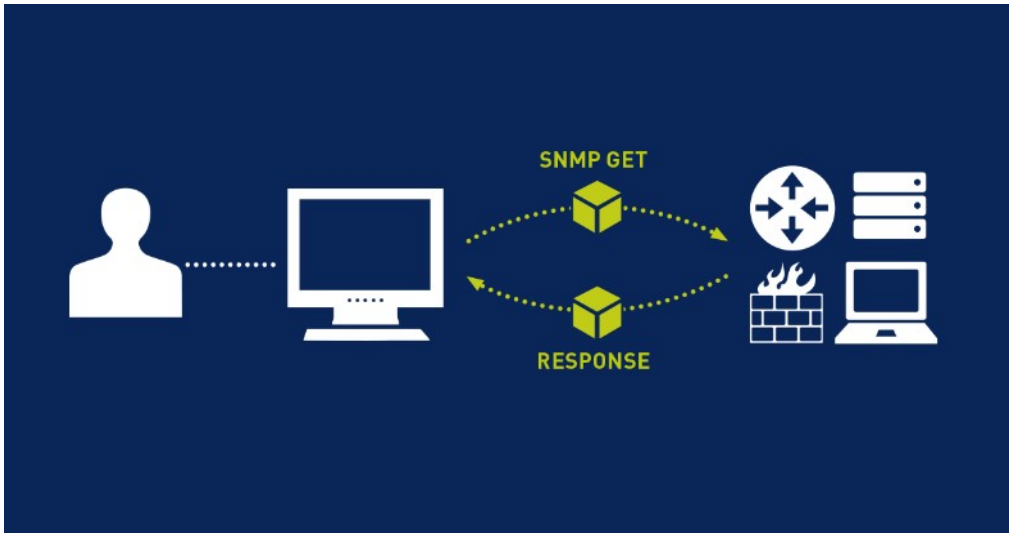
Currently, there are three major versions of SNMP. The first version was developed rather quickly in the late ´80s when network administration lacked suitable network administration tools that were not dependent on hardware manufacturers.
SNMP v1 was defined in 1988 and was based on SGMP (RFC 1028). Then, it was broadly accepted and used. It is still used today, almost 30 years later, which is nearly an eternity in IT. SNMP v1 provides the basic functionalities for data polling and is relatively easy to use. It doesn't create much overhead because it doesn't include any encryption algorithms. So, for security reasons, use SNMP v1 in LANs only. Its biggest limitation is its 32-bit counter architecture, which is not enough for today's gigabyte networks or larger.
If users want to manage networks in a WAN, the CMISE/CMIP protocol is the right protocol to go for.
SNMP v2 supports 64-bit counters but still sends critical data as clear text, so it does not really enhance security. And if users come across SNMP v2, it is mostly "SNMP v2c" that manufacturers or networkers are talking about, with the "c" standing for "community". Two other SNMP v2 versions exist, SNMP v2p and SNMP v2u, but they are only implemented in rare cases.

Credentials for SNMP devices

Defined in 2002, SNMP v3 includes the advantages of SNMP v2c and adds security solutions like user accounts, authentication, and optional encryption of data packages. This enhances security and makes SNMP v3 the recommended SNMP version when it comes to security. However, it also makes configuration more difficult, specifically user management. Therefore, much more processing power is needed, especially with short monitoring intervals that create a great number of SNMP messages.
SNMP v3 has three different security levels:

- *NoAuthNoPriv* – Stands for No Authentication, No Privacy. No authentication is required and messages are not encrypted. For obvious reasons, this should only be used in closed, secure networks.
- *AuthNoPriv* – Stands for Authentication, No Privacy. Messages must be authenticated to be acted upon; however, they are not encrypted during transmission. Theoretically, a malicious actor could still intercept data that was sent between agent and manager during authorized transmissions but could not introduce additional Get or Set requests.
- *AuthPriv* – Stands for Authentication and Privacy. This is the most secure implementation of SNMPv3. SNMP messages must be authenticated and all data is encrypted during transmission. This way, a malicious actor is prevented from sending their own Get or Set requests and from seeing the data generated by legitimate requests.

How does SNMP work?
A network normally has at least one computer or server running monitoring software. It is the managing entity. A network will also most probably have a few, or many, or even *really* many, other devices: switches, routers, workstations, server racks, printers, coffee machines, or anything else that needs to be monitored. They are the managed devices.
SNMP messages are sent and received between so-called managers and agents. Usually, the SNMP manager in the network is installed on the managing entity and the SNMP agents are installed on the managed devices.



These are the main runtime components in an SNMP-enabled environment:

- *SNMP-managed devices and resources :* These are the devices and network elements on which an agent runs.
- *SNMP agent :* This software runs on the hardware or service being monitored by SNMP, collecting data on various metrics like CPU usage, bandwidth usage or disk space. As queried by the SNMP manager, the agent finds and sends this information back to SNMP management systems.
- *SNMP manager :* also referred to as SNMP server. This component functions as a centralized management station running an SNMP management application on many different operating system environments. It actively requests agents send SNMP updates at regular intervals.

Basically, the transfer of SNMP messages can be compared to the typical communication between a client and a server, offering both pull and push technologies. The pull (or poll) technology is the most common communication type where a server, like the network management software on the managing entity, sends out a request to solicit a response from a agent, or managed device. Its counterpart, the push technology, allows the managed device to "speak" and send out an SNMP message upon an event.

SNMP message types
There are different types of SNMP messages that can be used to set up network monitoring via SNMP:

- *GetRequest* – This is the most common SNMP message that an SNMP manager sends out to request data. The targeted device returns the requested value with a Response message.
- *GetNextRequest* – The SNMP manager can send this message type to discover what information is available from the device. By starting at OID 0, the manager can continue to send a request for the next available data until there is no more "next" data. This way, users can discover all of the available data on a certain device even though they might not have had any prior knowledge of the responding system or device.
- *GetBulkRequest* – Added in SNMP Version 2, this is a newer, optimized version of GetNextRequest. The solicited Response will contain as much data as allowed by the request. Essentially, this is a way to do several GetNextRequests at once, which enables users to create a list of all available data and parameters.
- *SetRequest* – This is a manager-initiated command to set or change the value of a parameter via SNMP on the agent device or system. This message type can be used to manage or update configuration settings or other parameters. But be careful! An incorrect SetRequest may seriously damage systems and network setups.
- *Response* – The Response is the message that a device agent sends upon a Request from the manager. When sent in response to a GetRequest type, the packet contains the requested data or values. In the case of a SetRequest, the packet responds with the newly set value as a confirmation that the SetRequest has been completed successfully.
- *Trap(v2)* – A trap is sent ("pushed out") by the SNMP agent without having been requested by the manager. Rather, traps are sent upon determined conditions, such as in the event of an error, or upon crossing a preset threshold. If users want to benefit from traps for monitoring, which is an excellent idea in terms of proactive monitoring, they might have to configure traps first with the help of the SNMP manager.
- *InformRequest* – This message type was added in SNMP v2 to give the manager the possibility to confirm that it received an agent's trap message. Some agents are configured to continue to send a trap until an inform message is received.
- *Report* – SNMP v3 is needed to use Report messages. They allow an SNMP manager to determine what kind of problem was detected by the remote SNMP agent. Based on the detected error, the SNMP engine may try to send a corrected SNMP message. If that is not possible, it may pass an indication of the error to the application on whose behalf the failed SNMP request was issued. [RFC3412]

SNMP message transfer
The Simple Network Management Protocol is part of the Internet Protocol Suite as an application layer (layer 7) protocol of the OSI model.
SNMP uses the User Datagram Protocol (UDP) to transfer messages. It is necessary that UDP packets can make it from the agent to the manager for monitoring to be successful. This typically works by default on a local network but additional router configuration is needed to allow such packets to traverse wider networks.
SNMP agents receive UDP requests on port 161. Requests sent from an SNMP manager may be sent from any port. Usually, it's 161. Agents send traps via port 162. The SNMP manager also receives traps on port 162.

What are OIDs and MIBs?

OID

OID stands for Object Identifier. OIDs uniquely identify managed objects that are defined in MIB files. Example:

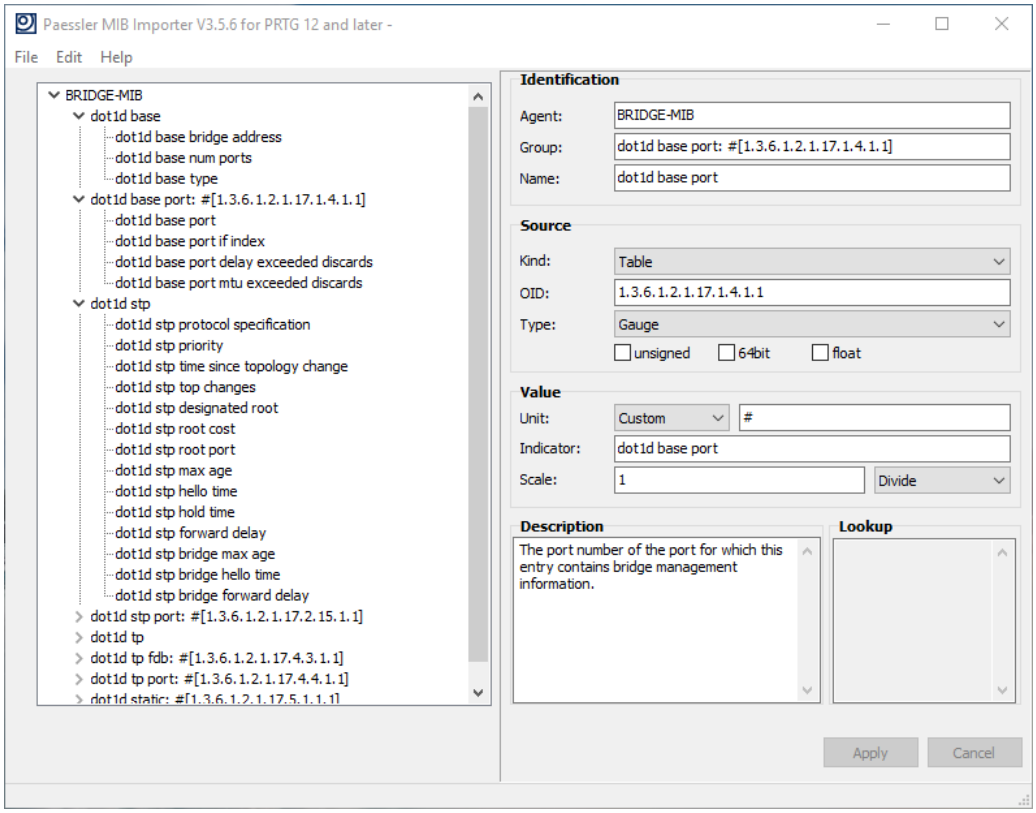
> On a printer, the typical objects to monitor are the different cartridge states and maybe the number of printed files. On a switch, the typical objects of interest are the incoming and outgoing traffic as well as the rate of package loss or the number of packets addressed to a broadcast address.
> The object (OID) hierarchy is generally depicted as a tree with different levels from the root to the single leaves. Each OID has an address that follows the levels of the OID tree.
> Sample:
> Here is a sample structure of an OID:
> Iso(1).org(3).dod(6).internet(1).private(4).transition(868).products(2).chassis(4)
> .card(1).slotCps(2)-cpsSlotSummary(1).cpsModuleTable(1).cpsModuleEntry(1)
> .cpsModuleModel(3).3562.3
>
> or just:
>
> 1.3.6.1.4.868.2.4.1.2.1.1.1.3.3562.3

Top level and general MIB object IDs are asssigned by different standard organizations like ISO. Vendors define OIDs for their own products in private branches of the OID tree.
There are two types of objects: scalar and tabular ones. Scalar objects define a single object instance whereas tabular objects define multiple related object instances grouped in MIB tables.

MIB

MIB stands for Management Information Base, and refers to an independent format for the definition of management information. In other words, MIBs contain OIDs in a well-defined way. In an MIB, every object gets its definition that defines its properties within the device to be managed. The objects are accessed using the SNMP protocol.

Why OIDs and MIBs are needed
Every piece of management information that can be obtained via SNMP – be it the memory usage of a server, the traffic on a switch, or the queued files on a printer – is individually addressed by its OID. This property is the reason why OIDs are needed. They help admins to identify and monitor the objects that they have in their network and thus make monitoring meaningful.
For the managing entity and a managed device in networks to communicate successfully, both of them need to know which OIDs are available.
This is the reason why MIBs exist and why system admins need them. Every object that has to be monitored on a device has to be provided by the device's MIB(s). Therefore, admins have to make sure that all necessary MIBs are stored on SNMP agent devices as well as on the managing entity system. An MIB file can be easily recognized via the extensions *.my* or *.mib*.



Device manufacturers normally deliver the necessary MIBs along with their SNMP-supporting products. Depending on the monitoring solution used, one might have to convert MIBs to product specific MIB versions.

For detailed information on public-known MIB, consult https://oidref.com/ .
For vendor-specific MIBs, please check out vendor's website and.or its documentation.
Using specifc tools like MIB browsers allow to check-out the MIB file itself which is most of the time self-explaining (for a good understander). Google for 'MIB browser" to find awhole list of them. Eg: https://www.ireasoning.com/mibbrowser.shtml

Why SNMP is so universal: SMI
There are various factors that make SNMP so universal. Structure of Management Information (SMI) provides a commonly understood and standardized structure for the data types and for the transfer of them. Please also consider this link to get to the bottom of why SMI is also inextricably tied to OIDs and MIBs. More about SMI here.

Value added by SNMP
Why would someone use SNMP? Well, as its name Simple Network *Management* Protocol says, it can be used for network *management*. In order to manage a network, admins need information about it. This is where SNMP has its greatest value. It gathers all the data from many devices and allows to put this data into context, which again allows to track issues, to make decisions based on real data, and to take control wherever necessary. That's what network management is all about. And that's why sysadmins will profit from using SNMP to monitor networks.
But there is more to it. A suitable monitoring tool will also help to get the most out of the data that admins receive thanks to SNMP and empower every network admin to monitor and manage their networks on time and proactively.

Being well organized
It can be an adventure to keep track of the vast quantity of network devices that modern networks comprise. Ideally, monitoring solutions support admins by offering a suitable way to structure and group devices and by presenting a clear overview that allows to go into detail whenever needed to ensure overall system health.

Alerting & notifying
Thanks to monitoring tools, sysadmins know exactly where to take action when a problem arises, and sometimes they even know before it occurs. Being informed on time is absolutely critical, so monitoring solutions should offer enough ways to notify the admin – like email, push notifications, SMS text messages (to be on the safe side when there’s no internet, for example), execution of a program, or even alerts on a smartwatch.

Reporting & statistics
Network monitoring will result in a lot of data in monitoring database. Let the numbers speak for themselves and use them to create reports and statistics. Analysis will not only provide insight into networks, but also help justify IT needs to the accounting or managing board. Visual dashboards will also help to be more successful.

Planning ahead
Data analysis will show the trends for individual networks and allow to plan ahead. The results: reliability, speed, and efficiency.

Comfortable setup
Setting up monitoring for large networks can be very tedious and time-consuming. Monitoring tools will automate and speed that up for the users and provide, for example, flexible network auto-discoveries or default templates for widely used network infrastructure products. And who would have thought? These technologies most probably use SNMP, too!

Some easy to use command line tools.

Before going into the ESP32 code, it is good to have some simple commandline tools available that allow you to get and set SNMP OID values.

For Linux, installing the package snmp eg: `sudo apt -y install snmp` gives you the tools `snmpget`, `snmpset`, `snmpgetnext`, `snmpwalk`

For Windows you find this nice set on https://ezfive.com/snmpsoft-tools including `snmpget`, `snmpset`, `snmpwalk` and `snmptragen`.

```
> snmpget

Usage:
  SnmpGet.exe [-q] -r:host [-p:port] [-t:timeout] [-v:version] [-c:community]
        [-ei:engine_id] [-sn:sec_name] [-ap:auth_proto] [-aw:auth_passwd]
        [-pp:priv_proto] [-pw:priv_passwd] [-ce:cont_engine] [-cn:cont_name]
        -o:var_oid

   -q               Quiet mode (suppress header; print variable value only)
   -r:host          Name or network address (IPv4/IPv6) of remote host.
   -p:port          SNMP port number on remote host. Default: 161
   -t:timeout       SNMP timeout in seconds (1-600). Default: 5
   -v:version       SNMP version. Supported version: 1, 2c or 3. Default: 1
   -c:community     SNMP community string for SNMP v1/v2c. Default: public
   -ei:engine_id    Engine ID. Format: hexadecimal string. (SNMPv3).
   -sn:sec_name     SNMP security name for SNMPv3.
   -ap:auth_proto   Authentication protocol. Supported: MD5, SHA (SNMPv3).
   -aw:auth_passwd  Authentication password (SNMPv3).
   -pp:priv_proto   Privacy protocol. Supported: DES, IDEA, AES128, AES192,
                    AES256, 3DES (SNMPv3).
   -pw:priv_passwd  Privacy password (SNMPv3).
   -cn:cont_name    Context name. (SNMPv3)
   -ce:cont_engine  Context engine. Format: hexadecimal string. (SNMPv3)
   -o:var_oid       Object ID (OID) of SNMP variable to GET.

> snmpset

Usage:
  SnmpSet.exe [-q] -r:host [-p:port] [-t:timeout] [-v:version] [-c:community]
        [-ei:engine_id] [-sn:sec_name] [-ap:auth_proto] [-aw:auth_passwd]
        [-pp:priv_proto] [-pw:priv_passwd] [-ce:cont_engine] [-cn:cont_name]
        -o:var_oid -val:value [-tp:type]

   -q               Quiet mode (suppress header)
   -r:host          Name or network address (IPv4/IPv6) of remote host.
   -p:port          SNMP port number on remote host. Default: 161
   -t:timeout       SNMP timeout in seconds (1-600). Default: 5
   -v:version       SNMP version. Supported version: 1, 2c or 3. Default: 1
   -c:community     SNMP community string for SNMP v1/v2c. Default: private
   -ei:engine_id    Engine ID. Format: hexadecimal string. (SNMPv3).
   -sn:sec_name     SNMP security name for SNMPv3.
   -ap:auth_proto   Authentication protocol. Supported: MD5, SHA (SNMPv3).
   -aw:auth_passwd  Authentication password (SNMPv3).
   -pp:priv_proto   Privacy protocol. Supported: DES, IDEA, AES128, AES192,
                    AES256, 3DES (SNMPv3).
   -pw:priv_passwd  Privacy password (SNMPv3).
   -cn:cont_name    Context name. (SNMPv3)
   -ce:cont_engine  Context engine. Format: hexadecimal string. (SNMPv3)
   -o:var_oid       Object ID (OID) of SNMP variable to SET.
   -val:value       Variable value to SET.
   -tp:type         Type of variable to SET. Supported: int,uint,str,hex,oid,ip.
Default: str
```

Arduino IDE and ESP32 code

Before starting to code SNMP, you need to load a SNMP library in you Arduino IDE.
Go to `https://github.com/0neblock/Arduino_SNMP` and download the ZIP file. Unzip, rename folder to Arduino_SNMP and copy it to the Arduino Library folder Eg:

```
C:\Users\<accountname>\Documents\Arduino\libraries
```

It's also important to read the `readme.md` to understand this library drawbacks.
Now that this is done, you can start coding for SNMP. Below you find my sample code.

The 3 major SNMP steps are

1. setup the vars you want to handled by SNMP
2. setup SNMP object
3. handle SNMP

As you will see, the SNMP handler only take pointers to vars and even for strings its a pointer to a pointer. Therefor some zextra manipulation is needed to get the memory setup for the strings and to point to them.

This SNMP library handles vars of the type integer, float (it claims but it returns an ingeter anyway :-( ), string, 32bit counter, 64bit counter and 32bit gauges

Note: the author of the library made a typo in the function and hence it is called `addGuageHandler` instead of `addGaugeHandler`.

The code below is straightforward and should be 'easy' to read for any programmer with a bit of experience and basic C-knowledge.

```
#include <Ticker.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#include <time.h>
#include <Arduino_SNMP.h>

#define hostname             "ESP32"
#define location             "Middenstraat-111"

//WiFi settings
#define wifiSSID             "Engrie-IoT"
#define wifiPassword         "13579iot24680"

// NTP Settings
#define ntpServer            "be.pool.ntp.org"
#define gmtOffset_sec        3600
#define daylightOffset_sec   3600

// easy wrapper for counting seconds instead of millisceonds
#define seconds()        ((int)(millis()/1000))

//--------------------------------------------------------------------------
// global variables
//--------------------------------------------------------------------------
// to hold MAC address
String wifiMACAddress = WiFi.macAddress();

// time related
unsigned long time_now;
char today[11];
// to hold date & time
char *dt;

//vars to hold SNMP
char *sysDescr;
int  sysUpTime;
char *sysContact;
char *sysName;
char *sysLocation;

char *pstrTime;
char *pstrMessage;
float fltNumber;
uint32_t u32Counter;
uint64_t u64Counter;
uint32_t u32Gauge;

//--------------------------------------------------------------------------
// objects
//--------------------------------------------------------------------------
// Initiate led blinker library
Ticker ticker;

WiFiUDP wifiUDP;
//Initialize SMMPAgent
SNMPAgent snmp = SNMPAgent("public");
```

```
//------------------------------------------------------------------------------
// Functions
//------------------------------------------------------------------------------

//.....................................................................
// Call-back functions
//.....................................................................
//** Call-back TICKER
//**************************************************************************
void tick()
{
  // get the current state of GPIO1 pin
  int state = digitalRead(LED_BUILTIN);
  // set pin to the opposite state
  digitalWrite(LED_BUILTIN, !state);
}
//**************************************************************************

//** Set-up WIFI
//**************************************************************************
void wifiConnect()
{
  WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE); // needed to allow
setting hostname
  WiFi.setHostname(hostname);
  WiFi.mode(WIFI_STA);
  WiFi.begin(wifiSSID, wifiPassword);

  Serial.print(F("WiFi attempting connection to ")); Serial.println(wifiSSID);

  uint8_t i = 0;
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print('.');

    // delay non-blocking
    time_now = millis();
    while(millis() < time_now + 500){}
    if ((++i % 16) == 0)
    {
      Serial.println(F(""));
      Serial.println(F("still trying to connect"));
    }

    if (i > 32)
    {
      Serial.println(F("Restarting..."));
      ESP.restart();
    }
  }

  Serial.println(F(""));
  Serial.println(F("WiFi connected"));
  Serial.print(F("  MAC Address: ")); Serial.println(wifiMACAddress);
  Serial.print(F("  IP Address : ")); Serial.println(WiFi.localIP());
  Serial.print(F("  Hostname   : ")); Serial.println(WiFi.getHostname());
}
//**************************************************************************
```

```
//** Set-up TIME
//*************************************************************************
void timeSetup()
{
  Serial.println(F("TIME retreiving"));

  //-------------------------------------------------------------------------
  // Get correct time
  //-------------------------------------------------------------------------
  // Init time
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  dt = LocalTime();
  strncpy(today, dt, 10);
  today[10] = NULL;

  Serial.print(F("  Today : ["));  Serial.print(today); Serial.println(F("]"));
  Serial.println(F("TIME retreived"));
}
//*************************************************************************

//** Get TIME
//*************************************************************************
char* LocalTime()
{
  struct tm now;

  if(!getLocalTime(&now))
  {
    Serial.println(F("Failed to obtain time"));
    return("");
  }

  static char dt[25] = {'\0'};
  snprintf(dt, sizeof(dt), "%04d-%02d-%02d %02d:%02d:%02d",  now.tm_year + 1900,
now.tm_mon + 1, now.tm_mday, now.tm_hour, now.tm_min, now.tm_sec );
  return(dt);
}
//*************************************************************************
```

```
//** Set-up SNMP
//************************************************************************
void snmpSetup()
{
  // init system vars
  sysDescr     = (char*)malloc(50);
  sysContact   = (char*)malloc(50);
  sysName      = (char*)malloc(50);
  sysLocation  = (char*)malloc(50);

  sysDescr     = strdup("SNMP Sample");
  sysUpTime    = ((int)(millis()/10));
  sysContact   = strdup("Marc Engrie");
  sysName      = strdup(hostname);
  sysLocation  = strdup(location);

  // init application vars
  pstrTime    = (char*)malloc(25);
  pstrMessage = (char*)malloc(256);
  memset(pstrTime,      0,  25);
  memset(pstrMessage,   0, 256);
  pstrMessage  = strdup("Hello");

  fltNumber = 0;
  u32Counter = 32;
  u64Counter = 64;

  randomSeed(analogRead(0));
  u32Gauge = random(1000);

  //Init snmp
  snmp.setUDP(&wifiUDP);
  snmp.begin();

  // Add default SNMP OID info: sysDescr, sysUpTime, sysContact, sysName, sysLocation
  //Add OID parameter sysDescr (read only)
  snmp.addStringHandler(".1.3.6.1.2.1.1.1.0", &sysDescr, false);
  //Add OID parameter sysUpTimer (read only)
  snmp.addIntegerHandler(".1.3.6.1.2.1.1.3.0", &sysUpTime, false);
  //Add OID parameter sysContact (read only)
  snmp.addStringHandler(".1.3.6.1.2.1.1.4.0", &sysContact, false);
  //Add OID parameter sysName (read only)
  snmp.addStringHandler(".1.3.6.1.2.1.1.5.0", &sysName, false);
  //Add OID parameter sysLocation (read only)
  snmp.addStringHandler(".1.3.6.1.2.1.1.6.0", &sysLocation, false);

  // Add application SNMP OID's
  //Add OID parameter time - string (read only)
  snmp.addStringHandler(".1.3.6.1.4.1.12345.0", &pstrTime, false);

  //Add OID parameter message - string (read/write)
  snmp.addStringHandler(".1.3.6.1.4.1.12345.1", &pstrMessage, true);

  //Add OID parameter fltNumber - float (read only)
  snmp.addFloatHandler(".1.3.6.1.4.1.12345.2", &fltNumber, false);

  //Add OID parameter u32Number - uint32_t (read only)
  snmp.addCounter32Handler(".1.3.6.1.4.1.12345.3", &u32Counter, false);

  //Add OID parameter u64Number - uint64_t (read only)
  snmp.addCounter64Handler(".1.3.6.1.4.1.12345.4", &u64Counter, false);

  //Add OID parameter u32Gauge -  uint32_t (read only)
  snmp.addGuageHandler(".1.3.6.1.4.1.12345.5", &u32Gauge, false);
}
//************************************************************************
```

```
//** Handle SNMP
//**************************************************************************
void snmpHandle()
{
  //do a SNMP loop
  snmp.loop();

  //if there was a write request, do it
  if(snmp.setOccurred)
  {
    //vars are updates with data recived, handle accordingly
    Serial.println(pstrMessage);
    //Reset flag
    snmp.resetSetOccurred();
  }
}
//**************************************************************************

//==========================================================================
// Setup
//==========================================================================
void setup()
{
  Serial.begin(115200);
  delay(1000);

  Serial.println(F("Starting up ..."));

  // Set led pin as output
  pinMode(LED_BUILTIN, OUTPUT);
  // Start ticker to indicate set-up mode
  ticker.attach(0.5, tick);

  //--------------------------------------------------------------------------
  // Startup WiFi
  //--------------------------------------------------------------------------
  wifiConnect();

  //--------------------------------------------------------------------------
  // Startup time
  //--------------------------------------------------------------------------
  timeSetup();

  //--------------------------------------------------------------------------
  // Startup SNMP
  //--------------------------------------------------------------------------
  snmpSetup();

  // Ending set-up mode, turn led off to save power
  ticker.detach();
  digitalWrite(LED_BUILTIN, LOW);

  Serial.println(F("Ready"));
}
//==========================================================================
```

```
//#############################################################################
// Main loop
//#############################################################################
void loop()
{
  // update Date-Time
  pstrTime  = LocalTime();
  // update sysUpTime
  sysUpTime = ((int)(millis()/10));

  fltNumber += 0.1;
  u32Counter += 32;
  u64Counter += 64;
  u32Gauge = random(1000);

  // handle SNMP
  snmpHandle();
  delay(1000);
}
```

Testing
Once you run the code on the ESP32, using the commandline tools, you can start testing.
Note: I used the Windows tools

```
> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.0

OID=.1.3.6.1.4.1.12345.0
Type=OctetString
Value=2021-04-06 18:15:41

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.1

OID=.1.3.6.1.4.1.12345.1
Type=OctetString
Value=Hello

> snmpset -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.1 -val:teststring

OK

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.1

OID=.1.3.6.1.4.1.12345.1
Type=OctetString
Value=teststring


> snmpset -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.1 -val:teststring2

OK

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.1

OID=.1.3.6.1.4.1.12345.1
Type=OctetString
Value=teststring2

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.2

OID=.1.3.6.1.4.1.12345.2
Type=Integer
Value=90

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.3

OID=.1.3.6.1.4.1.12345.3
Type=Counter32
Value=127

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.4

OID=.1.3.6.1.4.1.12345.4
Type=Counter64
Value=164

> snmpget -r:192.168.1.87 -c:public -o:1.3.6.1.4.1.12345.5

OID=.1.3.6.1.4.1.12345.5
Type=Gauge32
Value=201
```