SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

ArduAgent

An SNMPv1/SNMPv2c Agent for SAM3x and AVR Platforms

CENG447 - Spring 2016

Users Manual

Authors:

Adrian Del Grosso

Paul Christenson

Professor:

Dr. Randy Hoover

April 29, 2016

Contents

1.0 Introduction

This document outlines the implementation of an SNMPv1/v2c agent for use on ARM SAM3x and AVR8/16 platforms. Similar agents exist on the internet already, but all lack support for SAM3x devices (Arduino DUE).

2.0 SNMP Overview

The Simple Network Management Protocol (SNMP) is a Layer 7 network protocol originally defined in RFC 1228 by the Internet Engineering Task Force. It provides a highly structured, hardware agnostic, extensible interface between a programmer and data running on remote systems. In many respects, it is similar to the MODBUS protocol. Consider the following scenario: A data center contains many pieces of hardware, and it is crucial that the status of each device is known and monitored at all times. An ideal way to achieve this might be to have a central device that monitors all other pieces of hardware in the data center and can send alerts to staff when problems are detected. Is there a way to do this that is supported by all the devices in the data center? The answer is: yes! Generally, all modern consumer and enterprise devices with Ethernet capability support SNMP to some degree. It could be used in this case to aggregate up-time and performance data from all devices in the data center. This could, of course, remain true for any number of devices over any distance as long as the devices have a network connection.

2.1 SNMP Architecture

2.1.1 SNMP Basic Concepts

Consider a weather station composed of an Arduino, an Ethernet shield, and some sensors. Let's say you, as the hypothetical programmer of this weather station, want to log data from the station remotely, and it has an Ethernet-based network connection. An ideal way to monitor the weather data being produced by this weather station remotely might be SNMP. In the Arduino's running program, you as the programmer undoubtedly have a variable for storing the last measured temperature value the station monitored. By staring at the Arduino itself, you have no way of knowing what that variable contains at any given time. SNMP would allow you to "ask" the Arduino what that variable contains at any time. It could even allow you to change that variable's contents if you so desired. SNMP defines names for each part of this hypothetical system and process. The Arduino, or whatever device you wish to monitor, runs a piece of code called an "agent", which makes your program's variables available to users over Ethernet. The agent essentially acts as a driver between your running program and the Ethernet connection. The device that wants to "ask" the Arduino for some data is called a network monitoring station (NMS). Furthermore, the subset of variables you, as the programmer, choose to make available over SNMP is called a management information base (MIB). These definitions will be used throughout the remainder of this document.

2.1.2 Clients and Servers

SNMP uses a client-server architecture. The network monitoring station (NMS) is the "client". Devices running an SNMP agent are "servers". In SNMP, it is common that one client accesses a large number of servers to aggregate information across multiple devices. It is even possible to have multiple agents on the same server. These are called subagents.

2.1.3 Objects

Recall, the subset of variables that are monitored by an SNMP agent is called the MIB. In a MIB, assume, for example, you have an integer variable called "Temperature". If you want to know what the value of "Temperature" is over SNMP, you "ask" the agent for it using what's called an object identifier (OID). Each variable has a unique object identifier. OIDs are very specially formatted strings. They generally look like this: "1.3.6.1.2.1". Each number has a specific meaning. Essentially, however, each of these OIDs refers to a variable you can "ask" for from an SNMP agent. The agent might know what the OID means, or it might not. If it knows that the OID you "asked" for means Temperature, it will send a response containing the value of Temperature. if it does not know what the OID it received means, it will send an error response.

2.1.4 Authentication

In SNMPv1 and SNMPv2, simple authentication is used between agents and clients through "community names". These are essentially plain-text passwords that must be verified for an SNMP transaction to succeed. Typically, these community names are different depending on the operation that the client wishes to perform over SNMP. This will be discussed in greater detail in later sections.

2.2 SNMP Commands

2.2.1 GET

The GET command is how a client can ask an SNMP agent for data. It is the most basic and most important command. When an agent receives a GET command, it checks the OID in the GET request against its MIB to see if it can respond. It also verifies the community name in the GET request. If the community name is verified, and it has a variable in its MIB that corresponds to the OID in the GET request, the agent sends a GET response containing the data the client asked for.

2.2.2 SET

The SET command is how a client can alter variables remotely using SNMP. SET commands usually require a different community name than GET requests to succeed. Essentially, the process to set a variable over SNMP is the same as getting a variable. The only real difference is that the SET request also contains the desired new value for the variable. When an agent receives and processes a set request, it responds using a GET response to inform the client of the updated value in the variable it altered.

3.0 ArduAgent

3.1 Overview

ArduAgent implements a large subset of the commands in SNMPv1 and SNMPv2c. It supports all standard SNMP error codes for both standards and is very lightweight. It is written in C++. ArduAgent fully supports octet-stream responses (strings) as well as integer responses. It's easy to integrate ArduAgent into your projects! This section will outline the basics behind how ArduAgent works, and what its limitations are.

4.0 Integrating ArduAgent Into Your Programs

There are a few steps to integrating ArduAgent with your program. Each step will be explained in detail in this section. Please note: **An Ethernet Shield is required to run the example.**

These steps are:

1. Include the ArduAgent API
2. Define OIDs for each variable you wish to monitor using SNMP
3. Configure read/write permissions for the variables you wish to monitor

An example sketch can be downloaded from the project's Github page:
https://github.com/ad3154/ArduAgent. This section will make heavy reference to this example.

4.1 General Structure

The ArduAgent API example sketch contains three major functions: loop, setup, and pduReceived.

4.1.1 Setup

The "setup" function prepares the Arduino for operation. Most importantly, it prepares the Ethernet connection and begins DHCP configuration by default. This behavior can be overridden by replacing the line "Ethernet.begin(mac);" with "Ethernet.begin(mac, ip, dns, gateway, subnet);". It also enables serial with a connected host using a baud rate of 9600 for debugging and printing the IP address of the Arduino once DHCP has concluded.

4.1.2 Loop

This is the main body of the example program. This is where you, as the programmer, would put the main code for your program. In the example, loop() contains a brief amount of code to keep track of the system up-time, and poll for available SNMP requests in the Ethernet buffer. To get started programming quickly, simply add your code below these lines.

4.1.3 pduReceived

The pduReceived function is where you, as the programmer, can configure ArduAgent to monitor certain variables (OIDs), and how it should respond to requests. The main structure of this function is the large if/else structure in its center. This is where you'll tell the ArduAgent API how to monitor your program. This will be discussed in the sections titled, "Configure Permissions" and "OID Mapping".

4.2 Include ArduAgent

The first step to using ArduAgent is to download and include the library. Download the latest version of ArduAgent at https://github.com/ad3154/ArduAgent. Unzip the project, and place the folder named ”ArduAgent” into your C:/Users/(your username)/Documents/Arduino/Libraries folder. Launch the Arduino IDE. Then, just type ”#include <arduAgent.h>” at the top of your program.

4.3 Define OIDs

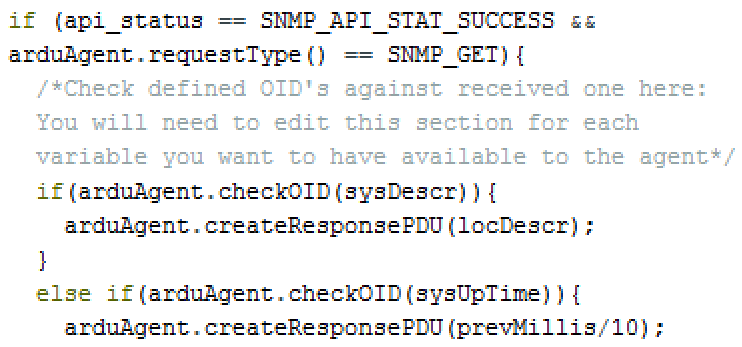
Each variable you wish to monitor using the ArduAgent api must have an associated OID. To do this, define a C style string containing the OID you wish to associate to a variable. Be aware, many OIDs are reserved for standards and enterprise groups. It's best to make sure the OID you are using is unique. Usually, this means registering your business or group with the Internet Assigned Numbers Authority. Once registered, you will receive a private OID space to use that is unique. This is normally within: iso.org.dod.internet.private.enterprise (1.3.6.1.4.1.x). Listing 1 shows an example declaration of an OID in an Arduino Sketch. You may simply edit this example to create your own OID. Be sure to add a trailing zero to the OID you declare as shown in Listing 1.

Listing 1: Declaring an OID

```
// .iso.org.dod.internet.mgmt.mib-2.hostresourcesMIB.hrUpTime
// (.1.3.6.1.2.1.25.1.1)
int hrUpTime[]       = {1,3,6,1,2,1,25,1,1,0};
```

4.4 Configure Agent

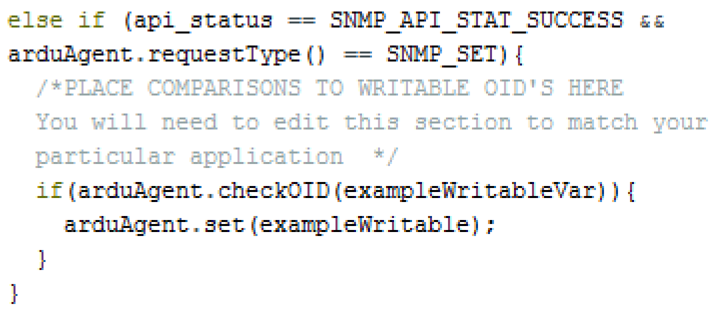
It is important to decide the read/write permissions you want for your monitored variable. If you wish for an OID to have read permissions, simply add your OID to the if/else structure shown in Figure 1. Simply add another ”else if” to the structure calling ”arduAgent.checkOID(yourOID)”, and if it matches, call ”arduAgent.createResponsePDU(yourVARIABLE)”, where yourOID is the C style string OID you defined for your monitored variable, and yourVARIABLE is the variable you're monitoring. Note: this must be either a C style string or an integer.



```
if (api_status == SNMP_API_STAT_SUCCESS &&
arduAgent.requestType() == SNMP_GET){
  /*Check defined OID's against received one here:
  You will need to edit this section for each
  variable you want to have available to the agent*/
  if(arduAgent.checkOID(sysDescr)){
    arduAgent.createResponsePDU(locDescr);
  }
  else if(arduAgent.checkOID(sysUpTime)){
    arduAgent.createResponsePDU(prevMillis/10);
```

Figure 1: Monitored Variables for GETs

If you wish for an OID to have write permissions, simply add it to the control structure shown below in Figure 2. Again, simply add an "else if" calling "arduAgent.checkOID(yourOID)", and if the OID matches, call "arduAgent.set(yourVARIABLE)" to execute the SET command, and return a get-response to the SNMP manager that made the request.



```
else if (api_status == SNMP_API_STAT_SUCCESS &&
arduAgent.requestType() == SNMP_SET){
  /*PLACE COMPARISONS TO WRITABLE OID'S HERE
  You will need to edit this section to match your
  particular application  */
  if(arduAgent.checkOID(exampleWritableVar)){
    arduAgent.set(exampleWritable);
  }
}
```

Figure 2: Monitored Variables for SETs

5.0 Troubleshooting

5.1 Errors

If you are receiving errors from SNMP transactions, common errors and their meanings are listed below.

1. SNMP_ERR_NO_SUCH_NAME

   This error means that the OID the SNMP manager has requested either is not in the agent's MIB, or the manager does not have permission to access the OID. If you are receiving this error, make sure the OID you are attempting to access is configure as shown in Section 4.4.

2. SNMP_ERR_TOO_BIG

   This error generally means the OID that has been received or processed is too large for ArduAgent to interpret. Shorten your OID definition from Section 4.3.

3. NULL2

   This error is synonymous with SNMP_ERR_NO_SUCH_NAME

4. SNMP_ERR_GEN_ERROR

   This error generally means the packet received by ArduAgent was invalid. Try using a different command or SNMP manager.

5. SNMP_ERR_AUTHORIZATION_ERROR

   If you are receiving this error, the OID requested from the SNMP manager is not writable and you are trying to SET the value. Reconfigure the agent as shown in Section 4.4.

6.0 Appendix

6.1 Example Sketch

```
#include <Ethernet.h>
#include <SPI.h>
#include <arduAgent.h>

static byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
//static byte ip[] = { 151, 159, 18, 201 };
//static byte gateway[] = { 151, 159, 18, 254 };
//static byte subnet[] = { 255, 255, 255, 0 };

// RFC1213-MIB OIDs
// .iso (.1)
// .iso.org (.1.3)
// .iso.org.dod (.1.3.6)
// .iso.org.dod.internet (.1.3.6.1)
// .iso.org.dod.internet.mgmt (.1.3.6.1.2)
// .iso.org.dod.internet.mgmt.mib-2 (.1.3.6.1.2.1)
// .iso.org.dod.internet.mgmt.mib-2.system (.1.3.6.1.2.1.1)
// .iso.org.dod.internet.mgmt.mib-2.system.sysDescr (.1.3.6.1.2.1.1.1)
int sysDescr[]     = {1,3,6,1,2,1,1,1,0};  // read-only  (DisplayString)
// .iso.org.dod.internet.mgmt.mib-2.system.sysObjectID (.1.3.6.1.2.1.1.2)
//static int sysObjectID[] = {1,3,6,1,2,1,1,2,0};  // read-only  (ObjectIdentifier)
// .iso.org.dod.internet.mgmt.mib-2.system.sysUpTime (.1.3.6.1.2.1.1.3)
int sysUpTime[]     = {1,3,6,1,2,1,1,3,0};  // read-only  (TimeTicks)
// .iso.org.dod.internet.mgmt.mib-2.system.sysContact (.1.3.6.1.2.1.1.4)
int sysContact[]    = {1,3,6,1,2,1,1,4,0};  // read-write (DisplayString)
// .iso.org.dod.internet.mgmt.mib-2.system.sysName (.1.3.6.1.2.1.1.5)
int sysName[]       = {1,3,6,1,2,1,1,5,0};  // read-write (DisplayString)
// .iso.org.dod.internet.mgmt.mib-2.system.sysLocation (.1.3.6.1.2.1.1.6)
int sysLocation[]   = {1,3,6,1,2,1,1,6,0};  // read-write (DisplayString)
// .iso.org.dod.internet.mgmt.mib-2.system.sysServices (.1.3.6.1.2.1.1.7)
int sysServices[]   = {1,3,6,1,2,1,1,7,0};  // read-only  (Integer)
// .iso.org.dod.internet.mgmt.mib-2.hostresourcesMIB.hrUpTime (.1.3.6.1.2.1.25.1.1)
int hrUpTime[]      = {1,3,6,1,2,1,25,1,1,0};
//      Example Writable OID    (.1.3.6.1.2.1.11.30)
int exampleWritableVar[]     = {1,3,6,1,2,1,11,30,0};
//
// Arduino defined OIDs
// .iso.org.dod.internet.private (.1.3.6.1.4)
// .iso.org.dod.internet.private.enterprises (.1.3.6.1.4.1)
// .iso.org.dod.internet.private.enterprises.arduino (.1.3.6.1.4.1.36582)
//
// RFC1213 local values
        static char locDescr[]              = "Description";// read-only (static)
        static uint32_t locUpTime           = 0;                    // read-only (static)
        static char locContact[20]          = "User";           // read-only (static)
        static char locName[20]             = "arduAgent";      // read-only (static)
        static char locLocation[20]         = "Somewhere␣USA";// read-only (static)
        static int32_t locServices          = 6;                        // read-only (static)
  // Example writable value
  int exampleWritable                   = 0;                    //Read-write

uint32_t prevMillis = millis();
SNMP_API_STAT_CODES api_status;
SNMP_ERR_CODES status;
```

```
void pduReceived()
{
        api_status = arduAgent.requestPdu();

        if (api_status == SNMP_API_STAT_SUCCESS &&
        arduAgent.requestType() == SNMP_GET){
                /*Check defined OID's against received one here:
                You will need to edit this section for each
                variable you want to have available to the agent*/
                if(arduAgent.checkOID(sysDescr)){
                        arduAgent.createResponsePDU(locDescr);
                }
                else if(arduAgent.checkOID(sysUpTime)){
                        arduAgent.createResponsePDU(prevMillis/10);
                        }else if(arduAgent.checkOID(hrUpTime)){
                        arduAgent.createResponsePDU(prevMillis/10);
                        }else if(arduAgent.checkOID(sysContact)){
                        arduAgent.createResponsePDU(locContact);
                        }else if(arduAgent.checkOID(sysLocation)){
                        arduAgent.createResponsePDU(locLocation);
                        }else if(arduAgent.checkOID(sysName)){
                        arduAgent.createResponsePDU(locName);
                        }else if(arduAgent.checkOID(exampleWritableVar)){
                        arduAgent.createResponsePDU(exampleWritable);
                        }else{
                        arduAgent.generateErrorPDU(SNMP_ERR_NO_SUCH_NAME);
                }

        }
        else if (api_status == SNMP_API_STAT_SUCCESS &&
        arduAgent.requestType() == SNMP_SET){
                /*PLACE COMPARISONS TO WRITABLE OID'S HERE
                You will need to edit this section to match your
                particular application  */
                if(arduAgent.checkOID(exampleWritableVar)){
                        arduAgent.set(exampleWritable);
                }
        }
        else if (api_status == SNMP_API_STAT_PACKET_INVALID){
                arduAgent.generateErrorPDU(SNMP_ERR_GEN_ERROR);
        }
}

void setup()
{
  Serial.begin(9600);

  Ethernet.begin(mac);

  Serial.print("My␣IP␣address:␣");
  for (byte thisByte = 0; thisByte < 4; thisByte++) {
    // print the value of each byte of the IP address:
    Serial.print(Ethernet.localIP()[thisByte], DEC);
    Serial.print(".");
  }
  Serial.println();



  api_status = arduAgent.begin();
  //
  //Serial.println("agent has begun");
  if ( api_status == SNMP_API_STAT_SUCCESS ) {
    arduAgent.onPduReceive(pduReceived);

    return;
  }


}


void loop() {
  // listen/handle for incoming SNMP requests
   arduAgent.listen();

  if ( millis() - prevMillis > 1000 ) {
    // increment previous milliseconds on Uptime counter
    prevMillis += 1000;

    // increment up-time counter
    locUpTime += 100;
  }
}
```

6.2 ArduAgent.h

```
/*
  arduAgent.h - An Arduino library for a lightweight SNMP Agent.
  Copyright (C) 2016 Adrian Del Grosso
  All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef arduAgent_h
#define arduAgent_h

#define SNMP_DEFAULT_PORT        161
#define SNMP_MIN_OID_LEN         2
#define SNMP_MAX_OID_LEN         64
#define SNMP_MAX_NAME_LEN        20
#define SNMP_MAX_VALUE_LEN       64
#define SNMP_MAX_PACKET_LEN      SNMP_MAX_VALUE_LEN + SNMP_MAX_OID_LEN + 25  //???
#define SNMP_MAX_SET_LEN 20 //Arbitrary

#include "Arduino.h"
#include "Udp.h"

extern "C" {
        // callback function
        typedef void (*onPduReceiveCallback)(void);
}

typedef union uint64_u {
        uint64_t uint64;
        byte data[8];
};

typedef union int32_u {
        int32_t int32;
        byte data[4];
};

typedef union uint32_u {
        uint32_t uint32;
        byte data[4];
};

typedef union int16_u {
        int16_t int16;
        byte data[2];
};

typedef enum SNMP_API_STAT_CODES {
        SNMP_API_STAT_SUCCESS = 0,
        SNMP_API_STAT_MALLOC_ERR = 1,
        SNMP_API_STAT_NAME_TOO_BIG = 2,
        SNMP_API_STAT_OID_TOO_BIG = 3,
        SNMP_API_STAT_VALUE_TOO_BIG = 4,
        SNMP_API_STAT_PACKET_INVALID = 5,
        SNMP_API_STAT_PACKET_TOO_BIG = 6,
        SNMP_API_STAT_NO_SUCH_NAME = 7,
};

typedef enum SNMP_ERR_CODES {
        SNMP_ERR_NO_ERROR                       = 0,
        SNMP_ERR_TOO_BIG                        = 1,
        SNMP_ERR_NO_SUCH_NAME           = 2,
        SNMP_ERR_BAD_VALUE                      = 3,
        SNMP_ERR_READ_ONLY                      = 4,
        SNMP_ERR_GEN_ERROR                      = 5,

        SNMP_ERR_NO_ACCESS                              = 6,
        SNMP_ERR_WRONG_TYPE                     = 7,
        SNMP_ERR_WRONG_LENGTH                   = 8,
        SNMP_ERR_WRONG_ENCODING                 = 9,
        SNMP_ERR_WRONG_VALUE                    = 10,
        SNMP_ERR_NO_CREATION                    = 11,
        SNMP_ERR_INCONSISTANT_VALUE     = 12,
        SNMP_ERR_RESOURCE_UNAVAILABLE   = 13,
        SNMP_ERR_COMMIT_FAILED                  = 14,
        SNMP_ERR_UNDO_FAILED                    = 15,
        SNMP_ERR_AUTHORIZATION_ERROR    = 16,
        SNMP_ERR_NOT_WRITABLE                   = 17,
        SNMP_ERR_INCONSISTEN_NAME               = 18
};

typedef enum SNMP_REQUEST_TYPES {
        SNMP_GET=0xa0,
        SNMP_SET=0xa3
```

```
};

class arduAgentClass {
public:
        // Agent functions
        SNMP_API_STAT_CODES begin();
        SNMP_API_STAT_CODES begin(char *getCommName, char *setCommName, uint16_t port);
        void listen(void);
        SNMP_API_STAT_CODES requestPdu();
        SNMP_API_STAT_CODES responsePdu();
        void onPduReceive(onPduReceiveCallback pduReceived);
        void createResponsePDU(int respondValue);
        void createResponsePDU(char respondValue[]);
        SNMP_API_STAT_CODES set(int & reqValue);

        // Helper functions
        bool checkOID( const int inputoid[]);
        void getOID(byte input[]);
        int getOIDlength(void);
        SNMP_API_STAT_CODES send_response(void);
        void print_packet(void);
        void generateErrorPDU(SNMP_ERR_CODES CODE);
        SNMP_ERR_CODES authenticateGetCommunity(void);
        SNMP_ERR_CODES authenticateSetCommunity(void);
        SNMP_ERR_CODES generalAuthenticator(void);
        SNMP_REQUEST_TYPES requestType(void);


private:
        byte _packet[SNMP_MAX_PACKET_LEN];
        uint16_t _packetSize;
        uint16_t _packetPos;
        uint8_t _dstIp[4];
        char *_getCommName;
        size_t _getSize;
        char *_setCommName;
        size_t _setSize;
        onPduReceiveCallback _callback;

        //New PDU structure
        byte ans1Header;
        byte pdu_length;
        byte version[4];
        byte lengthCommunityName;
        char communityName[20];
        byte request[2];
        byte requestID[10];
        short int requestIDlength = 1;
        byte errorStatusCode[3];
        byte snmpIndex[3];
        byte varbindList[2];
        byte varbind[2];
        byte objectID;
        byte oidLength;
        short int oid[64] = {0x00};
        int setValueInt =0;
        char setValueChar[SNMP_MAX_SET_LEN] ={0}; //Size arbitrary
        unsigned short int setLength = 0;
        byte nulValue[2]={0x05,0x00};
};
```

```
extern arduAgentClass arduAgent;

#endif
```

6.3 ArduAgent.cpp

```
/*
  arduAgent.cpp - An Arduino library for a lightweight SNMP Agent.
  Copyright (C) 2016 Adrian Del Grosso
  All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include "arduAgent.h"
#include "EthernetUdp.h"

// Create UDP object
EthernetUDP Udp;

/**************************************************************************//**
 * Function: begin(without parameters)
 *
 * Description:
 * This is the default setup for the arduAgent Class. It makes the agent
 * use the default values of "public" and "private" for community names
 * and port 161 for communication. If the user wants to use different
 * values, he/she should call the other version of this function that
 * takes 3 parameters.
 *
 *
 * Parameters:
 * None
 *
 * Returns:
 *  SNMP_API_STAT_CODES SNMP_API_STAT_SUCCESS - Agent started
 *
 ******************************************************************************/
SNMP_API_STAT_CODES arduAgentClass::begin(){
        // set community names
        _getCommName = "public";
        _setCommName = "private";
        //
        // set community name set/get sizes
        _setSize = strlen(_setCommName);
        _getSize = strlen(_getCommName);
        //
        // init UDP socket
        Udp.begin(SNMP_DEFAULT_PORT);
        //
        return SNMP_API_STAT_SUCCESS;
}

/***************************************************************************//**
 * Function: listen
 *
 * Description:
 * This is the function that runs every so often within the user's main program
 * to check to see if a PDU is available. If a packet is available, we jump
 * to our requestPdu function.
 *
 *
 * Parameters:
 * None
 *
 * Returns:
 *  None
 *
 ******************************************************************************/
void arduAgentClass::listen(void){
        // if bytes are available in receive buffer
        // and pointer to a function (delegate function)
        // isn't null, trigger the function

        /*If the callback variable is set,
        we actually go to the memory location of the function
        pduReceived and begin to run there. Its like a super
        ghetto goto.*/
        Udp.parsePacket();
        if ( Udp.available() && _callback != NULL ) (*_callback)();
}

/***************************************************************************//**
 * Function: begin(with parameters)
 *
 * Description:
 * This is the non-default setup for the arduAgent Class. It allows users
 * to modify the community names and port number the agent uses.
 *
 *
 * Parameters:
 * char *getCommName - The C string for the GET community
 * char *setCommName - The C string for the SET community
 * uint16_t port          - Port number up to 65535
 *
 * Returns:
 *  SNMP_API_STAT_CODES SNMP_API_STAT_SUCCESS - Agent started
 *  SNMP_API_STAT_CODES SNMP_API_STAT_NAME_TOO_BIG - community name exceeds
 *              the maximum allowed in arduAgent.h
 *
 ******************************************************************************/
SNMP_API_STAT_CODES arduAgentClass::begin(char *getCommName, char *setCommName, uint16_t port){
        /* THIS FUNCTION NEEDS TO BE REDONE*/
        // set community name set/get sizes
        _setSize = strlen(setCommName);
        _getSize = strlen(getCommName);
        //
        // validate get/set community name sizes
        if ( _setSize > SNMP_MAX_NAME_LEN + 1 || _getSize > SNMP_MAX_NAME_LEN + 1 ) {
                return SNMP_API_STAT_NAME_TOO_BIG;
        }
        //
        // set community names
        _getCommName = getCommName;
        _setCommName = setCommName;
        //
        // validate session port number
        if ( port == NULL || port == 0 ) port = SNMP_DEFAULT_PORT;
        //
        // init UDP socket
        Udp.begin(port);

        return SNMP_API_STAT_SUCCESS;
}

/**************************************************************************//**
 * Function: onPduReceive
 *
 * Description:
 * This function is how we jump back to the user's program after parsing
 * a received SNMP packet. We jump to the user's handler function
 * called onPduReceive. Consequently, their function MUST always
 * be named onPduReceive.
 *
 *
 * Parameters:
 * onPduReceiveCallback pduReceived
 *
 * Returns:
 *  None
 *
 ******************************************************************************/
void arduAgentClass::onPduReceive(onPduReceiveCallback pduReceived){
        _callback = pduReceived;
}

/**************************************************************************//**
 * Function: requestPDU
 *
 * Description:
 * This function does the real heavy lifting. It parses the entire SNMP
 * portion of a received packet into private data. It also performs several
 * error checks and authentication checks on the received packet.
 *
 *
 * Parameters:
 * None
 *
 * Returns:
 *  SNMP_API_STAT_CODES SNMP_API_STAT_SUCCESS - No errors - Packet parsed
 *  SNMP_API_STAT_CODES SNMP_ERR_TOO_BIG - Packet exceeds maximum packet size
 *            defined in arduAgent.h
 *      SNMP_API_STAT_CODES SNMP_API_STAT_PACKET_INVALID - Not an SNMP packet or
 *            client not authenticated
 *
 ******************************************************************************/
SNMP_API_STAT_CODES arduAgentClass::requestPdu(){
        SNMP_ERR_CODES authenticated = SNMP_ERR_NO_ERROR;
        unsigned short int errorStatusCodeBaseAddress;
        _packetSize = Udp.available();
        // reset packet array
```

```
for (int rstCounter=0; rstCounter < SNMP_MAX_PACKET_LEN; rstCounter++)
{
_packet[rstCounter] = 0;
}

//Validate Packet Size
if ( _packetSize != 0 && _packetSize > SNMP_MAX_PACKET_LEN ) {
        arduAgent.generateErrorPDU(SNMP_ERR_TOO_BIG);
        return SNMP_API_STAT_PACKET_TOO_BIG;
}

//Get the actual packet and store it for use
Udp.read(_packet, _packetSize);



//Check to see if the packet is a SNMPv1 packet
//This value should always be 0x30
if ( _packet[0] != 0x30)
{
        return SNMP_API_STAT_PACKET_INVALID;
}

/* We have a pdu structure that was passed in.
We'll now populate that structure from
data we received in the buffer*/

ans1Header = _packet[0];
pdu_length = _packet[1];
for(int i = 2; i<6; i++)
{
        version[i-2] = _packet[i];
}
lengthCommunityName = _packet[6];
for (int i = 0; i < (int) lengthCommunityName; i++)
{
        communityName[i] = _packet[7+i];
}
request[0] = _packet[7+lengthCommunityName];
request[1] = _packet[7+lengthCommunityName+1];
requestID[0] = _packet[7+lengthCommunityName+2];
requestID[1] = _packet[7+lengthCommunityName+3];
requestID[2] = _packet[7+lengthCommunityName+4];
if(requestID[1] > 1){
        requestIDlength=requestID[1];
        for(int i = 2; i<=requestIDlength; i++){
                requestID[i]=_packet[7+lengthCommunityName+5+i-2];
        }
}
errorStatusCodeBaseAddress = 7+lengthCommunityName+4+requestIDlength;
errorStatusCode[0] = _packet[errorStatusCodeBaseAddress++];
errorStatusCode[1] = _packet[errorStatusCodeBaseAddress++];
errorStatusCode[2] = _packet[errorStatusCodeBaseAddress++];
snmpIndex[0] = _packet[errorStatusCodeBaseAddress++];
snmpIndex[1] = _packet[errorStatusCodeBaseAddress++];
snmpIndex[2] = _packet[errorStatusCodeBaseAddress++];
varbindList[0] = _packet[errorStatusCodeBaseAddress++];
varbindList[1] = _packet[errorStatusCodeBaseAddress++];
varbind[0] = _packet[errorStatusCodeBaseAddress++];
varbind[1] = _packet[errorStatusCodeBaseAddress++];
objectID = _packet[errorStatusCodeBaseAddress++];
```

```
    oidLength = _packet[errorStatusCodeBaseAddress];
    for(int i=0; i < (int) oidLength; i++)
    {
        oid[i] = _packet[7+lengthCommunityName+16+i+requestIDlength];
    }
    if (arduAgent.requestType() == SNMP_SET)
    {
        // Read in the value the client wants to set
        //Serial.println("Set Command Detected");
        if (_packet[7+lengthCommunityName+16+(int) oidLength+requestIDlength]==0x02){
            setValueInt = _packet[9+lengthCommunityName+16+(int) oidLength+requestIDlength];
        }
        else if (_packet[7+lengthCommunityName+16+(int) oidLength+requestIDlength]==0x04){
            int len=0;
            len = _packet[8+lengthCommunityName+16+(int) oidLength+requestIDlength];
            if (len > SNMP_MAX_SET_LEN){
                return SNMP_API_STAT_PACKET_INVALID;
            }
            for(int i=0; i < len; i++)
            {
                setValueChar[i] = _packet[9+lengthCommunityName+16+(int) oidLength+requestIDlength+i];
            }
        }
    }
    authenticated = generalAuthenticator();
    if(authenticated == SNMP_ERR_NO_ERROR)
    {
        return SNMP_API_STAT_SUCCESS;
    }
        arduAgent.generateErrorPDU(authenticated);
        return SNMP_API_STAT_PACKET_INVALID;
}

/************************************************************************//**
 * Function: createResponsePDU (Integer)
 *
 * Description:
 * This function constructs an SNMP response packet specifically for the
 * SNMP data type "integer" and takes an integer as a parameter.
 * In other words, the int passed into this function will
 * be transmitted to the client in a GET response.
 *
 *
 * Parameters:
 * int respondValue - The integer the user wants to send to the client.
 *
 * Returns:
 *  None
 *
 ******************************************************************************/
    void arduAgentClass::createResponsePDU(int respondValue){
    int lsb = (respondValue >> (8*0)) & 0xff;
    int slsb = (respondValue >> (8*1)) & 0xff;
    int smsb = (respondValue >> (8*2)) & 0xff;
    int msb = (respondValue >> (8*3)) & 0xff;
    unsigned int baseResponseAddress = 7+lengthCommunityName+16+oidLength+requestIDlength;
    unsigned int total_len = 8+lengthCommunityName+16+oidLength+requestIDlength+5;
    _packet[7+lengthCommunityName] = 0xa2;  //Response
    _packet[baseResponseAddress] = 0x02;    //Integer
    _packet[baseResponseAddress+1] = 0x04;  //Length
    _packet[baseResponseAddress+2] = (uint8_t) msb;
```

```
        _packet[baseResponseAddress+3] = (uint8_t) smsb;
        _packet[baseResponseAddress+4] = (uint8_t) slsb;
        _packet[baseResponseAddress+5] = (uint8_t) lsb;
        _packet[1] = total_len -2;       //Null bytes not included in response
        //Recalculate packet lengths:
        _packet[7+lengthCommunityName+1] = _packet[7+lengthCommunityName+1]+4;
        _packet[7+lengthCommunityName+11+requestIDlength] = 10+oidLength;
        _packet[7+lengthCommunityName+13+requestIDlength] = 8+oidLength;
        arduAgent.send_response();      //Transmit the get response
}

/**************************************************************************//**
 * Function: createResponsePDU (C string)
 *
 * Description:
 * This function constructs an SNMP response packet specifically for the
 * SNMP data type "octet string" and takes a C string as a parameter.
 * In other words, the C string passed into this function will
 * be transmitted to the client in a GET response.
 *
 *
 * Parameters:
 * char respondValue[] - The C string the user wants to send to the client.
 *
 * Returns:
 *  None
 *
 *****************************************************************************/
void arduAgentClass::createResponsePDU(char respondValue[]){
                unsigned short int baseResponseAddress = 7+lengthCommunityName+16+oidLength+requestIDlength;
                unsigned short int stringLength = strlen(respondValue);
                unsigned short int total_len = 8+lengthCommunityName+16+oidLength+requestIDlength+1;
                _packet[7+lengthCommunityName] = 0xa2;  //Response code
                _packet[baseResponseAddress] = 0x04;    //Octet Stream Format
                _packet[baseResponseAddress+1] = stringLength;
                for (int i=0; i<stringLength;i++)
                {
                        //Build response
                        total_len++;
                        _packet[baseResponseAddress+1+(i+1)] = respondValue[i];
                        _packet[7+lengthCommunityName+1] = _packet[7+lengthCommunityName+1]+1;
                }
                //Recalculate packet lengths
                _packet[1] = total_len -2;
                _packet[7+lengthCommunityName+11+requestIDlength] = 6+oidLength+stringLength;
                _packet[7+lengthCommunityName+13+requestIDlength] = 4+oidLength+stringLength;
                arduAgent.send_response();
}

/**************************************************************************//**
 * Function: generateErrorPDU
 *
 * Description:
 * This function constructs an SNMP response packet based on the error code
 * that is passed in. It also sends the resulting packet to the client.
 *
 *
 * Parameters:
 * SNMP_ERR_CODES CODE - The error code for which a response should be
 *      generated.
 *
 * Returns:
 *  None
 *
 ****************************************************************************/
void arduAgentClass::generateErrorPDU(SNMP_ERR_CODES CODE){
        unsigned short int errorCodeLocation = 7+lengthCommunityName+6+requestIDlength;
        if (CODE==SNMP_ERR_TOO_BIG)
        {
                _packet[errorCodeLocation] = 0x01;
        }
        else if(CODE==SNMP_ERR_NO_SUCH_NAME)
        {
                _packet[errorCodeLocation] = 0x02;
        }
        else if (CODE==SNMP_ERR_BAD_VALUE)
        {
                _packet[errorCodeLocation] = 0x03;
        }
        else if (CODE==SNMP_ERR_READ_ONLY)
        {
                _packet[errorCodeLocation] = 0x04;
        }
        else if (CODE==SNMP_ERR_GEN_ERROR)
        {
                _packet[errorCodeLocation] = 0x05;
        }
        else if (CODE==SNMP_ERR_AUTHORIZATION_ERROR)
        {
                _packet[errorCodeLocation] = 0x10;
        }
        _packet[7+lengthCommunityName] = 0xa2;
        arduAgent.send_response();
}

/***************************************************************************//**
 * Function: getOID
 *
 * Description:
 * This function returns the OID stored in private data.
 *
 * Parameters:
 * byte input[] - A string into which the OID will be copied
 *
 * Returns:
 * None
 *
 ****************************************************************************/
void arduAgentClass::getOID(byte input[]){
        for (int i = 0; i < (int)oidLength; i++)
        {
                input[i] = oid[i];
        }
}

/***************************************************************************//**
 * Function: getOIDlength
 *
 * Description:
 * This function returns the length of the OID that was last received.
 * (The one stored in private data)
 *
 * Parameters:
 * None
 *
 * Returns:
 * int oidLength - The length of the OID in private data
 *
 ****************************************************************************/
int arduAgentClass::getOIDlength(void){
        return oidLength;
}


/****************************************************************************//**
 * Function: checkOID
 *
 * Description:
 * This function checks the OID in the received packet against the
 * OID that's passed in as a null terminated string. This is used
 * in the user's program to send the correct response based on the
 * OID's they have defined.
 *
 * Parameters:
 * None
 *
 * Returns:
 * false - OID didn't match
 * true - OID matched
 *
 ****************************************************************************/
bool arduAgentClass::checkOID(const int inputoid[]){
        for(int i = 2; i < oidLength; i++)
        {
                if(inputoid[i] != oid[i-1])
                {
                        return false;
                }
        }
        return true;
}

/****************************************************************************//**
 * Function: send_response
 *
 * Description:
 * This function transmits whatever is in _packet.
 *
 * Parameters:
 * None
 *
 * Returns:
 * SNMP_API_CODES SNMP_API_STAT_SUCCESS - No error (sent)
 * SNMP_API_CODES SNMP_API_STAT_PACKET_INVALID - bad packet (not sent)
 *
 ****************************************************************************/
SNMP_API_STAT_CODES arduAgentClass::send_response(void){
        if(!Udp.beginPacket(Udp.remoteIP(), Udp.remotePort()))
        {
                return SNMP_API_STAT_PACKET_INVALID;
        }
        Udp.write(_packet, _packet[1]+2);
        Udp.endPacket();
        return SNMP_API_STAT_SUCCESS;
}

/*************************************************************************//**
 * Function: print_packet
 *
 * Description:
 * This function is for debugging. It prints the received packet to a
 * serial port.
 *
 * Parameters:
 * None
 *
 * Returns:
 * None
 *
 *****************************************************************************/
void arduAgentClass::print_packet(void){
            for(int i =0; i < 50; i++)
        {
          Serial.print( _packet[i], HEX );
          Serial.print("␣");
        }

}

/*************************************************************************//**
 * Function: generalAuthenticator
 *
 * Description:
 * This function verifies the appropriate community name by calling
 * authenticateGetCommunity or authenticateSetCommunity.
 *
 * Parameters:
 * None
 *
 * Returns:
 * SNMP_ERR_CODES authd - The error code from authentication functions
 *
 *****************************************************************************/
SNMP_ERR_CODES arduAgentClass::generalAuthenticator(void){
        SNMP_ERR_CODES authd = SNMP_ERR_NO_ERROR;
        if (request[0] == 0xa0)
        {
                // Request was a GET request - Call authenticator
                authd = authenticateGetCommunity();
        }
        else //Result was not a GET request. SET is the only other implemented type
        authd = authenticateSetCommunity();//Call Authenticator

        return authd;
}

/*************************************************************************//**
 * Function: authenticateGetCommunity
 *
 * Description:
 * This function verifies that the GET community name in the received packet
 * matches the specified community names passed into begin(), or the
 * defaults if no parameters were passed to begin().
 *
 * Parameters:
 * None
 *
 * Returns:
 * SNMP_ERR_CODES SNMP_ERR_NO_ERROR if GET community matches
 * SNMP_ERR_CODES SNMP_ERR_AUTHORIZATION_ERROR if SNMPv2c  and
 *              GET community doesn't match
 * SNMP_ERR_CODES SNMP_ERR_NO_SUCH_NAME if SNMPv1 and GET community is wrong
 *
 *****************************************************************************/
SNMP_ERR_CODES arduAgentClass::authenticateGetCommunity(void){

        for (unsigned short int i=0;i<lengthCommunityName;i++)
        {
                if (communityName[i]!=_getCommName[i])
                {
                        // If SNMPv2c Request
                        if(_packet[4]==1)
                        {
                        return SNMP_ERR_AUTHORIZATION_ERROR;
                        }
                        // Otherwise, return SNMPv1 Error
                        else
                        return SNMP_ERR_NO_SUCH_NAME;
                }
        }
        return SNMP_ERR_NO_ERROR;
}

/****************************************************************************//**
 * Function: authenticateSetCommunity
 *
 * Description:
 * This function verifies that the SET community name in the received packet
 * matches the specified community names passed into begin(), or the
 * defaults if no parameters were passed to begin().
 *
 * Parameters:
 * None
 *
 * Returns:
 * SNMP_ERR_CODES SNMP_ERR_NO_ERROR if SET community matches
 * SNMP_ERR_CODES SNMP_ERR_AUTHORIZATION_ERROR if SNMPv2c  and
 *              SET community doesn't match
 * SNMP_ERR_CODES SNMP_ERR_NO_SUCH_NAME if SNMPv1 and SET community is wrong
 *
 *****************************************************************************/
SNMP_ERR_CODES arduAgentClass::authenticateSetCommunity(void){

        for (unsigned short int i=0;i<lengthCommunityName;i++)
        {
                if (communityName[i]!=_setCommName[i])
                {
                        // If SNMPv2c Request
                        if(_packet[4]==1)
                        {
                                return SNMP_ERR_AUTHORIZATION_ERROR;
                        }
                        // Otherwise, return SNMPv1 Error
                        else
                        {
                                return SNMP_ERR_NO_SUCH_NAME;
                        }
                }
        }
        return SNMP_ERR_NO_ERROR;
}

/***************************************************************************//**
 * Function: requestType
 *
 * Description:
 * This function returns the SNMP request code in the received PDU
 *
 * Parameters:
 * None
 *
 * Returns:
 * SNMP_REQUEST TYPES SNMP_GET (0xa0) request was a GET
 * SNMP_REQUEST TYPES SNMP_SET (0xa3) request was a SET
 ******************************************************************************/
SNMP_REQUEST_TYPES arduAgentClass::requestType(void){
        if (request[0] == 0xa3){
                //Request was a SET request
                return SNMP_SET;
        }
        else return SNMP_GET;
}

/***************************************************************************//**
 * Function: set (Integer)
 *
 * Description:
 * This function sets the variable passed into it using the integer
 * received over the network.
 *
 * Parameters:
 * int & reqValue - The user's program variable that they want set
 *
 * Returns:
 * SNMP_API_STAT_CODES SNMP_API_STAT_SUCCESS - No error
 * SNMP_API_STAT_CODES SNMP_API_STAT_PACKET_INVALID - Data type incorrect
 ******************************************************************************/
SNMP_API_STAT_CODES arduAgentClass::set(int & reqValue){

        if (_packet[7+lengthCommunityName+16+(int) oidLength+requestIDlength]==0x02)
        {
        reqValue = setValueInt;
        createResponsePDU(reqValue);
        return SNMP_API_STAT_SUCCESS;
        }
        else return SNMP_API_STAT_PACKET_INVALID;
}


// Create one global object
arduAgentClass arduAgent;
```