



SNMP Research International, Inc.

Developing SNMP Agents for Embedded Real-Time Systems

A Technical White Paper

Table of Contents

Overview.....	2
Introduction to EMANATE® Agent Development Tools.....	3
EMANATE Run-Time Extensible Agent System	4
EMANATE/Lite Compile-Time Extensible Agent System	6
Extending EMANATE Agents - The Five-Step Method.....	7
RTOS and CPU Support	11
EMANATE/Lite Footprint Information	12
Sources for More Information	12

Overview

A clear market requirement for all telecom and datacom networking equipment today is support for remote manageability. This is normally provided by an SNMP agent, although other access mechanisms (e.g. command line, web browser) may also need to be supported. In addition to the industry standard management objects defined by MIB-II and other IETF standards (e.g. bridge MIB, RMON MIB), the SNMP agents typically need to support their company's private or "enterprise" MIB objects.

This pervasive requirement for an SNMP agent extends beyond the traditional embedded communications equipment to include the new generation of Internet "appliances", which are frequently based on Linux, Solaris, or other popular operating environments. Companies who deploy these appliances want to use their standards-based network management applications to monitor and manage the processes and applications running on these systems.

The agent developer needs to produce an SNMP agent that is reliable, conformant with the standards documents, and ideally field-proven to interoperate with SNMP-based management systems. They need to complete the development within a defined budget and schedule, and with minimal risk. Frequently, the agent developer is familiar with the system being designed, but is not an SNMP "guru".

Today, powerful tools exist to aid the agent developer. SNMP Research offers agent development toolkits which automate much of the development process. This allows the developer to focus on what they know best, which is the architecture and internals of the device being managed, and not worry too much about the intricacies of SNMP. In addition to speeding up the development and testing process, this method assures field-proven standards compliance and interoperability, and provides a source for patches and upgrades to evolve with changes to the standards documents.

The basic steps to develop an agent are:

- Define, design, and write the MIB document, or import a standard MIB document.
- Run automated tools on the MIB documents to create the extended agent code framework.
- Identify and/or create system instrumentation required for support of MIB objects.
- Map the data structures in the agent code framework to the method routines or other access techniques for the instrumentation.
- Compile and link the complete agent subsystem and test.

The toolkit products come with extensive documentation and example code, and SNMP Research's exceptional support team provides assistance and consulting every step of the way. The agent developer's confidence level of completing an interoperable agent on schedule and under budget is very high.

Introduction to EMANATE® Agent Development Tools

SNMP Research's EMANATE® agent development tools are designed to take the guesswork out of adding management object support to the agent. Unlike many other SDKs, EMANATE's agent development tools assume you have limited SNMP expertise, and output nearly all the source code needed for a complete extension. All that is needed is a MIB document and limited C coding to fill in the stub routines. The development process is well defined and documented, and provides the developer with step-by-step source code examples designed to address both simple and more advanced concepts.

Here is what one developer said about our tools...

Kudos are due your developers at SNMP Research! I've used other SNMP SDKs, and read documentation for still others. They were all quite unpleasant, to be frank. I always felt that I should just have to compile the MIB, provide the MIB data ("instrumentation"), and provide some smarts for advancing table indices. But, the other packages always required me to do that and much more. Not so with EMANATE: there's no rough edges, all the corners have been rounded off. It was so pleasant, that not only was I able to focus on my instrumentation, but I also felt encouraged to flesh out the MIBs (RFC 2248 & 2249), providing instrumentation for some of the more esoteric variables which I otherwise would not have done (and did not do in the past when using other SDKs).

Thanks!

-Dan Newman, Senior Software Engineer, Sun Microsystems, Inc.

Because the EMANATE APIs are isolated from the system dependent layer of the agent code, a developer can reuse the same code to implement agents on multiple operating systems or new underlying chipsets. This means that agent code may be reused for future versions of the managed device. For example, a storage appliance based on Solaris may be re-implemented on the Linux/Intel architecture. The agent code simply needs to be recompiled for the new target system.

EMANATE tools maximize the developer's productivity and reduce the total development investment in the agent project. Agent SDKs which appear to be "free" at first glance begin to cost expensive engineering time and money when developers have to turn the code into something useful with limited documentation and support. Sorting through multiple public domain tools, downloads, and "not quite ready" sample code can be a confusing mess, and the agent developer does not have a clear and well-defined process for going from MIB document to tested agent. EMANATE and the SNMP Research support team provide the complete package and guidance for every step of the process.

EMANATE Run-Time Extensible Agent System

EMANATE: Architecture and Features

The EMANATE (Enhanced Management Agent Through Extensions) run-time extensible agent system was developed by SNMP Research under the direction of Dr. Jeffrey D. Case. Dr. Case is the co-author of the SNMPv1, SNMPv2c, and SNMPv3 standards documents and has been an influential pioneer in standards-based management since the late 1980's.

EMANATE agents are constructed in a modular, hierarchical manner and consist of a Master Agent and zero-to-many subagents which support various MIB objects. The subagents are permitted to connect and disconnect to the Master Agent at will, at which time their MIB objects are added and removed dynamically from the Master Agent. Examples of EMANATE subagents might be the MIB-II Subagent, the RMON Subagent, the Host MIB Subagent, and the FDDI Subagent.

The Master Agent and subagents communicate over an asynchronous message passing interface which is optimized for a particular architecture. Furthermore, EMANATE places no restrictions on the order in which the Master Agent and subagents may be started.

EMANATE provides an easy-to-use system independent API which developers can use for implementing support for their MIB objects in an EMANATE subagent. This system independent API hides the details of the particular system on which the subagent is being developed and permits much subagent code reuse across different architectures. Furthermore, the API provides a well-defined interface to the system dependent portion of the subagent code.

EMANATE was designed to address the problem of multiple SNMP agents on a single platform via a modularly constructed extensible agent. In particular, this architecture is designed to meet the needs of:

- Multiple development teams within a company who at present often must coordinate their agent extensions. EMANATE enables these separate teams to develop their subagents separately without the need for integration of code and compilation of all parts of the code into a single executable.
- Third party or customer extensions to an agent. These parties may sometimes have access to source code but EMANATE eliminates the need for them to integrate their new MIB extensions into this source.

- Subagent developers. As much as is possible of the complexity has been put into the Master Agent, leaving subagents as simple as possible. Through heavy use of MIB tools for code generation, subagent development reduces very much to a cookbook approach.

EMANATE was designed to allow multiple subagents to support the same MIB objects. That is, two subagents can support different rows of a table (e.g. the interfaces table). This is helpful for modular systems where subagents are associated with specific plug-in devices.

EMANATE Master Agent

The EMANATE Master Agent, which is SNMP protocol dependent but MIB independent, functions as the core of the SNMP agent. MIB independence means that the Master Agent does not need to contain MIB-specific knowledge of the objects that the agent supports. The Master Agent contains the agent protocol engine for SNMPv1, SNMPv2c, and SNMPv3, and is in charge of the authentication, authorization, access control, and privacy mechanisms. It is multi-threaded in that each SNMP request is handled by a different thread. This means that agent operation is asynchronous. The Master Agent receives the request, initiates the thread which will have control of that request, and can then process further requests. In summary, the Master Agent does the “difficult” work specific to SNMP encoding, decoding, and packet processing.

The Master Agent architecture includes both system dependent and system independent portions. The system dependent portion includes network packet initialization and routines for sending and receiving SNMP requests. Furthermore, the Master Agent end of the asynchronous message passing interface is system dependent, as are the multi-threading support and non-volatile storage access routines.

Processing of get-next and get-bulk requests, multi-phase set processing, registration, de-registration and lookup of subagents, and subagent management are entirely system independent, as is most of the trap handling.

In most cases of subagent development, the subagent developer need not work with the Master Agent but rather will concentrate entirely upon development of the subagent. This means, for example, that application management developers need know very little about SNMP specifics but can instead concentrate on that which they know best -- their system or application and how it should be managed.

EMANATE Subagents

EMANATE subagents, in contrast to the Master Agent, are protocol independent (they make no assumptions about whether SNMPv1, SNMPv2c, or SNMPv3 is used) and MIB dependent. Because the Master Agent is responsible for the “difficult” tasks means that subagents are relatively simple and easy to implement and test. Subagents, which are built with the Subagent Development Kit (SADK), are modular and are again divided into system dependent and system independent portions. The subagents consist primarily of method routines conforming to the system independent API. Subagents are essentially small programs which provide access to method routines. The typical subagent simply runs in a loop, and awaits requests from the Master Agent whereupon it gathers the information requested and returns the response. This is

an oversimplification; the subagent may do more than simply respond to messages, such as sending traps.

Method routines are the functions which actually collect the information specified in the MIB document. That is, the counters, gauges, etc. that are specified in the MIB document must be instrumented within the system or application. This instrumentation must in turn be accessed, and this occurs in the method routines.

The method routines are divided into a system independent part and a system dependent part. Most, and usually all, of the system independent method routines are generated automatically by the EMANATE MIB tools, and stubs are generated for the system dependent method routines (examples are given below). All of the C data structures and function prototypes necessary for the subagent are also generated by these same tools. Of course, code specific to the system or application being managed is not generated.

Much of the subagent code can be reused in multiple EMANATE environments. That is, if you have developed a subagent for one architecture, then you will be able to reuse all of the system independent code on a different architecture since the API is exactly the same. System dependent code and dependencies which the developer introduces into the subagent will have to be ported.

Additionally, since the APIs and development methodology are the same as that used in other EMANATE family products, the code developed for subagents can easily be integrated into a monolithic SNMP agent (and vice versa).

Subagent development using the Subagent Development Kit is a five step process, which is explained in detail below.

EMANATE/Lite Compile-Time Extensible Agent System

EMANATE/Lite: Architecture and Features

EMANATE/Lite is used to develop a monolithic SNMP agent for an embedded system. The agent is extended at compile-time, which means that in order to add new MIB variables to the agent, it is necessary to have access to the source code and to compile and link in the new extensions to the MIB already supported by the agent.

EMANATE/Lite provides the core SNMP protocol engine that should be installed within each SNMP managed element or device. It receives and responds to SNMP queries and commands issued from SNMP management stations. It provides access to management information for each of the managed protocol layers within the network element.

EMANATE/Lite can be compiled to support SNMPv1, SNMPv2c, and/or SNMPv3. This trilingual option preserves the investment in current management technology while providing a smooth transition to secure SNMPv3-based management.

EMANATE/Lite features include:

- Full support for SNMPv1, SNMPv2c and SNMPv3. This includes support for security and administration, authentication, authorization, access control, and privacy.
- A modular architecture that eases the task of porting EMANATE/Lite agents to new platforms.
- Extensibility using powerful code-generation tools.
- Support for all MIB II variables, including read-write variables (sets), that can be supported without altering the kernel or device driver source code.
- Support for multi-phase set routines which provide compliant write access to the MIB variables.
- A singly threaded agent that processes one protocol data unit (PDU) at a time (synchronous), first in, first out. The agent parses the PDU packet, processes it, and creates the response.

Extending EMANATE Agents - The Five-Step Method

It is likely that those who purchase an SNMP agent intend to add support for their own MIB objects to the agent. This section outlines the procedure to add additional MIB objects using the EMANATE tools.

Step 1: Design, Define and Write the MIB

SNMP Research recommends that developers use standard MIB documents whenever possible. This will save a great deal of time. Otherwise, write the MIB document in SMIV2 format (RFC-2578, RFC2579, RFC2580). The SMIV2 MIB document can be converted with a program called “mosy” into the SMIV1 Concise MIB format (RFC1212) if necessary (using the -1 argument).

Before proceeding to the next step, verify that the MIB document is syntactically and semantically correct. The mosy program will be helpful during the debugging phase.

Step 2: Create the Extension

The specific instructions for Step 2 vary slightly between the EMANATE and EMANATE/Lite products, but the general procedure is as follows:

- 1) There are several example agent extensions supplied with the product. Select one and copy the entire contents of its subdirectory to a new directory. For example, you might choose the agent extension that implements a sample MIB document called the THERMOMETER-MIB.
- 2) Type "make startover" (or "nmake startover" on Microsoft Windows® machines). This action removes any unwanted files that may have been copied to the new directory.
- 3) Put the name of the MIB document file on the "MIB =" line in the Makefile. For example, if the name of the MIB document file is mymib.my, then the appropriate line in the Makefile would be changed to: "MIB = mymib" Note that the line does not include the filename extension. Make sure that the MIB document is also copied into the directory.
- 4) Type "make" (or "nmake" on Microsoft Windows machines). This action will start the EMANATE code generation tools which create C code files. The C code files contain (empty) method routines for the MIB objects. The Makefile will also run the C compiler. The C code files will compile and link, but the resulting executable program will not do anything useful (yet).

Step 3: Create the Instrumentation

MIB objects can only report information which has been gathered by an application, and the agent extension must have access to the information which has been gathered. For example, it is impossible for an agent to support the TCP family of objects if the operating system kernel does not collect the appropriate TCP statistics. If the kernel gathers the necessary information, it must also be available to the agent's MIB structures.

It is possible that the developer might need to add instrumentation to support the new MIB objects. In the case of the TCP group, one might have to add counters or gauges to the kernel. This instrumentation will then be accessed by the agent's method routines.

If the application already collects the information modeled in the MIB document, or if the developer is replacing or augmenting a proprietary protocol with SNMP, then this step is greatly simplified and may even be bypassed entirely.

Step 4: Complete the Method Routines

The EMANATE code generation tool called “postmosy” automatically generates a set of method routine stubs from the MIB document. The method routines are those functions which access the instrumentation.

There are two parts to a method routine: the system independent function and the system dependent function. In most cases, the system independent method routines do not have to be modified by the agent developer. They provide a uniform interface to the system dependent functions, which they call. The system dependent method routines, found in a file called `k_<base>.stb`, localize that part of the code which can vary from architecture to architecture. It is the system dependent method routines which must be altered.

As an example, consider the following stub routine:

```
thermometer_t *
k_thermometer_get(serialNum, contextInfo, nominator)
    int serialNum;
    ContextInfo *contextInfo;
    int nominator;
{
#ifdef NOT_YET
    static thermometer_t thermometerData;

    /*
     * put your code to retrieve the information here
     */

    thermometerData.thrmTemperature = ;
    thermometerData.thrmUnits = ;
    SET_ALL_VALID(thermometerData.valid);
    return(&thermometerData);
#else /* NOT_YET */
    return(NULL);
#endif /* NOT_YET */
}
```

This stub was generated from an example MIB document with a family of scalar objects called `thrmTemperature` and `thrmUnits`. The lines

```
thermometerData.thrmTemperature = ;
thermometerData.thrmUnits = ;
```

are of course not compilable, but they make it very clear what needs to be done. The developer must know:

- what values belong in the various fields of the family structure (in this case, thermometerData), and
- how to access the instrumentation to obtain the values.

The system dependent method routine is where this information must be filled in. The completed function might look like this:

```
thermometer_t *
k_thermometer_get(serialNum, contextInfo, nominator)
    int serialNum;
    ContextInfo *contextInfo;
    int nominator;
{
    static thermometer_t thermometerData;

    thermometerData.thrmTemperature = check_thermometer();
    thermometerData.thrmUnits = get_thermometer_units();
    SET_ALL_VALID(thermometerData.valid);
    return(&thermometerData);
}
```

Step 5: Test the Instrumentation and Agent

After creating the instrumentation, completing the method routines, and building the subagent(s), the developer will need to test the completed agent subsystem. Note that the hardest part of this step may be ensuring that the instrumentation is collecting and furnishing the correct data.

RTOS and CPU Support

EMANATE and EMANATE/Lite are supported on the following embedded system environments. Most popular open systems (e.g. Solaris, Windows 2000/XP, FreeBSD, Linux) are also supported.

RTOS	Reference Processor	Cross-Development OS
Chorus ClassiX 3.1	Intel 80x86	Sun Solaris 2.x
HP/RT 3.01B	HP 9000/743	HP/UX 10.x
LynxOS 2.5.0	Intel 80x86	LynxOS 2.5.0
Nucleus NET 4.0*	AMD 186 (i80c186)	32-bit Windows
ENE A OSE 4.3	Soft Kernel	Solaris 2.x/32-bit Windows
OS/9 3.03*	Motorola 68030/68040	32-bit Windows
pSOS 2.2	Motorola 68030/68040	SunOS 4.1.x
VRTX	Motorola 68030/68040	SunOS 4.1.x/32-bit Windows
VxWorks 5.3/Tornado	Motorola 68030/68040	Solaris 2.x/32-bit Windows
Windows CE*	Intel 80x86	32-bit Windows

*Only available for EMANATE/Lite

The EMANATE family of agent source code products are processor independent. The source code is developed and tested on the specified reference processor. Building for another target CPU is typically a matter of changing a few make files or environment variables.

EMANATE/Lite Footprint Information

An agent's image size varies depending on the features and size-saving features that are turned on and off at compilation time. The following image size data is for the pSOS version of EMANATE/Lite. These values are taken from the linker maps.

- SNMPv1-only agent, with MIB-II support, and some optimization turned on:
 - The total image requires about 253k of ROM and about 128 of RAM, but these numbers include not only the agent, but also pSOS, PNA, probe, pHILE, pREPC, pRPC.
 - For only the agent-related components parts, the agent itself is about 25.9 Kbytes agent and about 49.2 Kbytes total.
- Trilingual SNMPv1, SNMPv2c, SNMPv3 agent on pSOS (but without all of the space-saving optimizations):
 - 328K code and 35.5k of initialized data again including agent with MIB-II but also pSOS, pROBE, pNA, pHILE, pREPC, pRPC.
 - For only the agent and MIB-II part, you get: about 64k decimal of code for the agent plus about 36.5k decimal of code for the libraries for a total of about 100.5k (decimal) of code. You may enable some space-saving optimization by turning off some more debug messages, and therefore be below 96k.

Sources for More Information

For further information about EMANATE, EMANATE/Lite, or other SNMP Research products, please visit our web site at www.snmp.com, or contact SNMP Research at the address below.

SNMP Research International, Inc.
3001 Kimberlin Heights Rd.
Knoxville, TN 37920
U.S.A.

Tel: +1 865 579 3311
Fax: +1 865 579 6565
E-mail: info@snmp.com