ESC-465: Developing a SNMP Agent on 8-bit Systems

By
Nilesh Rajbharti
Microchip Technology Inc.
Nilesh.Rajbharti@microchip.com

Table of Contents

Introduction

As the number of networked embedded systems increases, system designers must also provide the capability to monitor and control many devices, preferably in some automated fashion. Typically, this would require significant design and development decisions. But thanks to the Simple Network Management Protocol (SNMP), system designers can integrate a flexible ready-made solution. SNMP is a widely used management protocol for enterprise-computing networks. As we go through SNMP in more detail, it will become apparent that embedded systems can adopt the same protocol and reap the benefits of widely available SNMP tools. If you are an 8-bit system designer, you may quickly find that there are not many solutions available for 8-bit systems. There is plethora a of 16-bit and 32-bit SNMP Agents available, but there are very few that target 8-bit systems. Developing a SNMP Agent on 8-bit systems poses a unique challenge to system designers. A limited set of program and data memory requires creative thinking on the designer's part. One must make sacrifices in terms of protocol compliance and still keep it reasonably practical. This paper explains SNMP protocol along with its importance to embedded systems. The paper also discusses important SNMP Agent development issues one must consider when selecting or developing an SNMP Agent for an 8-bit system. This paper is expected to serve as a high-level guide for potential SNMP Agent application developers and system designers.

Assumption

The reader is expected to possess basic knowledge of TCP/IP protocols. A working knowledge of typical issues involved in remote device management is useful but not required.

Why SNMP?

When you have one or more devices connected to a system, it is imperative that the system provides some type of device monitoring and management functionality. As shown in Figure 1 Local Device Management, when devices are located in the physical proximity of one another, monitoring and management do not become a critical issue.

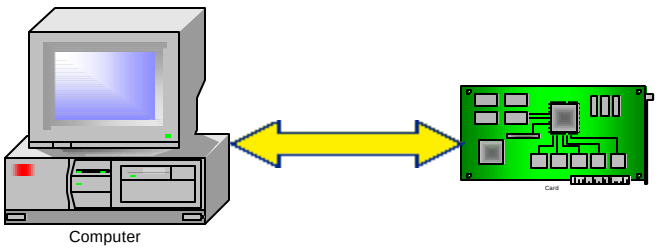

Figure 1 Local Device Management

However, as more and more devices are placed on a network that may span across the world, device management assumes an entirely new meaning. These devices may no longer be in the same physical proximity, and management software must be developed to provide the necessary status and control information. Remote devices must also execute a special management firmware module. System designers must decide to either

develop proprietary software or utilize an off-the-shelf solution. Some systems can afford to benefit from proprietary software, but the majority of systems use off-the-shelf, third party devices. In those systems, use of proprietary software means that the system console must keep separate management software for each manufacture specific device on the network.

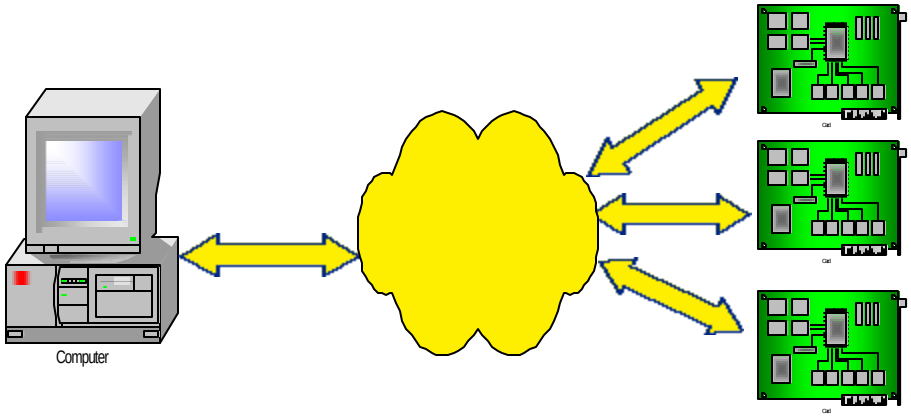

Figure 2 Network Devices Proliferation

SNMP was originally designed to solve this problem for the enterprise-computing industry. If used strategically, 8-bit embedded systems can benefit from the same protocol. One of the main advantages of using SNMP protocol is the availability of multiple third party PC-based monitoring software programs. As discussed in the next sections, SNMP is a very flexible and powerful protocol to transfer device management information.

What is SNMP?

The SNMP is a network-management protocol. It is primarily used to manage computer network devices such as file servers, print servers, printers, Ethernet hubs and other network-based systems. In enterprise-networks, SNMP is used to provide network statistical information such as the number of packets received, transmitted, dropped, etc. In addition, some devices may provide very specific information applicable to that device only.

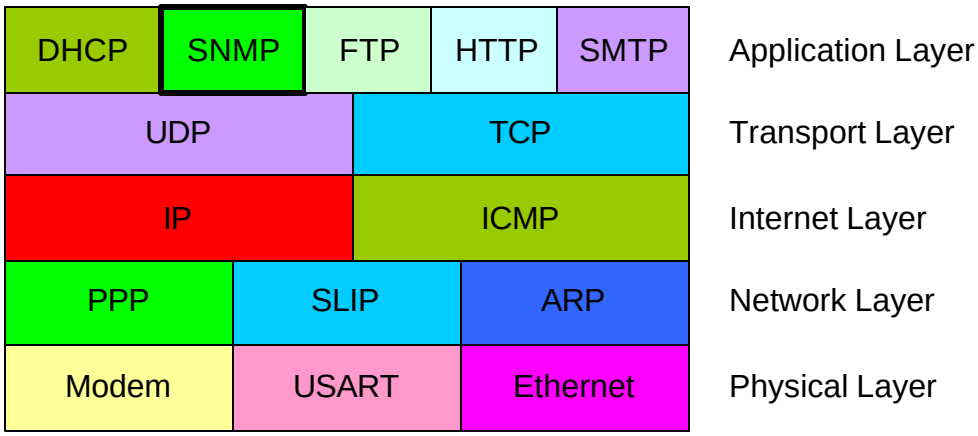

Figure 3 TCP/IP Stack

Figure 3 TCP/IP Stack, depicts a SNMP and resides at the application layer. SNMP is described in RFC 1157 in conjunction with other supporting RFC documents. The majority of the SNMP implementation utilizes a UDP (User Datagram Protocol) as its main transport layer. However, you may also use TCP (Transmission Control Protocol). UDP provides relatively small and simple transport services at the expense of reduced reliability. TCP, on the other hand, provides a more reliable transport at an increased program memory requirement. When employing a UDP, most of the SNMP management applications perform extra timeout and error checking to ensure a reliable management connection. As a result, the choice of using UDP or TCP as a transport service really depends on the amount of resources available. For 8-bit embedded systems with limited program and data memory, UDP is the choice of transport.

Protocols such as HTTP (Hyper Text Transfer Protocol) and SMTP (Simple Mail Transfer Protocol) are considered human-to-machine protocols. These protocols are primarily designed for humans as one end of the communication link. These protocols transfer various types of data, including graphics and video, which are meant for humans only. It would be difficult to have a machine decode web-page content and use it as an automated remote management interface. SNMP, on the other hand, is designed to be a machine-to-machine protocol. It provides strict rules that define exactly how various data information is transferred and interpreted. These characteristics allow us to program computer-based systems to request specific management information, interpret it and take appropriate actions, all without human involvement. The standard for data representation allows SNMP-enabled devices to communicate with one another, no matter what processor, application or operating system is in use. In order to standardize device management even further, SNMP defines how each device stores management data. This standardization allows the use of management software from any vendor and the ability to discover the management capabilities of any given device on a network.

There are three main versions of SNMP – v1.0, 2.0 and 3.0. Version 1.0 provides the basic functionality, while v3.0 provides advanced features. This paper focuses on v1.0 only.

SNMP Model

To understand SNMP-based systems, you must understand the SNMP Model. It defines the different components of SNMP-based systems and introduces various terminology.

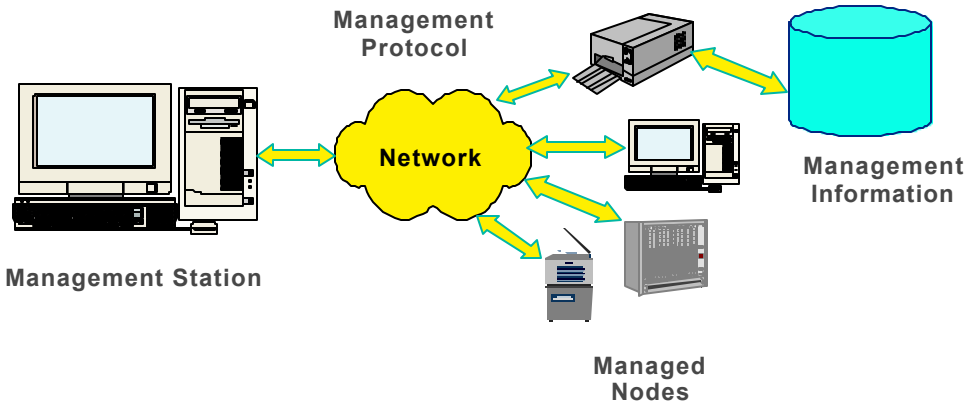

Figure 4 SNMP Model

A typical SNMP-based system consists of one or more management stations and one or more managed nodes. A management station is the console where all management activities are performed. Normally, a PC is used as a management station and is often called a Network Management Station or NMS for short. NMS is connected to one or more managed nodes via the SNMP protocol. Managed nodes are any devices or computers that need to be managed. Managed node is commonly referred as "SNMP Agent" or simply an "Agent." Each agent contains a special database called a Management Information Base (MIB). MIB contains actual data or variables that are used to manage that specific device. In the simplest view, each device exposes its own MIB to remote NMS, whereby NMS may modify or read one or more variables within the device MIB. As variables are modified, the device detects changes in values and takes appropriate actions. The NMS and Agent use SNMP commands and responses to transfer MIB information. Each packet in SNMP is called a Protocol Data Unit (PDU).

The SNMP-based system is an example of a client-sever application. Each agent device acts as a SNMP server, while NMS acts as a client. One NMS client may connect to many server devices. Similarly, one server device may serve many different NMS client. With this client-server approach, end users can use a single NMS software to manage hundreds of devices.

SNMP PDUs

SNMP defines five basic types of packets or PDUs.

1. `GET` to get one or more variables from the SNMP Agent
2. `GET-NEXT` to obtain the next variable from the SNMP Agent
3. `GET-RESPONSE` to respond to NMS from the SNMP Agent
4. `SET` to set one or more variables within the SNMP Agent MIB

5. TRAP to send an asynchronous message from the SNMP Agent to NMS

SNMP uses a command-response format to transfer data to and from the SNMP Agent and NMS. Normally NMS will always initiate a transfer by issuing the GET command. The SNMP Agent responds by sending a GET-RESPONSE message. The only exception to this is TRAP. The SNMP Agent can be configured to send a TRAP message to NMS if certain events occur. The TRAP is an asynchronous message to indicate the occurrence of an important event. NMS would receive this message and may act accordingly. Figure 5 shows a typical SNMP Agent, NMS transaction.

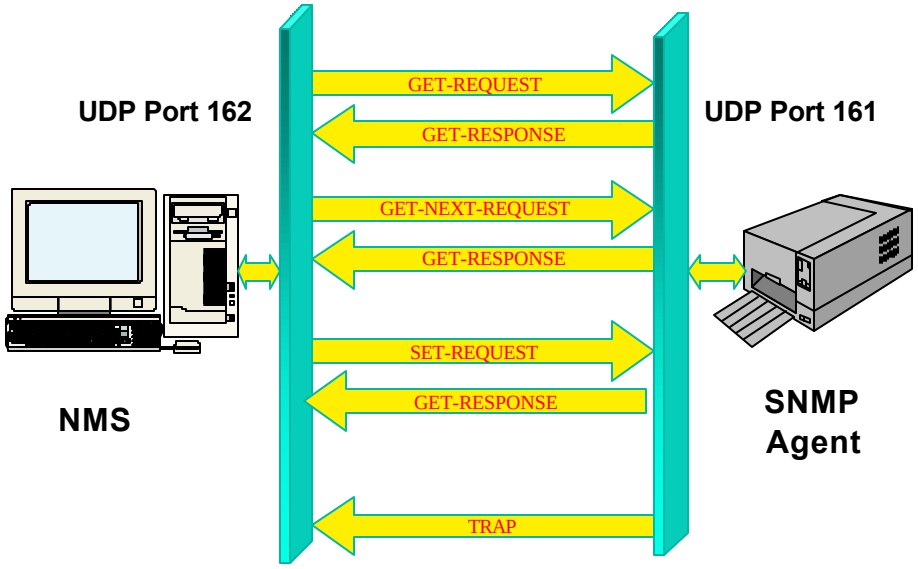

Figure 5 SNMP Interaction

If using UDP, the SNMP Agent listens on the UDP port 161 to receive commands from NMS. The SNMP Agent sends TRAP to NMS on UDP port 162. With separate UDP port numbers for receiving command and TRAP, a single device may act as both NMS and Agent. As you will see in later sections, this feature is useful for distributed control.

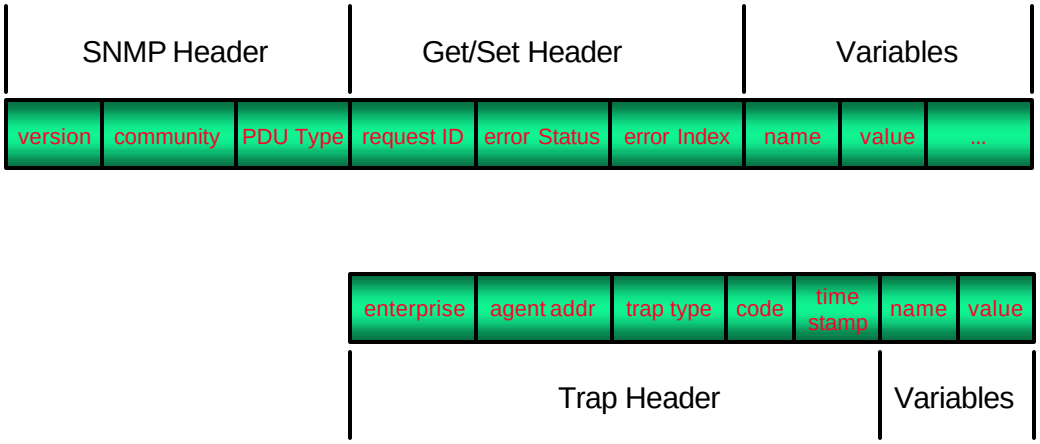

Figure 6 SNMP PDU Format

All SNMP PDUs use a common SNMP header. The *version* field describes the SNMP version in use. A value of '0' indicates SNMP v1.0. The *community* field is a name for a community of SNMP Agents that need to be accessed. In SNMP terminology, more than one SNMP Agent and NMS form a community. To gain access to community resources, you must know the name of the community. This is synonymous to a password, although there is no login name and password pair. Normally, this is a plain-text field unless the advanced encryption features of higher SNMP versions are used. By default, most SNMP Agents use the community name of *public*. Embedded systems that are exposing critical information should use some non-standard community name and perform extra checks to verify the NMS before responding. The *PDU Type* specifies the type of PDU this packet carries. For each of the five messages we reviewed earlier, there is a corresponding *PDU Type* value; the rest of the packet format depends on *PDU Type* value.

GET, GET-RESPONSE and SET messages use the same packet format, while TRAP uses a different format. The *request ID* is used by NMS software to track a command and its response. The *error status* and *error index* is used by the SNMP Agent to report any errors during the processing of the command. The *variable* section of the packet consists of zero or more variable-value pairs. The GET command packet uses NULL as the variable value to indicate that the value is not known or applicable. The GET-RESPONSE command from the SNMP Agent contains both name and value, unless the specific variable was unknown or invalid. For the SET command, the NMS must specify the value that is to be written to a specified variable. As shown in a later section, each variable has read/write permission. If an attempt is made to modify the read-only variable, the SNMP Agent will respond with the appropriate error values in an *error status* field with the *error index* pointing to the index of the variable that caused the error. A PDU may contain one or more variable-value pair. The exact number is limited by the total length of PDUs. For UDP transport, total PDU length, including MAC and IP header, is limited to maximum the MAC packet length. For example, for Ethernet it is 1536 bytes. In the case of TCP transport, PDU length can be more than the MAC limit. TCP will fragment and re-assemble long PDUs at the receiving end.

Unlike GET/SET PDUs, TRAP is a unique message. In the case of GET/SET PDUs, NMS always initiates the transfer, and the SNMP Agent simply responds to the NMS requests. The Agent is not expected to send unsolicited responses. If there is a need to do so, the Agent may use TRAP messages. TRAP, as the name suggests, is a trap set by NMS on certain variables in the SNMP Agent and MIB. The TRAP message specifies that when certain variables change beyond a given limit, the Agent should send out an asynchronous message to NMS. The Agent may use TRAP to notify NMS of some important events proactively, thus relieving NMS from continually polling the Agent. When there are many Agents on a network, TRAP can significantly reduce network traffic.

The *enterprise* field in a TRAP PDU identifies the Agent that issues this message. In a given system, this is usually a unique Object Identifier (OID) string. The *Agent Address*

is an IP address of the Agent. The *Trap type* specifies the type of trap. There are a total of seven types of traps; most of them are designed for high-end computer systems with more than one network interface. One that is important for 8-bit systems is the *enterprise-specific* trap. This trap type allows the system designer to identify a custom TRAP condition. The *code* is used to provide more information about TRAP. In case of the *enterprise-specific* TRAP, *code* may be used to describe the category or module that cause the trap. The *timestamp* is the time at which the Trap condition occurred. It is the hundredths of seconds since the Agent was last initialized. The TRAP may also contain the variable-value pair to provide the exact variable state at the time of the Trap condition. This allows the Agent to provide notification and related information in one packet, rather than NMS separately requesting it.

Management Information Base (MIB)

As discussed earlier, each SNMP-Agent device maintains a database called the MIB. The MIB is a collection of variables. If you are familiar with 'C' language, you may think MIB as a structured data file containing many variables. SNMP calls these variables as objects. Like 'C' language data variables, each variable in MIB is associated with a specific data type. This data type describes exactly how to interpret data contained by a specific variable. Each object in MIB is assigned a unique OID. The NMS uses OID to refer to specific variables in the Agent's MIB.

As we mentioned earlier, SNMP specifies how each data item should be stored in the Agent's database. To accomplish this, the SNMP uses another specification called Structure of Management Information (SMI). The SMI is a tree-like structure that specifies how individual variables are to be stored in a database.

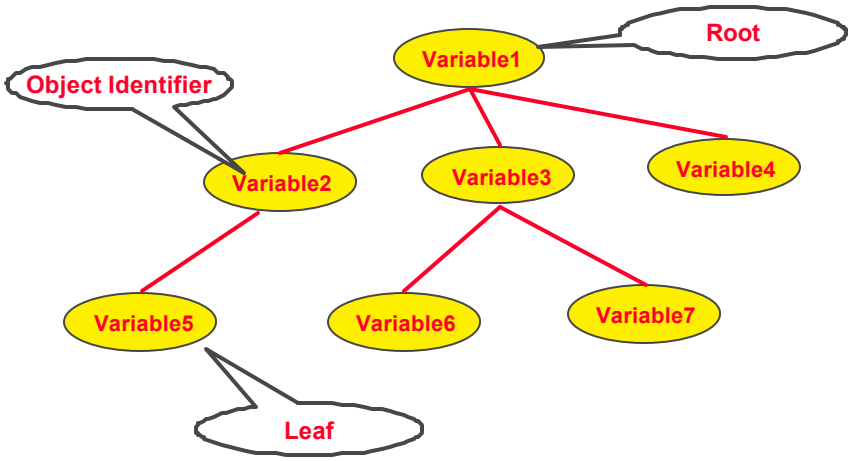

Figure 7 Structure of Management Information

At the top of SMI, there is a root node. Each node contains zero or more child nodes, with the exception of those at the bottom of the trees, called "leaf nodes". The leaf nodes contain the actual data. All nodes above the leaf nodes define a unique path to a specific leaf node. With MIB stored in such a tree-like structure, any NMS can traverse through

the entire database without visiting a variable more than once. This capability is essential given that NMS does not have to have prior knowledge of the Agent's database. The NMS must be able to determine all variables supported by the Agent.

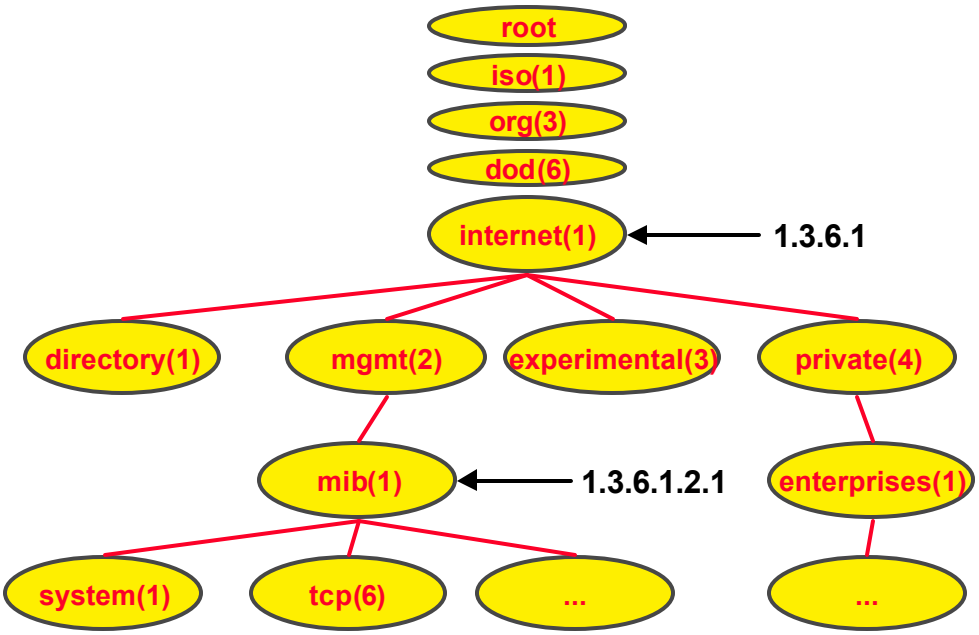

Figure 8 Example of SMI

The SMI is a method widely used by many operating systems and applications. SNMP uses a MIB tree named *internet* below the *dod* node. The sub-tree *mgmt* defines various sub-trees to describe IP protocol statistics and system information. RFC 1213 describes MIB-II that covers *mib* and all nodes below it.

In the SMI notation, each node in the tree is assigned an Identifier number and English name. The actual communication packets use numbers, while the names are for human readability only. Each node in the tree is uniquely identified by its unique path from the root. For example, *internet* node is identified as 1.3.6.1 or i*so*(1).*org*(3).*dod*(6).*internet*(1). This unique string of numbers is an OID, which is written in a dotted-decimal notation similar to one used to write IP addresses. There is no limit on the maximum value of node or length of OID. The identifier value of each node is assigned by the Internet Assigned Numbers Authority (IANA). To conserve the number of bytes required to encode OID, Internet node is referred to as 43.6.1. As a result, all SNMP MIBs will always have '43' as its first number.

Of all the nodes shown, *enterprises* (1) is an interesting one. Unlike other nodes, you may add a sub-tree below this node. This node is designed to host private-enterprise MIB sub-trees. As discussed in later sections, when you are developing your own SNMP Agent, you would need to create a MIB. To do that, you must first obtain your private OID. That private OID will exist below the *enterprise* (1) node.

Abstract Syntax Notation (ASN)

If you were to read RFC 1157, the document that explains the SNMP protocol, you will find that the contents of the SNMP packets are described using a special notation called the Abstract Syntax Notation version 1 (ASN.1). In addition, ASN.1 is also used to describe MIB structure. Figure 9 shows part of ASN.1 syntax that describes the Internet SMI object called “udpTable”.

```
org         OBJECT IDENTIFIER ::= { iso 3 }
dod         OBJECT IDENTIFIER ::= { org 6 }
internet    OBJECT IDENTIFIER ::= { dod 1 }
...

udpTable OBJECT-TYPE
      SYNTAX SEQUENCE OF udpEntry
      ACCESS not-accessible
      STATUS mandatory
      DESCRIPTION
            “A table containing...”
      ::= { udp 5 }
...
```

Figure 9 Example ASN.1 Syntax

ASN.1 syntax describes the different attributes such as OID value; data type, access type and so on. If you are developing an SNMP Agent and want to allow your end users to use third party NMS software, you must create an ASN.1 file describing the MIB of your device. To create the ASN.1 file, you do not have to learn the syntax. There are many commercially available ASN.1 syntax authoring and compiler tools. These tools may provide a graphical interface to define the MIB, and may generate an ASN.1 file for you.

Many of the commercially available PC-based SNMP-Agent development tools use an ASN.1 file to generate part of the Agent source code and the function stubs. Once you have the function stubs generated, you are required to provide necessary logic that allows a generic SNMP agent library to read and write specific the MIB variable. This type of development support can indeed shorten the overall SNMP-Agent development cycle. However, when it comes to 8-bit systems, you may not be able to afford this luxury. Later sections discuss this in more detail.

Binary Encoding Rule (BER)

In a previous section, we reviewed ASN.1 syntax. ASN.1 describes how MIB is structured. It does not, however, describe how MIB information should be transmitted or received. For that, SNMP uses another method, called Binary Encoding Rules (BER). BER tells us how each object in MIB is to be encoded and transmitted or received in a SNMP packet.

Figure 10 BER Syntax

According to BER, each data variable consists of a *Tag* byte, one or more *Length* bytes and zero or more *Value* bytes. The *Tag* byte describes the data type of the variable. The SNMP uses native data types, originally defined by SMI, and defines many more.

Figure 11 shows an example of BER encoding of Integer 49.

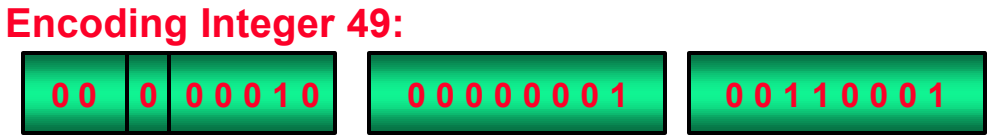

Figure 11 BER Example

Developing an 8-bit SNMP Agent

By now, you should have a basic understanding of SNMP and its usage. If you agree that SNMP is the solution to your remote-device management problem, you need to make an important decision. Should you use an off-the-shelf SNMP Agent toolkit or develop your own? In any case, developing an SNMP Agent library is not simple, even though SNMP stands for *Simple* Network Management Protocol. Irrespective of what method you select, there are certain steps you must follow.

1. Select SNMP version and transport protocol
2. Obtain a Private Enterprise Number
3. Define your device MIB structure
4. Write Get/Set logic for each MIB object
5. Test with desired NMS software

Select SNMP version and transport

As we reviewed earlier, there are multiple SNMP versions. SNMP v1.0 is the most widely used and easiest to implement. Higher versions add more commands and provide advanced features like encryption and remote configuration. For 8-bit systems, SNMP v1.0 is the most suitable version. Most of the 8-bit systems do not need advanced features offered by higher versions of SNMP. Basic command sets provided by v1.0 should be enough for most 8-bit systems. Due to its simplicity, SNMP v1.0 results in smaller code footprint and also requires less processing power.

The SNMP itself is an IP application that needs some underlying transport protocol to transfer its packets to other nodes. As a system designer, you may use either UDP or TCP as your transport protocol. UDP provides less reliable transport, but offers simple and relatively smaller implementation. TCP is a more reliable, connection-oriented protocol and results in larger, complex implementation. Since 8-bit systems have limited resources, UDP is the most desirable transport protocol for an SNMP Agent implementation. Many of the NMS software implement a response-timeout check. As a result, if a response from a specific Agent is not received within a preset timeout period, the NMS software would try again. This check would ensure reliable command-response

transfers. If a TRAP sent by the Agent is lost, the Agent may implement some simple handshaking mechanism to verify that the desired recipient did receive TRAP.

Private Enterprise Number

If you are planning to design a device that is expected to co-exist with other commercially available devices, you must obtain a "Private Enterprise Number." An Enterprise Number is one of the nodes in the Internet MIB. This number is unique to each device manufacturer. You must apply to the Internet Assigned Number Authority (IANA) at www.iana.org/cgi-bin/enterprise.pl to obtain your own number. Unlike other registration processes, this is completely free. But you may only obtain one enterprise number per organization. Once you have your own private enterprise number, you may then create child nodes below this node and add as many sub-trees as you like.

Define your MIB

The next step would be to design your MIB structure. This would depend on the extent of your device is remote management capability. You must describe your MIB structure using ASN.1 syntax. You may create as many ASN.1 files as required to describe your MIB structure. You would distribute this file to your end users, which would then load them into their own NMS software. As we discussed earlier, you may either create the ASN.1 file manually or use a commercially available "MIB builder" graphical tool.

An SNMP specification requires that certain MIBs be supported, while others are optional. The MIBs that provide protocol statistical information may not be appropriate for small 8-bit embedded systems. There are some "system" MIBs that you must support even if it is not applicable to your device. Many of the NMS software rely on the "system" MIB to identify and discover devices. If the expected MIB is not present, exact behavior is dependent on NMS software. In that respect, it is always prudent to test your SNMP Agent with as many NMS software as possible or at least with those that you know will be used by your end users.

Get/Set Logic for MIB

This is the most crucial step in your SNMP Agent development. This is where you write the code to bring your Agent alive. If you were developing on a 16- or 32-bit based system using a commercially available SNMP Agent toolkit, you might be following a similar procedure depicted in Figure 12.

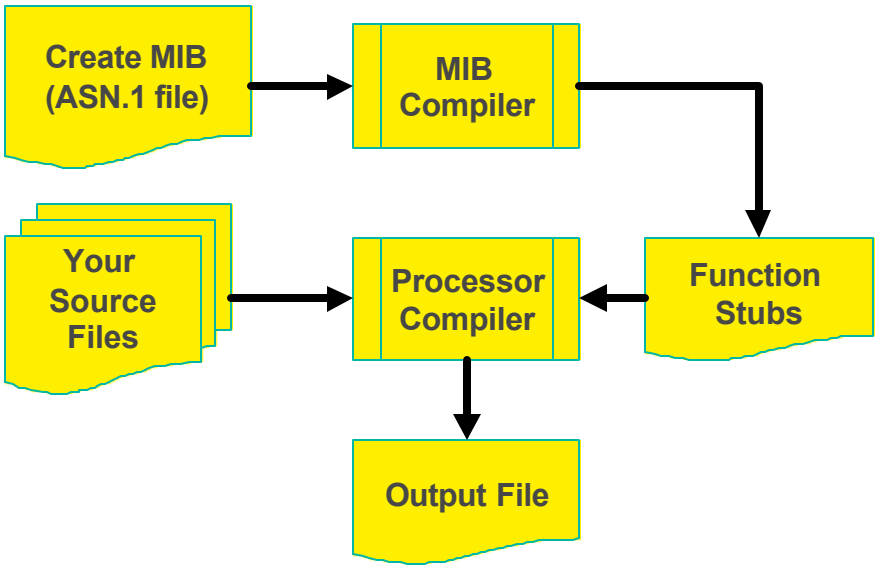

Figure 12 Typical High-end SNMP Agent Development

Typical high-end SNMP Agent toolkits provide an ASN.1 MIB compiler. You compile your ASN.1 MIB file using this compiler, and it generates 'C' language (or whatever programming language it supports) function stubs. There may be one or more function stubs for each MIB object. These function stubs are usually callback functions from the SNMP Agent library to your application. You are required to fill in these function stubs to allow SNMP Agent library to "Get" and "Set" its value. Toolkits like these can greatly reduce the overall development time and simplify the Agent implementation. There are very few SNMP Agent toolkits available for 8-bit systems. In addition, providing such an "automated" code generator might create unnecessary overhead for 8-bit systems. Microchip Technology provides a free SNMP Agent library for its PIC18 family of microcontrollers. This library defines its own custom MIB syntax, specially designed for 8-bit systems. Included PC-based command-line tool compiles the MIB file and generates compact binary file for microcontroller firmware.

The MIB Compiler approach requires that you provide run-time functions for every MIB object even if there are constant objects, whose value does not change over life the of the application. Since we are talking about 8-bit systems, our goal should be to use as few hardware resources as possible. It would be beneficial to store constant objects into the program memory and let the SNMP Agent library read it directly without involving the application. Also, many of the SNMP Agent libraries require run-time creation of the MIB structure even though the MIB never changes over the life of the application. These applications are required to call various library functions to create the MIB structure upon the application startup. If the library was to provide a utility that creates the MIB at the design-time, you could save a significant an amount of program memory and processing time.

NMS Software

Before you begin the SNMP-Agent development process, you will need to select NMS software. NMS software is the front end of your SNMP Agent device. There are many commercial and non-commercial NMS software programs available, with varying features and capabilities. Although SNMP is a standard protocol, and SNMP Agents should seamlessly work with all SNMP-compliant NMS software, it is always a good idea to verify your device operation. Any given NMS software may expect certain MIB to be present before proceeding with other operations. If your device does not support that, the MIB and NMS software may not work properly. The NMS software requires that you load ASN.1 file for Agent device(s) to be managed. With ASN.1 file loaded into its memory, NMS software can label each MIB object in a user-friendly manner. Most NMS software can traverse any SNMP Agent and display its MIB content, even if the ASN.1 file was not preloaded. In that instance, the NMS software would not be able to label MIB objects in English-like names – all you would see are the dotted-decimal OID names. Figure 13 illustrates how most NMS software operations work.

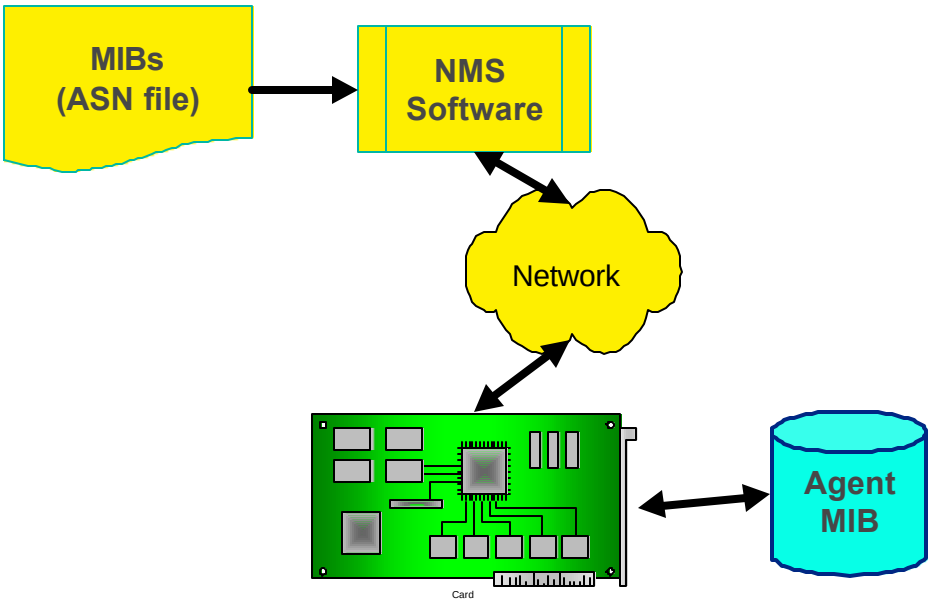

Figure 13 Typical NMS Software Operation

Many of the NMS software are also capable of displaying the Agent information in a graphical format. It may allow you to create graphical objects representing the Agents being managed, thus may simplify management of the devices. Some software provides script capability. You may program such NMS software to automatically respond to certain Agent activities. When you have many devices to manage, automated management support can reduce management mistakes and provide consistency.

There may also be some cases where off-the-shelf NMS software may not be sufficient for your application. Your application may require some custom behavior or processing in a given amount of time. You may also need to integrate NMS software functionality in your custom application. All these requirements call out for your own custom NMS software. Many commercial software vendors provide a SNMP NMS software

development kit in the form of a library or an Active X objects for Microsoft Windows® operating system. If you prefer a challenge, you may even use basic socket API to create your own SNMP NMS software. A custom NMS software may also allow distributed control, where a device may act as both the NMS and the Agent. With such a dual role, a device may request status information from other devices and take appropriate actions locally.

Conclusion

The SNMP, although originally developed for computer networks, can be used for embedded systems. With careful decisions, you can use it on 8-bit embedded systems. There are many commercial and non-commercial SNMP solutions available for 16- and 32-bit platforms. For 8-bit systems, the development options are limited. Even with the few that are available, you need to review them carefully and select the one that best suits the application. This paper provided a high level overview of the SNMP protocol and the various steps involved in developing a SNMP Agent on 8-bit embedded systems.

References

Case, J., Fedor. M., et. al. (1990). *A Simple Network Management Protocol (SNMP)* : RFC 1157.

McCloghrie, K., Rose, M. (1991). *Management Information Base for Network Management of TCP/IP-based internets: MIB-II* : RFC 1213.

Rajbharti, N. (2003). AN870 - An SNMP Agent for the Microchip TCP/IP Stack. Microchip Technology, Inc. www.microchip.com.

Stevens, W. (1994). TCP/IP Illustrated Volume 1. p. 359-389. Addison-Wesley.

Tanenbaum, A. (1996). Computer Networks (3rd ed.). p. 630-643. Prentice Hall : NJ.