Adding custom MIBS to NET+OS V7.X's SNMP component

1 Document History

Version	Initials	Change Description	
V1.0	JZW	Initial entry	
V1.2	JZW	Add Glossary of terms, general clean up	
V1.3	JZW	Roll in editorial feedback	
V1.4	JZW	Further edits	

2 Table of Contents

3 Introduction

This white paper contains a more detailed explanation of the methods and procedures required for adding a custom MIB to NET+OS V7.X SNMP component. This includes both converting your MIB into C code and adding callback functions to the application thus giving the SNMP component of NET+OS access to your device data. For a quick overview of the process required for adding MIBs to your application, Digi recommends that you read the readme file associated with the example application entitled snmpv3.

3.1 Problem Solved

Integrating SNMP into an embedded device is not easy. The protocols and standards integrated into SNMP are made to make interfacing into SNMP generic and flexible but in doing so, understanding the how to is complex. The NET+OS's SNMP component has gone a long way to separating the work of accessing your device data from the mechanics of integrating your data into the SNMP data structures.

This white paper shows that using NET+OS V7.X's SNMP component, following the directions already contained within the NET+OS documentation set and using some additional details laid out here, integrating your data into SNMP is just not that hard. We would like to think that when you have completed reading this paper and looking at the associated sample application and sample MIB that you'll say, "Oh is that all there is!"

3.2 Audience

This paper is of a technical nature. It is intended for software engineers and software practitioners with knowledge of SNMP, MIBs, NET+OS V7.X and development experience in Green Hills, GNU (CLI) or GNU/ESP environments. In order to debug an SNMP-based application you will also need to have some knowledge of the TCP/IP protocol stack and the use of a network protocol analyzer.

3.3 Assumptions

This paper assumes that you have access to a Green Hills Multi or GNU NET+OS V7.X development environment. We do not believe you will take in as much by just passively reading this document as you will by reading it and trying out the sample application simultaneously. Additionally, we presume that you have access to a MIB browser. You will require a MIB browser in order to access the SNMP component of NET+OS. The MIB browser must also have the ability to integrate new MIBs into its MIB database (sometimes called MIB compiling).

3.4 Scope

This document adds additional details to already existing SNMP and MIB-related NET+OS V7.X documentation. It is intended to help customers get "over the hump" of adding a custom MIB their application. The feeling is that once a customer has added one, adding additional custom MIBs should be that much easier.

The following is a list of items and subjects that this document does NOT cover:

- Instructions in C programming
- Instructions on the details of building a custom MIB
- Details of the SNMP protocol
- Details of ASN1
- Instructions of developing NET+OS applications under Green Hills or GNU
- Instructions on running a MIB browser or compiler
- Instructions on TCP/IP protocols
- Instructions on using a network protocol analyzer
- Instructions on downloading NET+OS applications into a Digi device
- Anything referencing .NET or LINUX

Our feeling is that there are a plethora of texts dealing with these subjects and taking time to cover these subjects would be a waste.

If you have weaknesses in some of these areas, the following are some texts that we would recommend:

Understanding SNMP MIBS by Perkins & McGinnis
Managing Internetworks with SNMP by Miller
Internetworking with TCP/IP by Comer
The Digi web site http://www.digi.com

3.5 Theory of Operation

Device data is your data. You created it; you own it and you understand it. It is not the purpose of the SNMP component of NET+OS to have any understanding of your data. What the SNMP component does do is act as a formatter and a conduit for your data between your Digi device and an SNMP MIB browser.

SNMP uses a complicated protocol for formatting and transferring data between your device and your SNMP management station (generally an SNMP Browser). The NET+OS SNMP component abstracts the ASN1 formatting complexities from your application. Thus you do not have to worry about it. All you do have to worry about is physically accessing your data, through callback functions, and returning that data to an unknown caller. If you have used the "stub function" callbacks that the AWS component utilizes, these MIB-related callback functions are not that far removed from the AWS-related stub functions.

So what's required? You need to write your MIB. Your MIB is where this entire process starts. You'll need to run your MIB through Digi's MIB to C utility (mib2c). The mib2c utility reads in your MIB file and generates a number of C and H files, specifically geared to your application and your MIB. You'll need to make some edits (to your code) that are explained in a file created as a result of mib2c converting your MIB (<mib file name>.ins). Next you'll need to write callback functions that are tasked with either reading or writing your device data. This will be the most time consuming part of this

process. Once you have completed that, you'll need to compile and build your application, with the SNMP component included. Before downloading your application into your device, for the first time, you'll need to "compile" your MIB into your MIB browser. You will save yourself a lot of time by performing this step before attempting to debug your application.

What should happen? You'll point your MIB browser at your device. You'll connect to your device. You'll tell your MIB browser to walk through your device's MIB. If you set a breakpoint inside your callback functions, your application should "break" when the NET+OS's SNMP component attempts to read the first item of your new MIB. At that point you can debug your application.

3.6 Conventions

4 Basics

4.1 SNMP

Simple Network Management Protocol (SNMP) has become the protocol used for managing network-connected devices. SNMP is layered on top of UDP/IP. SNMP can be broken into two primary components, namely managers and agents. The manager might be the user using a MIB browser to access some device the user wishes to manage (control in some way). The agent is a software layer, generally residing within a device that accesses device data.

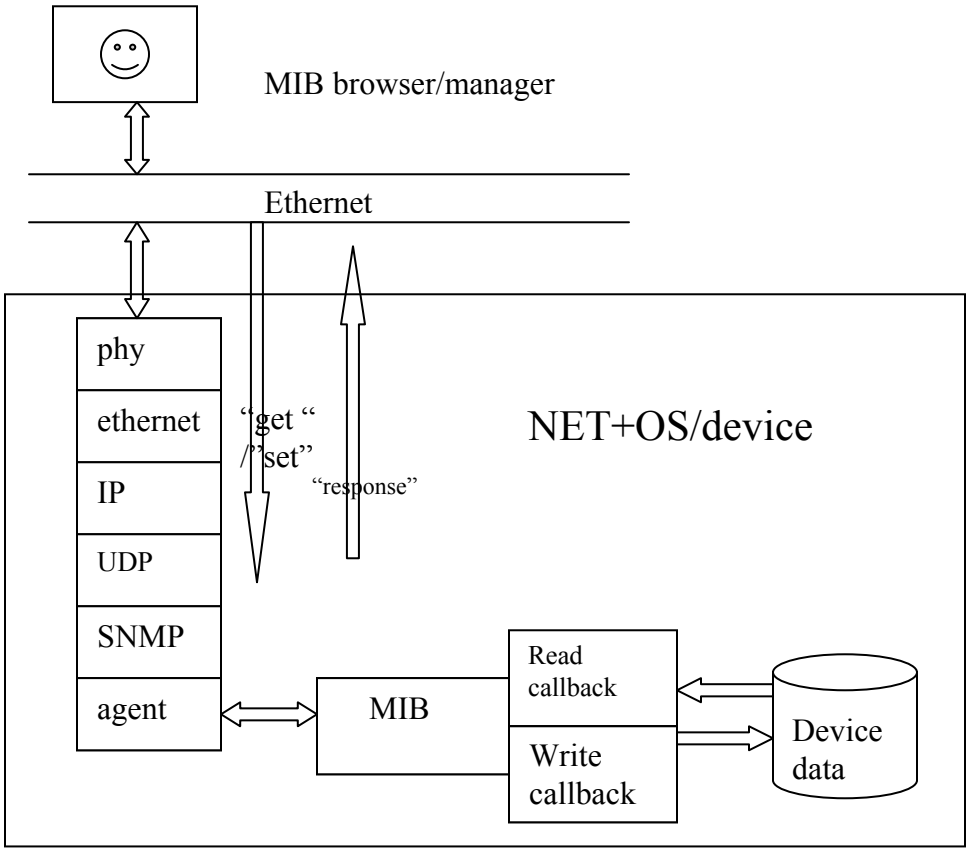
The device being managed contains objects. The objects are housed in databases, accessible by the manager through the agent. These databases are arranged in logical structures entitled management information bases (MIBs).

4.2 MIBS

As mentioned above, your device data is contained in data structures entitled MIBs. A MIB is a very precisely defined structured document that you create. It is editable with a text editor. The structure, layout and keyword set of a MIB are beyond the scope of this white paper. To further understand MIBs we recommend the book Understanding SNMP MIBs by Perkins and McGinnis and published by Prentice-Hall PTR. Your MIB(s) is used by the mib2c utility to convert your MIB into C code that can then be integrated into your application. The MIB is used to generate data structures and functions used to access the device data that the MIB represents to the agent.

4.3 Connections to Device Data

The following diagram describes the parts of an SNMP application:



4.4 Utilities

4.4.1 Mib2c

The mib2c utility converts your MIB into .c and .h files that you can then integrate into your application. In addition, the mib2c utility produces an .ins file which contains instructions and code that needs to be moved into some of the .c and .h files produced by running the mib2c utility. In addition to structures, the utility generates stub functions that you fill in with the actual implementation of your get and set operations. The files generated by virtue of running mib2c are as follows:

- <your mib name>.ins
- <your mib name>_local.c
- <your mib name>_local.h
- <your mib name>_var.c
- <your mib name>_var.h

Additionally, the example application snmpv3's readme file contains an excellent explanation of usage model for the mib2c utility. Digi recommends that you read this file before attempting to run the mib2c utility.

4.5 Your tasks

In order to integrate your MIB into your application, you have three main tasks as follows:

- Run your MIB through the MIB2C MIB compiler
- Write your callback functions into the .c files generated by the MIB2C utility
- Integrate (compile) your MIBs into your MIB browser

The first task was discussed in the previous section entitled utilities. The second two tasks are discussed below.

4.5.1 Writing callback functions

The most application specific task involved in implementing MIBs is writing the callback functions. You'll need two types, namely get and set. The get type callback function reads data from the MIB and returns that data to the manager. The set type callback function takes updates from the manager and writes that data to the MIB. In the section entitled Application Explanation, I will explore the specifics of the application associated with this white paper.

Your callback functions are called from functions in the file <your app name>_local.c in a function probably called something like tf<mibname>_entry_get(). The function has two parameters, namely exact and a pointer to your data structure. Exact tells your callback function how to interpret the index (used with tabular data). Exact tells whether you are looking for an exact match or the next one closest to the index you are looking

for. The comments generated instruct you on how to manage the index. The index itself is contained in the data structure.

Additionally you'll need to create an initialization routine. The purpose of the initialization routine is to copy initial values from some tables you'll need to set up into the in-memory MIBs. In this way your MIBs will have values when your MIB browser first makes contact with your device. These could have real values, or they could be blank strings and integer zero. This piece is application dependant (up to you).

4.5.2 Compiling MIBs into the MIB browser

Before you can successfully walk your MIB, including your newly included MIB, you need to perform the task of "compiling" your MIB into your MIB browser. In the book "Understanding SNMP MIBS", Perkins refers to this as a "backend MIB compiler" built into a MIB browser. The purpose of this step is to allow the MIB browser to access the definition objects in the MIB in a more expeditious manner.

The specifics of MIB browser-based backend MIB compilers is quite MIB browser specific and thus beyond the scope of this paper. You will, therefore, need to consult the help and/or documentation of your MIB browser to understand how it handles this activity. But I must emphasize that you should not begin attempting to walk your MIB, looking for your new MIB until you have performed this task.

5 Application Explanation

At this point in this paper, you will want to begin editing the source files of the attached application. I will be making liberal references to certain files in order to explain how to bolt your callback functions into the SNMP component.

Example Source

5.1 jzwpeoplemib_local.c

I chose to place my application-specific code in a separate file entitled mibstub.c. This follows the model of the AWS coding. In jzwpeoplemib_local.c there are two functions, entitled tfpeopletable_entry_get() and tfwrite_peopletable_entry(). You will notice that all these functions do is call my functions. I could have done all of the callback function coding in this file, but I felt it was architecturally purer to implement the functions in two files.

5.2 mibstubs.c

mibstubs.c is the file where all the real work gets done. You will notice that there are nine (9) static tables. This is the initialization data used to fill the MIB. Since the MIB contains people information, the application contains tables of names, addresses, cities, etc.

RAMTableInit() is used to initially fill in the MIB with initialization data. You'll notice that I have defined a variable numberOfEntries to calculate the number of loops needed to fill the table. This is done so that you could add entries and recompile without touching loop code. You do, though need to ensure that you add entries to all of the tables.

Space is allocated for the MIB (if required). After that the values in the static tables are written into the RAM-based MIB. When done a global flag is set, signifying that initialization has been performed. We'll discuss why this is done, later.

getMeATableEntry() implements the get operation. The purpose of this operation is to return one set of entries for one index of the MIB (assuming a table). The pointer to the entry (the place where you place the data) comes into the callback function containing an index. This tells you which entry to return. Because of the way SNMP gets are performed the index is used in conjunction with the exact parameter. If exact is one, then that exact entry must be available, else an error is returned. If exact is 0, then if the index entry is not available then the next one can be returned. Bottom line is all this is doing is retrieving the ith entry (based on index) and returning that entry. Everything else is taken care of by the NET+OS SNMP component.

writeMeATableEntry() is a little more complex than getMeATableEntry(). Your write callback function is passed an index, a "what", a value and a length (of the value). The index is straight forward. The "what" represents the object within the MIB that is to be updated. In our case, the "what" could represent the person's name, address, city, state, etc. A write operation only updates one object.

There is one quirk with the indexing. Index is 1-based while (in most cases) the index into the table attached to the MIB is 0-based. So by decrementing the index passed in to the callback function by one, you create parity.

File mibstubs.c contains a global flag entitled RAM_table_initialized. This is set at the end of function RAMTableInit(). If, through some coding mistake you do not call RAMTableInit() before entering one of your get or set functions, this flag allows you to check for a lack of initialization state, and call RAMTableInit() before actually truing to get or write data.

6 Conclusion

Though implementing an SNMP agent implementation might be difficult, I believe I have shown that added a custom MIB to an existing NET+OS SNMP implementation is not really that complicated. By writing a couple of callback functions, implementing read and write operations, your device-based MIB will be accessible, over the network, to your SNMP agent.

7 Glossary of Terms

Callback function – (generally) a user function called by the operating system. Used to allow the user to slightly change the functionality of a running system. Also used to give the operating system access to user (or device) data

CLI – Command line interface. A non-graphical method of accessing commands

ESP – Digi's graphical development environment

MIB – Management information base. A description of data layout used by SNMP for accessing user data

MIB Browser – A utility that is SNMP-aware and capable of exchanging SNMP messages with an SNMP-capable device

mib2c utility – A utility provided as part of the NET+OS V7.X product, that converts MIB files into C files for inclusion into a NET+OS SNMP-based application

NET+OS – A Digi produced embedded operating system and development environment for developing embedded applications for running on the family of Digi microprocessors and modules

Network protocol analyzer – either a piece of software or a software/hardware component that is capable of accessing (capturing) some or all packets on a LAN and displaying those packers in a formatted, human readable format. Examples are etherpeek, wireshark, ethereal

SNMP – Simple Network Management Protocol, a network protocol, which is generally encapsulated within UDP/IP and is used for managing objects on the network.

TCP/IP – Transmission Control Protocol/ Internet Protocol – A connection-based protocol that is part of the IP suite of network protocols

UDP/IP – User Datagram Protocol/Internet Protocol – A datagram-based protocol that is part of the IP suite of network protocols