The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS℠
VOLUME 4, NUMBER 2 APRIL, 1996

The Simple Times is an openly-available publication devoted to the promotion of the Simple Network Management Protocol. In each issue, ***The Simple Times*** presents technical articles and featured columns, along with a standards summary and a list of Internet resources. In addition, some issues contain summaries of recent publications and upcoming events.

In this Issue:

The Simple Times is available via both electronic mail and hard copy. For information on subscriptions, see page 24.

Introduction to Agent Extensibility

Bob Natale
*ACE*COMM*

As chairman of the IETF's SNMP Agent Extensibility (AgentX) working group <agentx-request@fv.com>, I am very grateful to the publishers of ***The Simple Times*** for dedicating this issue to a series of articles on matters of direct relevance to the AgentX effort. Prior to the IESG approving a formal IETF mission in late 1995, a sizable portion of the SNMP community – mostly implementors rather than users – spent several years in self-organized preparatory efforts. To many informed observers, over much of that time, these informal industry efforts seemed at best unproductive and at worst futile. A great deal of effort was spent enumerating and dissecting all of the significant problems that a standard extensible agent technology would have to confront. In the meantime, a variety of semi-standard and proprietary solutions emerged which seemed to meet many users' needs. None of them, however, achieved overwhelmingly widespread deployment, leading to a mix of seemingly permanent islands of technology. That overall course of events seemed to confirm the official IETF view of the time that a standard solution would either be impossible to reach and/or would be woefully ineffective if, in fact, one could be fabricated.

Nonetheless, the need for a standard base from which to respond to the growing demand for more flexible and dynamic management of multi-component *systems* – as opposed to single devices – continued to rise in importance. It became common, for example, to see computers of the server variety as a collection of manageable components, including interfaces (e.g., MIB-II), operating system and associated resources (Host Resources MIB), peripheral devices (e.g., Printer MIB, Modem MIB, RAID MIB), and key software applications (e.g., RDBMS MIB, MADMAN MIB). And users want – indeed, *expect* – to be able to manage this collection as a single system. Further, the population of possible managed components continues to grow rapidly (e.g., the emerging Applications MIB will open new floodgates). What were once relatively simple devices (from the SNMP perspective, at least) are now complex collections of manageable components. Consider, as but a single example, any modern product which integrates bridge, hub, router, and switch functionality, which also supports a variety of interfaces and protocols (e.g., serial, 10Mb/100Mb Ethernet, FDDI, Token Ring, ISDN, and ATM). The surge in demand to manage the desktop as part of the enterprise and, eventually the requirement to manage a myriad of home appliances and personal accessories as integrated components of an individual's "techno-life support system" both serve to quash any semblance of doubt with respect to the need for and value of a standards-based solution to the problem.

The keys to success in meeting this burgeoning demand, mirror those of the undeniable success of the first generation of SNMP itself: deploy low-cost agents on all the components as quickly as possible; ensure that those agents do not interfere with the components' principal functions; and, give those agents a standard way to interoperate with higher-order software elements.

The IESG chartered the AgentX working group accordingly. Its goal is to define standards-track technology for SNMP agent extensibility. The resulting technology specification(s) must allow independently developed subagents to communicate with a master agent running on an Internet device. (This last sentence merits re-reading.)

The charter stipulates that the AgentX technology specification(s) will consist of at least one to as many as three parts:

- (mandatory) a platform-independent protocol which supports intra-agent communication within a device or local area network;
- (optional) a MIB module, which, when implemented by a master agent, allows an SNMP-based management application to monitor and control the intra-agent communication service; and,
- (optional) a programmatic interface to the services offered by that protocol.

The IESG explicitly directed the working group to develop a solution which is adequate to achieve transparency with respect to whether a SNMP request is processed by a master agent and/or one or more subagents. This is the second of the two major architectural imperatives of the AgentX protocol, the first being the analogous transparency with respect to the subagents' view of the master agent's lineage.

Furthermore, the working group was directed to use good engineering judgment in developing an approach with the smallest reasonable footprint to achieve intra-agent communication. As a consequence of that, the working group may choose to avoid complete transparency, if, at its discretion, this proves to be the more effective approach. In that case, however, the working group is obliged to document its decision criteria for this engineering trade-off.

Finally, the IESG stipulated that:

> "Although the working group will solicit existing specifications and experience in this area, it will produce a vendor-neutral technology specification."

Given that directive, and in light of the sizable amount of time spent previously on analysis of the problems, issues, and requirements, the chair sought to orient the working group toward an initial review of the solution-space (the set of published extensible agent protocols and product specifications that were, individually, meeting users' needs in some fashion). The first article in this special issue, then, is by Dale Francisco `<dfrancisco@strata.com>` of StrataCom, who also serves as the AgentX Editor. This article will give you a brief introduction to the major primary sources and pointers to them so that you can follow up in more depth.

The attempt to focus the working group's attention on this solution-space was not intended to avoid having to deal with the problems, issues, and requirements; however, it was a recognition that the community has collectively spent a lot of time and effort on almost all of those issues. At some point in such a process, you have to be able to weigh the arguments and then make some hard choices. Many people have helped us identify the problem areas over the years and quite a few have helped to formulate a decision and build a consensus around it. However, no one has done more to help us all get to the crux of almost every issue than Randy Presuhn `<rpresuhn@peer.com>` of BMC/Peer. His article is more than just a masterful summary of those past contributions. It is also a new tool for the working group to use in achieving closure on some more of these issues. I am sure that you will have a good appreciation for both the complexity of this problem-space and for the possibilities for effective and efficient solutions after reading Randy's article.

One of the technologies in the AgentX solution-space is Digital's eSNMP, which can be seen as one possible evolution of DPIv2, which is itself another element of the solution-space. In his article, Mike Daniele `<daniele@zk3.dec.com>` of Digital Equipment Corporation, implementer of eSNMP, provides a detailed overview of the protocol itself and explains some of its differences from DPIv2 and its possible relevance to the eventual AgentX protocol. Those readers who would like a concrete idea of what an extensible agent protocol looks like and how it operates will get that from Mike's article. And I should observe that Mike is teamed with

Bert Wijnen <wijnen@vnet.ibm.com> of IBM as "point men" on AgentX protocol design. Their job in that role is to take the consensus positions as they begin to emerge via the working group deliberations and turn them into AgentX protocol specifications via appropriate modifications to the existing DPIv2 specifications. This is how we are mapping the problem-space resolutions back on to the solution-space foundation.

Finally, Dave Bridgham <dab@epilogue.com> of Epilogue presents an uncommonly readable and persuasive analysis of the *proxy* approach to agent extensibility. Subtly, but clearly, Dave hits on the major design difference between the proxy model and the extensible agent model being pursued in the AgentX effort – transparency to the management applications. The goal, in that respect, for the extensible agent model, is to allow the management applications to remain unaware of the fact that the apparently monolithic native agent that is responding to their requests at the well-known service location (such as UDP port 161) is actually an *agent system* of sorts, consisting of a *master agent* and potentially many *subagents*, each representing a particular managed component. Dave's article in this issue outlines the cost/benefit outcome of the information hiding in the agent. The proponents of the extensible agent model and the proxy model have plugged different factors and formulas into that calculation and, not surprisingly, each model will produce a positive cost/benefit ratio in certain scenarios. Hence, both technologies have a role to play in the marketplace. As time goes on and we all gain more experience and the general technology base continues to improve and end-user sophistication continues to increase, it may well be that there will be a pronounced converge of the two models into a single solution.

In Conclusion

Regardless of the arguments, the marketplace will appreciate and benefit from a higher degree of standardization in this area of agent extensibility. Suppliers and consumers of SNMP products and services will gain with respect to both economic and usability metrics. AgentX is the IETF strategy to make that happen!

Overview of the AgentX Solution-Space

Dale Francisco
StrataCom

Like many other successful technical innovations, SNMP was confronted fairly early with the problems posed by widespread acceptance and use. One of the first such problems to arise was this: if you're going to manage everything on your network with SNMP, how can you write SNMP agents quickly enough to keep pace with a constantly changing array of network equipment?

This problem is harder than it may first appear. One of the things that makes it hard is that users of network management systems stubbornly persist in viewing the devices on the network, routers, telco switches, workstations, and so on, as unitary entities that, in principle, can be meaningfully reduced to green, yellow, or red icons on a topology map. Builders of routers, telco switches, and workstations are equally adamant in viewing these devices as small ecosystems (or perhaps bioregions) of sometimes cooperating, sometimes antagonistic, software and hardware entities.

The designer of an SNMP agent is somewhere in the lonely middle between these two camps, aware of the reasonableness and even the necessity of both points of view. Viewing the agent as a provider of management information, not only does it make conceptual sense for one agent to provide all the information for one device, it seems to be enshrined in SNMP itself, for it is written that the agent shalt listen to the manager on UDP port 161, of which there is exactly one per managed device. But viewing the agent as a gatherer of management information, the designer is forced to face the messy world of agent instrumentation. In a workstation, the knowledge of management information is quite likely distributed among many user processes; in a telco switch, it may reside in different modules with different processors and different operating systems. This real-world messiness would seem to be best handled by lots of little agents rather than one big one.

When confronted with such a dilemma, SNMP agent designers, having read the same books and gone to the same schools, soon think "divide and conquer", and not long after that they think of interposing a new layer or protocol. And as is often the case with ideas whose time has come, different people come to similar designs independently. Two of the earliest such designs for extensible agents were DPI and SMUX.

Paleo-agent-ology: SMUX and DPIv1

In 1990 or thereabouts, several people got together one afternoon and designed SMUX, which was later specified in RFC 1227. Their idea was clear and clearly expressed: let the SNMP agent listen for and respond to all management requests for a single device, but let it multiplex the handling of requests among (possibly several) local entities, called SMUX *peers*, each of which is responsible for instrumenting some part of the device's

MIB. By allocating to two different entities the tasks of providing and gathering management information, and interposing a protocol between these entities, the designers of SMUX hit upon an idea that is at the core of all extensible agent schemes that have followed.

Let's pause here to define some terms. An *extensible SNMP agent* is one which allows the addition, at run-time, of support for arbitrary MIB modules that were not known at the time the agent was built. This is quite different from the first generation of SNMP agents, known as *monolithic agents*. An extensible agent is made up of what is known as a *master agent* (SNMP agent in SMUX), a single entity that is responsible for receiving management requests and sending management responses; and zero or more *subagents* that are responsible for instrumenting various parts of a MIB. A subagent takes responsibility for a part of the device's MIB by *registering* with the master agent. The division of labor between the master agent and its subagents allows the extensible agent to appear to be a single entity to the management station; in fact, from the outside, it is indistinguishable from a monolithic agent. But internally, the extensible agent has the flexibility to adapt to changed circumstances by registering new subagents that are responsible for new functionality.

Despite its simplicity and conceptual integrity, SMUX had its problems. Notable among these were:

- The `sysUpTime` problem. RFC 1156 defined `sysUpTime` as "the time (in hundredths of a second) since the network management portion of the system was last re-initialized." This is fine for a monolithic agent, but what does it mean, exactly, when what appears to the management station to be a single agent is actually a menagerie of a SNMP agent and several SMUX peers, all possibly coming and going at different times?

- The out-to-lunch SMUX peer problem. Section 3.1.4 of the SMUX specification (RFC 1227) says that the SNMP agent should process each varbind in an incoming request sequentially, and block when a SMUX peer is contacted. What happens when one peer goes away or hangs, but the agent itself and possibly other peers who are responsible for some of the varbinds in the request are still functional?

- The performance problem. SMUX was essentially an elaboration of SNMP itself; though there were new messages, for instance, to handle the registration of a SMUX peer with the SNMP agent, the core protocol between the agent and its peers was SNMP, so that the number of SNMP PDU decodings and encodings to handle for each management request was, in the best case, doubled.

Having briefly examined SMUX and gained an appreciation of some of the basic features (and problems) of extensible agents, we can step back slightly in time to 1989 and examine another early extensible agent protocol, the SNMP Distributed Protocol Interface (DPI) version 1.0, which was specified in 1991 as RFC 1228. DPI was developed by IBM beginning in 1989; in that year DPI agents were actually being used to manage the NSFnet backbone. IBM eventually developed versions of DPI for OS/2, AIX, AS/400, and for the mainframe operating systems MVS and VM.

Like SMUX, DPI separated the agent functionality into an SNMP agent and one or more subagents. Again as with SMUX, the SNMP agent was responsible for the interface with the management application, while the subagents were responsible for different subtrees of managed objects. An important difference, and one that has appeared in many extensible agent implementations since, was that DPI, instead of using SNMP PDUs between the SNMP agent and the subagent, specified its own lightweight protocol for agent-subagent communication. (In addition, the original version of the DPI specification included an example subagent API.) In 1994, three years after the publication of DPIv1, based on their considerable implementation experience, IBM researchers offered a much more detailed version of the DPI specification, DPIv2, in RFC 1592.

The original version of DPI, though it offered improved performance relative to SMUX, suffered from some of the same problems as SMUX with regard to `sysUpTime` semantics and hung subagents. And perhaps even more than with SMUX, which included a two-phase commit, there were problems satisfying the SNMP requirement that `set` requests appear to take effect "as if simultaneously" when the processing of `set` requests was distributed across multiple subagents.

Recent extensible agents

Based on the promising experience of SMUX and DPI, several extensible agent products and implementations have appeared in the last few years. Extensible SNMP was developed at Digital in 1995 as a sort of modified DPI. It followed the basic DPI model, but simplified some of the message formats by reducing configurable options.

In parallel, as it became clear that there was a large market for extensible agents, other commercial products such as Envoy (Epilogue Technologies) and EMANATE (SNMP Research) appeared on the scene. Typically, the advantages that commercial products offered over some of their publicly-specified predecessors were increased performance (with versions optimized for particular hardware and particular operating systems), and greater flexibility; for instance, increased options for subagents to register which parts of a MIB module, even down to the instance level, that they were responsible for.

Though there's no question that commercial extensible agents have improved the state-of-the-art, it's clear that a standard for interoperability is needed if SNMP agent extensibility is to achieve its true promise. At present, the dream of having a single network device that comprises modules from different vendors, each with one or more subagents cooperating with a device-wide master agent, remains unattainable unless all the vendors happen to be using the same extensible agent product.

The Next Step

The participants in the SNMP agent extensibility working group of the IETF are in the fortunate position of having as examples several alternative implementations of extensible agents, some of which have been field tested for years. We also have a clear and compelling rationale for creating a standard SNMP extensible agent protocol: it is unlikely, in the face of the continuously accelerating growth in networking technology and innovation, that SNMP will remain a universal and useful protocol unless there is a standard way to dynamically add support for new MIB modules to existing agents. As seen in the next article, experience shows what is required of an extensible agent protocol in order to provide the same functionality as the best of the current implementations, and to provide it in an open standard that vendors will find it in their own interest to support.

Overview of the AgentX Problem-Space

Randy Presuhn
Peer Networks, a division of BMC Software

Developers in the area of agent extensibility have identified seven broad areas of requirements for a subagent protocol specification:

1. transport requirements;
2. association requirements;
3. operational requirements;
4. registration requirements;
5. inter-subagent requirements;
6. visibility requirements; and,
7. specification requirements.

Subsequent exploration has led to a more detailed set of possible requirements. This article maps out each of these areas, identifying regions of consensus, disagreement, and research.

Transport Requirements

> *What level of QoS (Quality of service) is required to carry the subagent protocol? Are specific transports needed?*

Subagent protocols can be used both within and between systems. This leads to the requirement that at least one standard transport mechanism be defined. This does not preclude vendors from employing additional transports optimized for specific environments. For example, UNIX domain sockets or IPC mailboxes might be attractive on some platforms. It is generally agreed that vendors should be free to support such transports in addition to any standardized ones. In pre-AgentX discussions, there seemed to be strong resistance to defining a specific transport. Within the AgentX effort, there appears to be general recognition of the value of defining a standard transport, even if vendors will define additional ones of their own.

The minimal quality of service required by most subagent protocols appears to be an 8-bit clean, connection-oriented byte-stream. (There is also one proposal that is based on intra-system UDP, assuming a very low probability of packet loss or duplication.) TCP/IP is recognized as a widely-available transport meeting these requirements. For a subagent protocol to be carried over something not meeting these requirements, such as an unreliable datagram transport or noisy byte-stream, a convergence layer would be needed.

The strongest argument for using a connectionless transport like UDP with a convergence layer is given in RFC 1592: per-process limitations on the number of open file descriptors in some environments. By separating the design of a convergence layer from that of the subagent protocol, we avoid having to add the complexity of data integrity and retransmission mechanisms to the subagent protocol.

A very special case of the transport issue is the possible choice of DLL as a communication mechanism. In this case, however, the level of interoperability is really at the API level, and will not meet inter-system requirements. It is possible to define an API that makes the choice of communication mechanism transparent to the subagent developer.

An issue that straddles the border between transport and association requirements is the handling of confidentiality and authentication. In all the proposals to date, any confidentiality has to be provided by the underlying transport, with no standardized transport mapping to support confidentiality defined. In the existing proposals that provide some level of authentication, weak authentication is built into the agent-subagent protocol. The requirements for authentication and confidentiality between the master agent and its subagents has not received much discussion.

Association Requirements

> *What information is needed to establish, maintain, and terminate an association between master agent and subagent?*

Five major sub-areas of association requirements are:

- privacy handling;
- authentication handling;
- details of association establishment;
- details of association maintenance; and,
- details of association termination.

As mentioned above, whether privacy handling should occur as part of the subagent protocol or be left as a matter for the underlying transport is a design decision with significant implications. There have not been any calls to incorporate privacy into the subagent protocol itself.

The association establishment phases of the published subagent protocols provide for the identification of the subagent using a trivial form of authentication. This minimal identification is required in order to maintain the `sysORTable` and various subagent MIB modules. Whether stronger authentication is needed (in either or both directions) remains an open issue. Whether strong authentication should be left to the underlying transport or included as part of the subagent protocol is also an open issue.

For most subagent protocols, association establishment is a simple two-way handshake. This allows limited negotiation of parameters for the association and transfer of useful bits of information. The parameters that have generated the most interest are limits on the number of varbinds per PDU, timeouts used to detect lockup, and identification of the naming scope. The additional bits of information that have been considered include the time base used to compute `sysUpTime`.

The significance of the requirement for negotiation of the number of varbinds per PDU is currently being debated. The arguments for are primarily in terms of providing some level of compatibility with existing DPI instrumentation; the arguments against are in terms of efficiency and protocol complexity.

For the remaining details of association establishment, the recurring issue is whether specific parameters should be considered properties of a particular registration or of the association as a whole, or whether the association level parameters would serve as defaults for registrations. Since there is a strong requirement to support multiple naming contexts over a single association, and since the timeout characteristics of naming scopes may differ, it looks like the design will be left with a choice between treating these as part of the registration dialog or handling them in both places. A subtle point is that there must be a `sysUpTime` for each naming context, and that these don't necessarily all have the same value.

Association maintenance can take several forms. Depending on the characteristics of the underlying transport, some form of keep-alive may be useful to detect subagent lockup. In fault-tolerant configurations, it may also be desirable for subagents to detect loss of connectivity to the master agent. Previous discussions of the impact of transport outages led to the conclusion that there was no requirement for an association to be maintained across successive transport connections. In environments with highly unreliable transports, a convergence layer providing session maintenance for the association could be used.

It may be convenient to handle `sysUpTime` maintenance (e.g., notification of discontinuities) as part of a keep-alive mechanism, but, since `sysUpTime` discontinuities may be asynchronous to subagent operations, the protocol elements to support `sysUpTime` maintenance need to be decoupled from management-initiated protocol operations.

Most subagent protocols have some provision for an orderly association termination procedure. Only two issues have surfaced here: whether the extensive status codes provided by SMUX and DPIv2 have actual value, and whether termination of a subagent association should affect `sysUpTime`. The emerging consensus seems to be that `sysORTable` is the place to record the comings and goings, and that `sysUpTime` should be unaffected.

Operational Requirements

> *All SNMP and SNMPv2 operations must be supported (at least in the agent role), and the behavior must, from a protocol perspective, be indistinguishable from a monolithic agent's behavior. That the boundaries between subagents should not be visible at this level is taken as a fundamental requirement.*

These requirements raise a number of issues:

- handling multiple naming scopes;
- multi-phase commit protocols to preserve `set` semantics;
- access control issues; and,
- `inform` handling.

A key operational issue is the representation of multiple naming scopes for operations. Tied to registration issues (see below), the essence of the problem is that the mapping of community strings (or the old SNMPv2 contexts) to naming scopes is not one-to-one. Unfortunately, the work in the Entity MIB working group `<entmib-request@cisco.com>` does not appear to be resolving this problem. Conflicting goals surface here for different applications of subagent technology. In some cases, there is a desire to have access to the literal value of the community string from the original SNMP request. In most other cases, what is needed is an unambiguous cookie that will allow a subagent to determine which naming scope should be used to perform an operation.

Discussion of the requirements for multi-phase commit protocols to support `set` semantics has to begin with the recognition, formalized in the error codes of SNMPv2, that there are cases where the "as if simultaneously" requirement simply cannot be met, even in monolithic implementations. Consider, for example, Keith McCloghrie's discussion of this topic in Volume 2, Number 6 of ***The Simple Times***.

The changes required to existing protocols to support the SNMPv2 error codes are relatively minor: adding a response to the second phase of the SMUX commit, or a transaction end on the DPIv2 commit. However, additional phases help in handling important, if seemingly pathological, cases.

The protocol needs to take account the various resource reservation and release strategies that are possible, and to not assume that all subagents have been implemented using the same allocation discipline, since that discipline may be inalterably embedded in the design of the system, of which the subagent is a minor component.

The most difficult case to handle is where the acceptability of a proposed value for a variable is dependent on a variable supported by some other subagent, which is to be modified in the same request. To a certain extent, one might argue that this is poor implementation strategy, poor MIB design, or both.

In general, rollback may not be possible for `sets` with action semantics. Although adding additional phases can bound the problem, the consensus seems to be that these cases are better handled as MIB design issues.

The requirements appear to boil down to a four-phase procedure:

1. local checking and resource allocation;
2. cross-checking for shared resources, which cannot occur until it is known that all involved subagents have done their resource allocation;
3. commit/rollback, which cannot occur until all cross-checking is complete; and,
4. resource release/undo is needed to bound the transaction and to ensure that the master agent gets the correct response code.

In order to generate the response codes required by SNMPv2, each of these requires a message to the subagent and a corresponding response.

As an optimization of the high frequency case, where all the varbinds in a request will be handled by a single subagent, it has been suggested that the protocol should employ a single exchange, rather than a multi-phase transaction.

A final nasty aspect of `set` processing, which is really a general SNMP issue, is whether it is permissible to concurrently process `set` operations for different naming scopes. It is fairly clear that concurrent processing of `set` operations within a single naming scope would be risky, since there is no reasonable way to predict the side-effects of an operation. Whether an operation (like reset) must be assumed to potentially affect multiple naming scopes requires additional discussion.

Another area with operational implications is the handling of access control. The current consensus is that all access control handling is the responsibility of the master agent. It should be noted that some MIB modules, such as the RMON `alarm` group, may require knowledge of a system's access control policy.

The final area of operational issues has been the handling of `informs`. Part of the problem is coming to an understanding of what `informs` are. Even this has been subject to considerable debate within the SNMP community. There may be a requirement for subagents to subscribe to, and to issue, `informs`.

Registration Requirements

> *How much power and flexibility are needed for subagents to identify their area(s) of responsibility?*

The registration process (and, as a result, the details of operation dispatch) has been the subject of extensive discussion. The following requirements have emerged, based on the capabilities of existing solutions:

- support for registration of entire groups;
- support for registration of individual objects;
- support for registration of whole tables;
- support for registration of table rows (different columns, index value constant);
- support for registration of table slices (different index values, set of columns constant); and,
- support for registration of table pages (constant set of columns, subset of indexes constant).

Some existing protocols handle only some of these; others can accommodate all of them, with varying degrees of efficiency in the registration process. All of these can be described in terms of subtree registration. Adding a flag to identify single-instance registration can significantly optimize master agent operation dispatch. The registration of individual table rows, slices, and pages could be done more efficiently if the subtree were represented as a limited regular expression for a family of subtrees, rather than requiring a separate registration for each column of the table in question. The tradeoff is between the number of transactions required to perform registration and the complexity of the syntax representing a registration.

Additional registration issues include:

- handling multiple naming scopes;
- handling priority; and,
- handling overlapping requests.

The only difficulty in supporting multiple naming scopes has been to reach agreement on the representation of a naming scope identifier. From an implementation perspective, the only operations needed on this type are assignment and comparison for equality.

Discussions of handling registration priority led to the conclusion that registration priority is needed for handling redundant and fault-tolerant configurations, that the complexity is equivalent to that of handling registration-time based precedence, and that the notion is needed within an implementation to handle collisions anyway.

Registration overlap occurs when an OID is a subtree of two registrations at the same priority. The most useful way to handle this case is to treat the longer registration as having the better priority. This conclusion is the result of implementation experience with protocols using different resolution strategies. The deciding case is where one subagent is responsible for handling requests for the creation of arbitrary new rows, and the new rows, once created, will be the responsibility of separate subagents. (For example, consider application processes forked off as a result of create operations – if the shorter subtree registration took priority, the table entries for the forked processes would not be manageable.)

Inter-subagent Requirements

> *What support for information access between subagents is needed (e.g., can one subagent search another's tables)?*

The requirements for inter-subagent communications include:

- index reservation and coordination;
- retrieval of arbitrary MIB variables by subagents; and,
- special-case MIB variables, such as `sysUpTime`.

Any such operations will have to be qualified by a naming scope. Allowing subagents to talk to each other, even indirectly, raises issues of access control. Ideally, these issues could be resolved in a manner consistent with an emerging SNMPv3 access control framework.

The requirement for index reservation and coordination has found vocal and convincing support. The problem, simply stated, is that when different subagents implement different rows of a table, there is a need for a coherent index reservation policy. For some indexes, this policy is inherent in the index semantics, such as the use of a process ID. For others, such as an `ifIndex`, more sophisticated infrastructure is needed to ensure consistency of index values and references.

A key requirement is that the protocol state machine for index reservation and query must be able to function independently of the state machine for processing `set` and `get` operations, since row creation can happen due to local action as well as due to management request.

Index reservation is distinct from the registration protocol for several of reasons. The most important one is that the lifetime of a reservation may be far greater than that of a registration. For example, the reservation may be a result of system configuration or provisioning, determined long before a specific subagent is activated. Another reason is that the same reservation may be of interest to a number of entities, as in the case of `ifIndex`.

There appears to be general agreement that the full range of index syntaxes should be supported by the solution, although merely handling `ifIndex` alone would have significant value.

Subagent access to another subagent's MIB variables is not fully settled as a requirement. If this is accepted as a requirement, it will be necessary to define the access protocol so as to avoid deadlock situations with respect to SNMP-initiated operations. A special case of this is support for MIB modules like the RMON `alarm` group, which also require knowledge of access control information.

Visibility Requirements

> *What information should appear in a subagent MIB to meet the requirements of managers that need to know the internal structure of the managed system?*

Although a manager will generally not be interested in what specific constellation of subagents is used to instrument a system, there are cases, especially in debugging scenarios, where a manager may need to find out which subagent is responsible for what. MIB modules have been defined for existing protocols. They are remarkably similar; most of the discussion, product deployment, and research experience leads to the conclusion that these MIB modules are probably overkill, recording more information than is actually useful. Trimming the excess from these MIB modules remains to be done.

Specification Requirements

> *What abstract syntax notation should be used for the protocol definition, and which set of encoding rules should be used?*

Two major debates have occurred in the area of specification requirements. The first is the choice of protocol definition language. Some subagent protocols have been defined using ASN.1; others have used ad hoc notations. The AgentX effort is, at this time, basing its work on a document which uses an ad hoc notation. The issues are ones of clarity and rigor.

The second debate, quite distinct from the choice of specification language, is the choice of transfer syntax. The choices are between the formally defined encoding rules, such as BER, PER, DER, or XDR, and various ad hoc schemes. The AgentX effort is, at this time, basing its work on a document which uses an ad hoc encoding scheme. At issue are clarity, implementation complexity, and verifying the correctness of implementations, as well as performance.

In Conclusion

By now it should be clear that the overall requirements and many design tradeoffs are well understood. With the breadth and depth of expertise in the working group, coming to conclusion on these fairly clear-cut issues should not be hard.

The difficult part of the AgentX working group's task will be to reach agreement where there's no clear leader among the technical alternatives, or where the group is not able to agree on metrics for evaluating alternatives. Here we must stick to principles of clarity and simplicity, tempered by practical experience and sensitivity to our customers' needs.

eSNMP, An Extensible SNMP Agent

Mike Daniele
Digital Equipment Corporation

eSNMP is the native extensible agent and associated framework for Digital UNIX (formerly known as DEC ALPHA OSF/1). It was developed by the operating system group, to provide a mechanism for ISVs and customers to share in the SNMP on our platform. It is not a commercially-available drop-in solution for other environments.

Overview

eSNMP uses the common paradigm of a single *master agent* and multiple independent *subagents*. The protocol used between them is a variant of DPIv2, modified as necessary to support our required functionality.

The following diagram illustrates the major components of the framework:

```
          Master Agent                              Subagents
+------------------------------+            +------------------------------+
| +--------------------------+ |          +------------------------------+ |
| |         Registry         | |          |                +-----------+ | |
| | o          o       o     | |          |                | Method    | | |
| | | \       /|\     / \    | |          |  +-----------+ | Routines  | | |
| | o  o     o o o   o   o   | |          |  |           | |           | | |
| |    |       |             | |          |  |    API    | | +-------+ | | |
| |   / \    |-|-|-|-|       | |          |  |   client  | | |Object | | | |
| |  o   o   o o o o o       | |          |  |           | | |Type   | | | |
| |                          | |          |  +-----------+ +-|       |-+ | |
| | +----------++---------+  | |          |                  |Table  |   | |
| | |Dispatcher||Registrar|  | |          |=================|       |===| |
| | +----------++---------+  | |          | |  API routines +-------+  | |
| +--------------------------+ |          | |                          | |
| +--------+ +---------------+ |          | +--------------------------+ |
| | SNMP   | | eSNMP/DPI     | |          | |    eSNMP/DPI  Engine     | |
| | Engine | | Engine        | |          | +--------------------------+ |
| +--------+ +---------------+ |          | +--------------------------+ |
| +--------+ +---------------+ |          | |         AF\_UNIX          | | |
| |  UDP   | |   AF\_UNIX     | | |         | +--------------------------+ |-+
| +--------+ +---------------+ |          +------------------------------+
+------------------------------+
                                                    | ^              ^ |
   /\   ||     ^ |     | ^                          | |              | |
   ||   ||     | |     | |       {Control}          | |              | |
   ||   ||     | |     | +-OPEN,REG,UNREG,TRAP,PING-+ |              | |
  SNMP  ||     | |     +--------RESP, CLOSE-----------+              | |
Request ||     | |                                                   | |
   ||   ||     | |                 {Data}                            | |
   ||   ||     | +------GET,NEXT,BULK,SET,COMMIT,UNDO,CLEANUP-----+ |
   ||   ||     +-------------------RESP------------------------------+
   ||  SNMP
   ||  Response
   ||   ||
        \/
```

eSNMP/DPI runs over UNIX domain sockets, so both sides need to support this transport and know where to rendezvous.

The master agent's dispatcher and registrar are the algorithms by which it accepts registrations, associates requested MIB variables with a particular subagent, and communicates with those subagents. The object-type table in the subagent is emitted by a MIB compiler back-end tool (we used mosy and snmpi from ISODE, with some modifications).

SNMP is transmitted between the master agent and the management application **only**. Communication between the master agent and subagents is via eSNMP/DPI **only**. DPI is not encoded using the BER, and we kept the same general encoding and packet formats. As a result, both sides contain an engine for eSNMP/DPI handling. Each packet has some explicit (header) format and some variable length fields. The variable length fields (including OIDs) are encoded as null-terminated ASCII strings.

Each eSNMP/DPI packet starts with a fixed header:

```
.-----------------------------------------------------.
| Layout of eSNMP packet header.  Present in all packets |
+--------+------+--------------------------------------+
| OFFSET | SIZE | FIELD                                |
+--------+------+--------------------------------------+
| 0      |  2   | packet length                        |
+--------+------+--------------------------------------+
| 2      |  2   | packet ID                            |
+--------+------+--------------------------------------+
| 4      |  1   | protocol major version               |
+--------+------+--------------------------------------+
| 5      |  1   | protocol minor version               |
+--------+------+--------------------------------------+
| 6      |  1   | packet-flags                         |
+--------+------+--------------------------------------+
| 7      |  1   | packet type                          |
+========+======+======================================+
```

This permits code to assign a header pointer to the start of a suitably aligned receive buffer. The remainder of the packets are dependent on packet type. We opted for native-byte ordering in the headers (which is possible since eSNMP is restricted to a single host system), but eSNMP uses network-byte ordering of data lengths and values. This doesn't reduce performance as these fields fall on arbitrary byte boundaries and must be parsed byte-by-byte. (We are also hedging against future off-host subagents.)

The API

It's clear from the diagram above that eSNMP operations are more complex than traditional SNMP operations; however, most of the eSNMP complexity is hidden behind an API (represented by the double horizontal line above). Everything below that line is contained in a shareable runtime library provided by the operating system.

To build a subagent the developer must:

- compile the relevant MIB specs;
- write the API client code that handles the Control part of the eSNMP/DPI protocol;
- write its objects' method routines; and,
- link against the shareable library.

The API is quite simple:

- `esnmp_init()` sets up a logical connection with the master agent via the OPEN PDU, and returns a socket descriptor;
- `esnmp_register()` registers the subagent's hierarchies of managed objects with the master agent, via REG PDUs; and,
- `esnmp_poll()` is called whenever data is pending on the socket. Its main job is to handle all of the Data PDUs that have arrived. It locates the correct object-type in the table, checks access, and calls the object-type's method routine. For `get-next` and `get-bulk` requests it traverses the locally-held objects, and so on. Data or error conditions returned by the method routine are passed up as an eSNMP/DPI RESPONSE and sent to the master agent.

This is the basis of the API (some minor calls, e.g., those dealing with `trap` generation, are omitted). The subagent developer must check API return status codes to discover possible loss of connection and restart the protocol with `esnmp_init()`. The important point to make is that it is the shareable library code, **not** the subagent developer's code, that acts as the eSNMP/DPI protocol peer and communicates with the master agent. Such a framework affords the subagent developer a way to write method routines and export them into a managed system, which is relatively low in both programming complexity and runtime overhead. In our environment, this is more appropriate than packet-multiplexing approaches, because it reduces both runtime overhead and configuration issues, and does not reduce transparency to management stations. Finally, it's worth noting that this type of framework is common to most commercially available extensible SNMP products.

Design Tenets

These are the major requirements and design goals we had for eSNMP:

- no changes are necessary in a management application in order to access MIB variables instrumented in subagents;
- subagents are developed completely independently, and overlapping registration is permitted;

- subagents are completely decoupled (not started or stopped in step with the master agent).
- subagents participate in eSNMP without impeding other, potentially more important, functions (e.g.,, on our platform the *gated* routing daemon also acts as an eSNMP subagent);
- communication between a master agent and its subagents occurs using local transport only (not TCP), as we have no requirement for remote subagents, and didn't want to introduce new security problems;
- registration is completely dynamic (subagents can add or subtract from the MIB hierarchy at will);
- the master agent is a repository of OIDs, NOT object-types, and has no knowledge of any MIB specification;
- only subagents know how to instantiate its objects, and how to perform `get`, `get-next`, `get-bulk`, and `set` operations on them;
- table sharing or instance-level registration must be allowed;
- the master agent performs authentication of received requests.

Connection Establishment/Maintenance

The master agent (snmpd) starts at system boot and binds a socket to `/var/esnmpd`. When requested, the runtime library code in the subagent sends an OPEN PDU to this endpoint, establishing a logical connection:

OFFSET	SIZE	FIELD
... header ...		
8	2	timeout
10	2	max-vbs = 0
12	var	subagent description (null-terminated)

The timeout field permits a subagent to indicate longer than normal latency. The `max-vbs` field is unused. The description identifies the subagent, and must be unique among connected subagents (it is typically the fully qualified path of the command executing the subagent). There is no on-disk configuration of subagents. This protocol exchange is the only way the master agent is made aware of available subagents.

Once so connected, a subagent may send any of the control PDUs, specifically it may register. Data requests may be sent to a subagent whenever the dispatching policy chooses any of its registered subtrees.

If the master agent receives a CLOSE packet from a subagent, or detects from the underlying transport that the subagent cannot receive data, the master agent unregisters all of its subtrees and destroys the logical connection.

The PING PDU is available for subagents to verify the status of the master agent.

Registration

The registration packet looks like:

OFFSET	SIZE	FIELD
... header ...		
8	2	priority
10	2	timeout
12	2	view-sel
14	2	bulk-sel
16	var	subtree OID (null-terminated, length L)
16+L	var	sub-tree description (null-terminated)

The unit of registration is a single OID. By registering an OID, the subagent indicates it will handle management requests for objects within the subtree named by that OID. A subagent may register any OID at any priority, there is no policy to limit subagents. The OID may represent an entire MIB, a MIB group, a table entry, a partial instance, or a full instance. This is left entirely to the discretion of the subagent developer.

The master agent handles overlapping registration by splitting affected subtrees into smaller ranges that are either exact duplicates, or no longer overlapping. For instance, suppose component agent A registers `ip` (1.3.6.1.2.1.4) and subsequently component agent B registers `ipNetToMediaTable` (1.3.6.1.2.1.4.22). The master agent splits the registry into 3 OID ranges:

```
(1.3.6.1.2.1.4    upto 1.3.6.1.2.1.4.22)  subagent A
(1.3.6.1.2.1.4.22 upto 1.3.6.1.2.1.4.23)  subagents A, B
(1.3.6.1.2.1.4.23 upto 1.3.6.1.2.1.5)     subagent A
```

Here "upto" means "up to but not including". If component agent C now registers `mib-2` (1.3.6.1.2.1) this is split into OID ranges, resulting in:

```
(1.3.6.1.2.1      upto 1.3.6.1.2.1.4)     subagent C
(1.3.6.1.2.1.4    upto 1.3.6.1.2.1.4.22)  subagents A, C
(1.3.6.1.2.1.4.22 upto 1.3.6.1.2.1.4.23)  subagents A, B, C
(1.3.6.1.2.1.4.23 upto 1.3.6.1.2.1.5)     subagents A, C
(1.3.6.1.2.5      upto 1.3.6.1.2.2)       subagent  C
```

The policy we used for overlapping registrations is that only 1 of them is active at a given time. That's the one with the highest priority, or, in the case of a tie, the one most recently registered. Hence, in the example above, assuming all subagents used the default priority, subagent C would have the active ranges.

Subsequent UNREGISTER packets of course cause inactive ranges to bubble up. If subagent C unregistered `mib-2`, then B would own the active range from 1.3.6.1.2.1.4.22 upto 1.3.6.1.2.1.4.23, and A would own two ranges, 1.3.6.1.2.1.4 upto 1.3.6.1.2.1.4.22, and 1.3.6.1.2.1.4.23 upto 1.3.6.1.2.1.5.

We didn't include any way for a subagent to be aware of what ranges it is active for, mainly because we couldn't think of what subagent code could do about it. When signaled, the master agent dumps the current registry to a file. This enables local operator intervention when a configuration is unsatisfactory.

Priorities are viewed as mechanism used by cooperating subagents, to provide precedence and backup coordination without explicitly having to check for the other's existence.

Dispatching

When the master agent receives an SNMP request, each varbind in the request is processed in turn according to this general algorithm:

1. search the list of ACTIVE subtrees and find the lexicographically first candidate;
2. associate this varbind with the subagent that registered this active subtree;
3. assign a timeout value with each packet that is the maximum of all the timeout values that were registered with the subtrees found in the first step;
4. encode a packet for each involved subagent, containing all of its associated varbinds, and only those varbinds;
5. send the packets and start a timer;
6. marshall the responses and if a subagent times out mark those varbinds with `genErr`;
7. when processing `get-next` and `get-bulk` requests, for all varbinds returned with `endofMibView`, start over with the first step, but don't start a new timer; and,
8. when all varbinds have been returned with data, returned in error, or timed out, formulate an appropriate SNMP response message and send it to the requesting application.

Note that in the first step only the active subtree (ranges) are searched. We chose this dispatching policy so that between registration changes, a MIB variable's instrumentation cannot change.

Data Request/Response

Data request and response packets contain variable length varbind sections:

```
.---------------------------------------------------------.
|                 Layout of a varbind section             |
+--------+------+-----------------------------------------+
| OFFSET | SIZE | FIELD                                   |
+--------+------+-----------------------------------------+
...
+--------+------+-----------------------------------------+
| L      |  1   | varbind-flags                           |
+--------+------+-----------------------------------------+
| L+1    | var  | Starting OID (null-terminated string)   |
+--------+------+-----------------------------------------+
| L+M+2  | var  | Ending OID   (null-terminated string)   |
+========+======+=========================================+
The following only on SET and RESPONSE to GET, GETNEXT, and
GETBULK:
+--------+------+-----------------------------------------+
| L+M+N+3|  1   | varbind variable type                   |
+--------+------+-----------------------------------------+
| L+M+N+5|  2   | varbind data length (network-byte order) |
+--------+------+-----------------------------------------+
| L+M+N+6| var  | varbind data value                      |
+--------+------+-----------------------------------------+
...
+---------------------------------------------------------+
| Notes:   L - current position in message                |
|          M - strlen(Starting OID), without terminator   |
|          N - strlen(Ending OID),   without terminator   |
`---------------------------------------------------------'
```

eSNMP supports both SNMPv1 and SNMPv2 SMI. GET, GETNEXT, and GETBULK packets use the following format:

```
+--------+------+-----------------------------------------+
| OFFSET | SIZE | FIELD                                   |
+--------+------+-----------------------------------------+
... header ...
+========+======+=========================================+
| 8      |  4   | non-repeaters                           |
+--------+------+-----------------------------------------+
| 12     |  4   | max-repetitions                         |
+--------+------+-----------------------------------------+
| 16     |  4   | sec-len                                 |
+--------+------+-----------------------------------------+
| 20     | var  | security data (octet sec-len bytes long) |
+--------+------+-----------------------------------------+
| 20+M   | var  | varbinds, (see above for layout)        |
+--------+------+-----------------------------------------+
|                                                         |
| Notes:  M is size of the security data which is value of |
| sec-len.                                                |
`---------------------------------------------------------'
```

For these PDUs, the `packet-flags` field of the header contains the SNMP version of the original request. `non-repeaters` and `max-repetitions` are both 0 for GET and GETNEXT packets. For GETBULK packets, `non-repeaters` may be initially adjusted in light of the varbinds sent to each agent, and `max-repetitions` may be adjusted subsequently if `endofMibView` is returned for any varbind and more GETBULK packets are issued.

Currently the security field is always empty, although there is a separate `sec-len` field in anticipation of passing formatted data for security/naming scope information.

For a GET packet, the starting OID is the requested OID. It must be within the range of a subtree registered by the subagent. For the GETNEXT and GETBULK packets, the starting OID is either the requested OID (if that falls within a registered subtree), or the OID of the registered subtree that is lexi-next after the requested OID. In this latter case the `varbind-flags` field is set to indicate this.

The ending OID is empty for GET packets, and on GETNEXT or GETBULK packets it is set to the end (noninclusive) of the range of OIDs that this subagent may respond for this varbind. (Due to overlaps and splitting by the master agent, this search range does not necessarily include the entire registered subtree; however, the search range will NEVER span multiple registered subtrees.)

When the subagent receives these requests, it searches its registered subtrees for the one containing starting OID, and then through a linkage to the object type table finds the correct object type. It then calls the indicated method routine, passing the value of starting OID and an indication of the request type (along with other information specific to the API).

If the `varbinds-flag` bit is set, the subagent modifies this procedure slightly to perform a `get` operation, regardless of the actual request. This is how possible instance level registrations are handled, since the master agent is unaware of what the registered OIDs mean. If the GET processing fails, the subagent proceeds with normal NEXT/BULK processing if that was the original request. (Recall that the subagent developer is unaware of this processing.)

The subagent processes all varbinds in the request packet in this manner with the inclusion of a step that checks the `packet-flags` field of the header for the SNMP version of the request before calling any method routine. Object types in the table with SNMPv2-only syntax are ignored on SNMPv1 requests.

The data or error information for each varbind is eSNMP/DPI encoded and a RESPONSE packet sent back to the master agent. In these packets the starting OID and associated data are the varbind to return to the original requester. The varbinds in this response packet are returned in the order they were sent in the request packet.

Set Processing

eSNMP supports "as if simultaneous" set operations that span subagents by using the SET, COMMIT, and UNDO PDUs found in DPI. We added a CLEANUP PDU to indicate the end of processing to allow resource release. We also added an optimization for the typical case of only a single subagent handling all the varbinds for the `set` operation.

The SET, COMMIT, UNDO, and CLEANUP packets use the same format as the GET* packets, the only difference is that the varbinds sections contains data in a SET packet.

The dispatch processing at the master agent is the same as for GET* requests except that another bit in the header's `packet-flags` fields is set if all requested varbinds are dispatched to the same subagent. In this case, the subagent performs the SET, COMMIT, potentially UNDO, and CLEANUP phases itself, and returns a single RESPONSE. Otherwise, the master agent must itself initiate the SET, COMMIT, optionally the UNDO, and the CLEANUP phases by sending these packets to each involved subagent.

Note that no check-consistency PDU needs to be issued by the master agent, since the subagent receives ALL varbinds destined for it in any request PDU. Consistency checking (and other very useful aids to help subagent developers perform `set`-related operations) then become an API issue.

Responses

All eSNMP/DPI PDUs require a RESPONSE in return except CLOSE, CLEANUP, and TRAP. All RESPONSE packets are formatted as:

```
+--------+------+----------------------------------+
| OFFSET | SIZE | FIELD                            |
+--------+------+----------------------------------+
... header ...
+========+======+==================================+
| 8      |  4   | error-code                       |
+--------+------+----------------------------------+
| 12     |  4   | error-index                      |
+--------+------+----------------------------------+
| 16     | var  | additional data (see discussion) |
+--------+------+----------------------------------+
```

Responses to requests use SNMPv2 compatible error codes, and use the `error-index` field. The master agent maps the code to SNMPv1 if required, and maps the error index for all requested varbinds (not just those sent to this subagent.) Successful response varbinds can contain `endofMibView`, indicating the master agent needs to roll-over to the next registered subtree. SNMPv2 support in the master agent is being added at the time of this writing.

Responses to the OPEN PDU contain a UNIX-style timestamp, from which an API routine can calculate `sysUpTime`-like timetick values. Responses to other PDUs contain no additional data.

Appraisal With Respect to AgentX

eSNMP did not have the same functional requirements as have emerged for AgentX. Some aspects of eSNMP are discussed in this light.

eSNMP permits "table sharing by brute force". Subagent A can register `ifIndex.1`, `ifDescr.1`, etc., and subagent B can register `ifIndex.2`, `ifDescr.2`, and so on. It works, but AgentX needs to do better in 2 areas. First, the registration syntax should probably allow registering an entire row in one operation. Second, a bigger problem is that of index collision in shared tables. AgentX will need to provide mechanisms for subagents to reserve indexes, so that subagents that share tables can be truly independent, relying only on AgentX services (as opposed to allocating indexes amongst themselves).

AgentX may contain explicit support for augmenting tables, and may provide an index subscription service, so subagents may learn about existing rows in tables of interest, and be notified of changes. Since eSNMP does not have special registration syntax for sharing tables, and its dispatching policy limits each varbind to at most 1 subagent, it is impossible to create rows in shared tables. This is unacceptable for AgentX. In contrast, eSNMP exists on a UNIX platform in which network interfaces are rationalized within the kernel, and the operating system implements MIB support for MIB-II, media-specific MIB modules, and Host Resources MIB. This removed any need to share `ifTable` between subagents, and so removed most of the impetus to provide more explicit support of table sharing.

Finally, eSNMP does not carry context/naming scope information, nor does it carry access control information (community names, views). AgentX is likely to require support for naming scope at least.

Conclusions

From the perspective of providing an agent that represents the entire managed node correctly and with good performance, this design is a success. SNMP request/response round trip times are increased by very small values, compared to the monolithic agent, typically less than .002 seconds on a 150MHz system. No changes in snmpd's configuration file or startup mechanism are required, and the statistics in the system group are correct!

From the perspective of providing an application development environment, feedback has reenforced this division of labor and general framework. Issues typically involve the clarity of the documentation, particularly at the method routine interface.

My personal belief is that subagent API/toolkit development is best left to the vendors who specialize in that area, and that differentiation in this area is not a "Bad Thing". The challenge before the AgentX working group is to provide a standard subagent protocol that is functionally equivalent to, though not binary compatible with, currently deployed protocols. This would enable the wide variety of independently-developed subagents and master agents to interoperate, which is the ultimate goal of our efforts.

I look forward to the day I can support AgentX in our operating system!

An Alternative Perspective on Agent Extensibility

Dave Bridgham
Epilogue Technology Corporation

It wasn't long after the first SNMP agent was fielded that someone asked if there was a way to have several cooperating agents appear to be a single agent to the management station. This person no doubt had several things in a single box to manage with SNMP; things that were separate development projects and so naturally the development of the management code should also be separate projects. So the question became:

> "How do I get my management information into the SNMP agent that another group provided?"

And from someone who was producing the SNMP agent:

> "How do I allow all these separate projects (separate processes, cards, devices, whatever) get their management information to me?"

These are natural enough questions given the circumstances. Unfortunately they are not the correct questions. The questions, as asked, specify the answer. If you back up a little and look at the larger management picture, rather than concentrating solely on the agent, then you get a different question and possibly different answers. The question we should be asking is:

> "How do we get management information from a variety of sources on a host, back to the manager?"

This article looks at using SNMP itself as the means for doing this communication. After all, moving management information around is what SNMP was designed to do. The technique is called SNMP proxy. For proxy to extend agents requires no new protocol design though a proxy registration MIB would be helpful. It would allow for smoother, automatic management station configuration and dynamic reconfiguration of agents.

Many people I've talked to seem astonished even to hear "proxy" and "agent extensibility" in the same sentence. Apparently, they think the two are completely unrelated. So let's look at how something as simple as proxy can help.

How Proxy Works

I assume most readers are familiar with SNMP proxy, at least conceptually. But the term can be and has been used in quite a few situations with SNMP so I'll briefly go over how I'm using it here. Also, I'll hopefully clarify a few other technical terms that I'll be using throughout the remainder of the article.

As I use the term in this article, *proxy* is essentially what SNMPv2 classic meant by proxy. An SNMP packet arrives at a well known service location (such as UDP port 161) where it is received by the SNMP *proxy agent*. This agent determines, from the community string in the packet, that the packet is intended for a different SNMP agent called the *proxy target*. The proxy agent relays the SNMP packet, selecting a community string for the proxy target along with a new request-ID. The proxy agent also keeps a bit of state around, referenced by the new request-ID, containing the original requester's addressing information, community string, and request-ID so when the proxy target responds to the proxy agent, it is able to relay the response back to the original requester.

How Proxy Provides Agent Extensibility

Now let's look at a few common situations that want an extensible agent, and see how proxy technology could be used. What we'll find is that the agent end is just proxy as I've already described above. All the work is in the management station in putting the information together and displaying it in a useful manner to the human manager. That's okay though. After all, that's what network management stations are supposed to do: collect and organize network information, and display it in a useful manner.

The most common use of extensible agents is the splitting of MIB modules over several agents. Maybe it's different cards in a backplane, each with its own processor. You don't want to burn the MIB module into a single processor, thereby limiting flexibility in adding new MIB support as you come out with new cards for your system. Maybe it's different processes on a UNIX system, written in some cases by different companies.

On the agent end, just run one agent as the proxy agent and management stations talk to all the other agents in the system through it. When the management station first contacts the agents, it reads `sysObjectID` and other similar information from each agent to discover what that agent does. Remember, each proxy target is an SNMP agent and has its own `system` group. Then, when asked to show some aspect of the managed device, the management station directs its requests to the right proxy target by selecting the right community string and retrieves information as usual with SNMP.

What if the information is spread across multiple proxy targets? This isn't likely to be a common situation, as any information that's likely to be grouped on the manager's screen is also likely to be grouped into a single proxy target, but it's easily handled. The manager simply makes requests of each proxy target, collects all the information available from the lot of them, and displays the results in a unified manner.

Another common situation is the splitting of an SNMP table across several agents, most often the interface table, e.g., consider a backplane system where each interface card wants to run its own entry in the interfaces table and its own `transmission` group MIB module. Each interface card runs its own proxy target with those MIBs. Obviously, each interface needs to pick values for `ifIndex`, and they're very likely going to all pick 1. That's not a problem, since the management station knows they're different interfaces because it talked to each proxy target separately to retrieve the entire interface table. If the management station were to simply display the values of `ifIndex`, it would be rather confusing. So, instead of displaying the raw information, a very simple transformation of the data gives each interface a unique number, or even a name, which is likely to be more palatable to most human users.

The final common example we'll examine here is multiple instances of the same MIB module. You have a MIB module which instruments some part of your system and one day you get a second instance of that part of your system and so need a second instance of that MIB. This may be a duplicate board plugged into a backplane, or some instrumented UNIX application that was run twice. With proxy, operation on the agent's end is just like any other extended agent: same MIB module, different MIB module, it just doesn't matter.

The management station has a couple of choices, though. It could show the two instances of the same MIB module as if there were two devices, or it could try to merge the two into one, like merging multiple `ifTables` into a single large `ifTable`. The choice comes down to which provides the clearest picture for the human user, and the choice can be left to the management station rather than trying to pre-decide in the agent.

Since each of these proxy targets is an SNMP agent in its own right, why bother with proxy at all? Why not have the mangers just talk directly to the targets? There are two main reasons: security and configuration.

Security

With the most recent SNMPv2 RFCs, SNMP lost its security capabilities, but the security effort has not ended and I think most of us in the network management community believe that security of some sort for SNMP is coming. The hardest part of security and, in particular, it seems, network security, is not securing the data itself; it's the administration of it all. Proxies (and other agent extension methods) let you put the security configuration of an extended SNMP agent in the proxy agent only. This can be an enormous reduction in security configuration, as the proxy targets then only need security sufficient to the environment they live in. In other words, within an embedded system probably no additional security is needed at all, and within a multiuser system, the system itself has security capabilities suitable for an operating system, which are much simpler to use and administer than those needed across a network.

Even though it might be possible to talk directly to proxy targets, rather than through the proxy agent, concentration of security configuration makes proxy still useful.

Configuration: Registration and Discovery

While SNMP proxy can provide agent extensibility with no protocol extensions at all, it would be somewhat unusable from a configuration standpoint. It's just not feasible to expect network managers to configure each management station with access information (communities) for every proxy target as well as each proxy agent. Also, if we have dynamic proxy targets then we'll need a standard way for proxy targets to register themselves with a proxy agent.

One possibility is to design a proxy registration protocol, but a simpler solution is simply to write a registration MIB module to be implemented by the proxy agent. To register, proxy targets write themselves into the table in the proxy agent. The information in this MIB need only be contact information for the proxy target: the community string to use and the transport type and address. No information about the proxy target itself is needed, because management stations can query the proxy target directly for anything they want to know. This MIB module also then provides the means whereby a management station can discover all the proxy targets it might want to talk to.

Actually, one additional piece of information would be useful in this proxy registration MIB. Some proxy targets are agent extensions, and others are just other agents being accessed through a proxy. For a management station which is trying to do a good job of displaying an extended agent as a single device, it must distinguish the two cases.

Another issue with this registration table is how it gets garbage collected. If a proxy target goes away willingly, it can obviously just remove itself from the registration table. But what about otherwise? Since a proxy agent needs to keep state around to eventually relay the response, it also needs to timeout this state in the event that the proxy target disappears. A few of occurrences of this, and the proxy agent could mark this proxy target in the proxy table as having a problem. At that point the proxy agent could actively interrogate the proxy target, or it could wait for a few more timeouts and then remove the entry from its table.

Why You Haven't Seen Proxy as the Agent Extensibility Solution

Listen in on MIB design sessions and you find that MIB designers do their work as if humans will be operating on the MIB variables directly. The variables are crafted to provide the highest level of semantic content possible. They design MIBs as if the MIB were the user interface. That's because it is! Most management stations do little more than provide a way for a human to interact with a remote MIB module. The management station provides the screen, keyboard, and mouse; the MIB module provides the the user interface.

You could operate a proxy system this way, but it would be painful. Agent extension through proxy assumes that MIBs are a machine interface and that something, a network management station, sits between this machine interface and the human to makes things look nice, and perhaps even does a little analysis on the data before displaying it. Since this is not the case, agent vendors have little choice but to continue, as best they can, to put the user interface into their agents.

In Conclusion

Obviously, proxy technology provides a simple and clean means to extend agents. Agent extensions through proxy don't require changing the SNMP protocol, re-designing existing MIBs with agent extensibility in mind, or a grand design effort to produce a new, standard protocol.

If the only part of the system you control is an SNMP agent, then you have to do whatever it is you're trying to do within that agent. Stepping back and looking at a larger network management system lets you ask,

> "Where's the best place to aggregate information from a variety of sources?"

You may get a different answer than when you restrict your vision to just the agent.

The SNMP Framework

Keith McCloghrie
Cisco Systems

In the last issue, this column examined some of the changes to SNMPv2 incorporated in RFCs 1901-1908. Seven of these documents were published as approved Draft Standards, with the other one (RFC 1901) published as an experimental RFC. Specifically, in the last issue, we examined the changes to the SNMPv2c administrative framework, and to the SNMPv2 protocol and transport mappings. In this issue, we'll detail the changes to the rules for defining MIB modules.

The rules for defining MIB modules are generally referred to as the Structure of Management Information (the SMI). There are three documents which specify these rules:

- the SMI itself (RFC 1902);
- the Textual Conventions (RFC 1903); and,
- the Conformance Statements (RFC 1904).

Changes to the SMI

RFC 1902 defines a number of changes from RFC 1442:

- Three changes to the allowed data types: `NsapAddress` is removed since its usage turned out to be unnecessary; `UInteger32` is replaced by `Unsigned32`, and `BIT STRING` is replaced by the `BITS` construct. Both of the latter two changes promote better backward-compatibility with SNMPv1, since `Unsigned32` is encoded on the wire in an identical manner to the `Gauge32` data type, and the `BITS` construct is encoded on the wire as an `OCTET STRING`. Thus, a SNMPv2 to SNMPv1 MIB conversion utility can produce compatible (but semantically-poorer) definitions for `Unsigned32` and `BITS` in an SNMPv1 MIB format. These new data type definitions should not cause problems with existing SNMPv2 MIBs, since the usage of RFC 1442's new data types was discouraged while the SNMPv2 SMI was still at Proposed Standard status.
- For object descriptor names and enumerated labels: the rules on disallowing hyphens are tightened, and the length of these descriptors are recommended to be 32 characters or less.
- Usage of the IMPLIED keyword is restricted so that it can only be used for the last object in an INDEX clause, in order to avoid ambiguity.
- `accessible-for-notify` is defined as a new keyword for the ACCESS clause of the OBJECT-TYPE macro; the meaning of this keyword is nearly identical to `not-accessible` except that it is allowed to be referenced by the OBJECTS clause of a NOTIFICATION-TYPE macro, whereas a `not-accessible` object is not.
- `zeroDotZero` is defined using an OBJECT IDENTITY macro as the value `0.0`.
- The `deprecated` keyword is allowed in the STATUS clause of an OBJECT-IDENTITY macro.
- `read-only` auxiliary objects are allowed when converting a SNMPv1 MIB to an SNMPv2 MIB.
- The next to last sub-identifier of any newly-defined notifications is required to be zero in order to be compatible with the rules for proxy conversion of SNMPv1 traps into SNMPv2 traps.

RFC 1902 also expands and clarifies the text in a number of places, including:

- An Appendix is added to document existing practice in the usage of ASN.1 rules for sub-typing, including allowing the sub-typing of `Integer32`.
- The maximum length of an `OCTET STRING` is limited to 65535 octets.
- The meaning of `deprecated` is clarified as a definition which is obsolete, but may still be supported for backward-compatibility.
- The REVISION clause of the MODULE-IDENTITY macro is clarified as being required for the initial revision.
- An Appendix is added to explain the UTC time format (used in the LAST-UPDATED and other clauses).

Changes to the Textual Conventions

RFC 1903 defines a number of changes from RFC 1443:

- Usage of the new `BITS` construct is allowed in new Textual Conventions.
- The behavior of a `TestAndIncr` variable at agent re-initialization is specified; the variable must either be incremented from the value it held prior to the re-initialization, or must be set to a unpredictable value.

- The `InstancePointer` TC is obsoleted and replaced with two new TCs: `VariablePointer`, for pointing to an object instance; and `RowPointer`, for pointing to a conceptual row.
- For the `RowStatus` TC, the potential error conditions are clarified when a value is written to a column other than the status column of a non-existent row; and the definition of a specific `RowStatus` object in a MIB module is allowed to override the age-out timer for non-active rows.
- Three new TCs are added: `StorageType`, `TDomain` and `TAddress` (all of these were previously defined in the now historic RFC 1447).
- The DISPLAY-HINTS clause is extended to define the implied position of a decimal point when rendering a decimal value.
- The meaning of various NVT-ASCII sequences (e.g., "CR LF") in a `DisplayString` is clarified.

Changes to the Conformance Statements

RFC 1904 defines a number of changes from RFC 1444:

- The NOTIFICATION-GROUP macro is added to allow MODULE-COMPLIANCE statements to refer to groups of notifications, and to allow AGENT-CAPABILITIES statements to define variations on implementing notifications. The definition and usage of the NOTIFICATION-GROUP macro is exactly corresponds to the OBJECT-GROUP macro.
- Every object in a MIB module (other than those defined as `not-accessible`) is required to be contained in at least one object group. This allows a MIB compiler to flag the common error of inadvertently forgetting to include objects in groups.
- The `accessible-for-notify` keyword is allowed to be present in the STATUS clause of the two relevant conformance macros: MODULE-COMPLIANCE and AGENT-CAPABILITIES.
- The `deprecated` keyword is allowed in the STATUS clause of the OBJECT-GROUP macro.
- The need to import objects is clarified: when a MIB module is IMPORTed by a MODULE-COMPLIANCE statement or by an AGENT-CAPABILITIES statement, then that IMPORTS clause implicitly imports all object definitions from the MIB module; thus, any objects referenced from that MIB module do not themselves need to be explicitly IMPORTed.
- Usage of the value of an AGENT-CAPABILITIES macro is clarified as being intended for use as the value of `sysORID`, but not of `sysObjectID`.

Note that the above changes and clarifications in all three documents were designed such that MIB modules written according to the recommended usage of RFCs 1442-1444 are all still valid under the rules defined by RFCs 1902-1904.

Frequently Asked Questions

Kaj Tesink
Bell Communications Research

Some questions never go away. Here are three questions on the Trunk MIBs (RFC 1406,RFC 1407,RFC 1595), which are currently being revised.

Q: SYNTAX, `Gauge`

Why do the performance counters in the Trunk MIBs have the SYNTAX `Gauge`?

A: Response:
Performance counts in the Trunk MIBs are kept in 15 minute intervals. That is, you keep on counting for 15 minutes, store it, and start a new interval. Looks like a `Counter`? Well, thats what we thought when we started out (see RFC 1232 and RFC 1233). But we were wrong. A `Counter` is defined as monotonically increasing (and then wraps). Something that is reset every 15 minutes doesn't qualify as such and therefore the correct SYNTAX should be `Gauge`.

Q: Discontinuities, Interval Counts

What happens when an interval count for an interface becomes unavailable, for example, because of restart of the agent? If an implementation supports the objects but an instance is unavailable then how should an agent respond? Should the agent:

- return `noSuch*`;
- return `noError` along with a value of zero; or,
- return `genErr`?

A: Response:
When object instances are not available for whatever reason, they are simply unavailable. As such, the agent should respond with `noSuch*`. Returning 0 would be misleading because the instance is unavailable and the value 0 may be wrong.

Nevertheless, this interval issue has been debated many times. For example, what happens with the table totaling all intervals? Unfortunately, there is only so much you can do to convey correct information, and the law of diminishing returns is looming. The problem may be partly solved by mandating when for calculating the totals the value 0 should be used for gaps in the table; and, by defining a separate object indicating potentially missing intervals.

Q: Persistent Data, Cumulative Data

Why is so much history kept in the Trunk MIB interval tables, and why is there a separate table totaling everything up? Doesn't the SNMP principle of simple agents apply? Shouldn't this be a manager task?
A: Response:
Here is an example where traditional telecommunications collides with traditional data communications. Traditional SNMP relies on continuous counters relying on calculating the difference between subsequent samples. Traditional practice in telecommunications is to keep performance information in 15 minute intervals. Notice that CSU/DSUs do the same thing. When the trunk MIBs were defined they did nothing else than mib'ifying what was there anyway.

Of course, this practice is redundant and therefore should be avoided. But is it wrong? In the interests of simple agents this practice should be avoided as general policy. But an argument a large network is not helped by managers having to retrieve large numbers of counters continuously, and is better off by storing some history in the agents (for example, this capability could be made configurable allowing probing in particular areas of interest in a network).

Industry Comment

Marshall T. Rose
Dover Beach Consulting

Welcome to the year's second issue of ***The Simple Times***.

The good news is that I've received only positive comments on the previous issue. One reader exclaimed:

> "More technical content per square of paper than I've seen in a long time..."

Similarly, the *unofficial* index of IETF MIB modules `http://www.simple-times.org/pub/simple-times/html/`, has also proven popular, with many asking why all RFCs couldn't be published in HTML.

Special Issues

In the last issue, I noted that the editorial policy of ***The Simple Times*** was changed to solicit more outside contributions instead of featured columns. (The old policy had one technical article per issue, whilst the new policy requires three.)

Reader feedback was positive on this change, and, as luck would have it, a major topic in the SNMP community, agent extensibility, is now reaching a consensus point. So, this is a special issue of ***The Simple Times*** focusing on that special topic, with coverage of five articles!

Historically, I have long opposed a standards-based effort to agent extensibility. I felt that the issues are too implementation-specific to favor standardization; further, I felt that an imperfect standardized resolution of these issues would diminish the correct behavior of an SNMP implementation. Over time however, the issues have become clearly understood and there is now sufficient experience for a standards-based effort to proceed and succeed. As such, I am pleased that the major contributors to the IETF effort on agent extensibility were able to contribute to ***The Simple Times***.

I think it fair to speculate that we will have another special issue later this year, dealing with another major topic as it approaches consensus. I'll leave that to the reader to guess which topic that might be. (Hint: it doesn't, thankfully, deal with security!) Of course, ***The Simple Times*** still needs your help: please consider contributing a technical article to the community! The publication schedule is quarterly, so that's plenty of time for you to do some serious writing.

Standards Summary

SNMPv1 Framework

Consult the latest version of *Internet Official Protocol Standards*. As of this writing, the latest version is RFC 1920.
Full Standards:

- RFC 1155 - Structure of Management Information (SMI);
- RFC 1157 - Simple Network Management Protocol (SNMP); and,
- RFC 1212 - Concise MIB definitions.

Proposed Standards:

- RFC 1418 - SNMP over OSI;
- RFC 1419 - SNMP over AppleTalk; and,
- RFC 1420 - SNMP over IPX.

SNMPv2 Framework

Draft Standards:

- RFC 1902 - SMI for SNMPv2;
- RFC 1903 - Textual Conventions for SNMPv2;
- RFC 1904 - Conformance Statements for SNMPv2;
- RFC 1905 - Protocol Operations for SNMPv2;
- RFC 1906 - Transport Mappings for SNMPv2;
- RFC 1907 - MIB for SNMPv2; and,
- RFC 1908 - Coexistence between SNMPv1 and SNMPv2.

Experimental:

- RFC 1901 - Introduction to Community-based SNMPv2;
- RFC 1909 - An Administrative Infrastructure for SNMPv2; and,
- RFC 1910 - User-based Security Model for SNMPv2.

MIB Modules

An *unofficial* index of IETF MIB modules is available.
`http://www.simple-times.org/pub/simple-times/html/`
Full Standards:

- RFC 1213 - Management Information Base (MIB-II); and,
- RFC 1643 - Ether-Like Interface Type (SNMPv1).

Draft Standards:

- RFC 1493 - Bridge MIB;
- RFC 1516 - IEEE 802.3 Repeater MIB;
- RFC 1559 - DECnet phase IV MIB;
- RFC 1657 - BGP version 4 MIB;
- RFC 1658 - Character Device MIB;
- RFC 1659 - RS-232 Interface Type MIB;
- RFC 1660 - Parallel Printer Interface Type MIB;
- RFC 1694 - SMDS Interface Protocol (SIP) Interface Type MIB;
- RFC 1724 - RIP version 2 MIB;
- RFC 1742 - AppleTalk MIB;
- RFC 1748 - IEEE 802.5 Token Ring Interface Type MIB;
- RFC 1757 - Remote Network Monitoring MIB; and,
- RFC 1850 - OSPF version 2 MIB.

Proposed Standards:

- RFC 1285 - FDDI Interface Type (SMT 6.2) MIB;
- RFC 1315 - Frame Relay DTE Interface Type MIB;
- RFC 1354 - IP Forwarding Table MIB;
- RFC 1381 - X.25 LAPB MIB;
- RFC 1382 - X.25 PLP MIB;
- RFC 1406 - DS1/E1 Interface Type MIB;
- RFC 1407 - DS3/E3 Interface Type MIB;
- RFC 1414 - Identification MIB;
- RFC 1461 - Multiprotocol Interconnect over X.25 MIB;
- RFC 1471 - PPP Link Control Protocol (LCP) MIB;
- RFC 1472 - PPP Security Protocols MIB;
- RFC 1473 - PPP IP Network Control Protocol MIB;
- RFC 1474 - PPP Bridge Network Control Protocol MIB;
- RFC 1512 - FDDI Interface Type (SMT 7.3) MIB;
- RFC 1513 - Token Ring Extensions to RMON MIB;
- RFC 1514 - Host Resources MIB;
- RFC 1515 - IEEE 802.3 Medium Attachment Unit (MAU) MIB;
- RFC 1525 - Source Routing Bridge MIB;
- RFC 1565 - Network Services Monitoring MIB;
- RFC 1566 - Mail Monitoring MIB;
- RFC 1567 - X.500 Directory Monitoring MIB;
- RFC 1573 - Evolution of the Interfaces Group of MIB-II;
- RFC 1595 - SONET/SDH Interface Type MIB;
- RFC 1604 - Frame Relay Service MIB;
- RFC 1611 - DNS Server MIB;
- RFC 1612 - DNS Resolver MIB;

- RFC 1628 - Uninterruptible Power Supply MIB;
- RFC 1650 - Ether-Like Interface Type (SNMPv2);
- RFC 1666 - SNA NAU MIB;
- RFC 1695 - ATM MIB;
- RFC 1696 - Modem MIB;
- RFC 1697 - Relational Database Management System MIB;
- RFC 1747 - SNA DLC MIB;
- RFC 1749 - 802.5 Station Source Routing MIB; and,
- RFC 1759 - Printer MIB.

Experimental:

- RFC 1187 - Bulk table retrieval with the SNMP;
- RFC 1224 - Techniques for managing asynchronously generated alerts;
- RFC 1238 - CLNS MIB; and,
- RFC 1592 - SNMP Distributed Program Interface (SNMP-DPI); and,
- RFC 1792 - TCP/IPX Connection MIB Specification.

Informational:

- RFC 1215 - A convention for defining traps for use with the SNMP;
- RFC 1270 - SNMP communication services;
- RFC 1303 - A convention for describing SNMP-based agents;
- RFC 1321 - MD5 message-digest algorithm;
- RFC 1470 - A network management tool catalog; and,
- RFC 1503 - Automating Administration in SNMPv2 Managers.

Historic:

- RFC 1156 - Management Information Base (MIB-I) (see RFC 1213);
- RFC 1161 - SNMP over OSI (see RFC 1418);
- RFC 1227 - SNMP MUX protocol and MIB;
- RFC 1228 - SNMP Distributed Program Interface (SNMP-DPI) (see RFC 1592);
- RFC 1229 - Extensions to the generic-interface MIB (see RFC 1573);
- RFC 1230 - IEEE 802.4 Token Bus Interface Type MIB;
- RFC 1231 - IEEE 802.5 Token Ring Interface Type MIB (see RFC 1748);
- RFC 1232 - DS1 Interface Type MIB (see RFC 1406);
- RFC 1233 - DS3 Interface Type MIB (see RFC 1407);
- RFC 1239 - Reassignment of experimental MIBs to standard MIBs;
- RFC 1243 - AppleTalk MIB (see RFC 1742);
- RFC 1248 - OSPF version 2 MIB (see RFC 1252);
- RFC 1252 - OSPF version 2 MIB (see RFC 1853);
- RFC 1253 - OSPF version 2 MIB (see RFC 1850);
- RFC 1269 - BGP version 3 MIB (see RFC 1657);
- RFC 1271 - Remote LAN Monitoring MIB (see RFC 1757);
- RFC 1283 - SNMP over OSI (see RFC 1418);
- RFC 1284 - Ether-Like Interface Type MIB (see RFC 1398);
- RFC 1286 - Bridge MIB (see RFC 1493 and RFC 1525);
- RFC 1289 - DECnet phase IV MIB (see RFC 1559);
- RFC 1298 - SNMP over IPX (see RFC 1420);
- RFC 1304 - SMDS Interface Protocol (SIP) Interface Type MIB (see RFC 1694);
- RFC 1316 - Character Device MIB (see RFC 1658);
- RFC 1317 - RS-232 Interface Type MIB (see RFC 1659);
- RFC 1318 - Parallel Printer Interface Type MIB (see RFC 1660);
- RFC 1351 - SNMP Administrative Model;
- RFC 1352 - SNMP Security Protocols;
- RFC 1353 - SNMP Party MIB;
- RFC 1368 - IEEE 802.3 Repeater MIB (see RFC 1516);
- RFC 1389 - RIPv2 MIB (see RFC 1724);

- RFC 1398 - Ether-Like Interface Type MIB (see RFC 1643);
- RFC 1441 - Introduction to SNMPv2 (see RFC 1901);
- RFC 1442 - SMI for SNMPv2 (see RFC 1902);
- RFC 1443 - Textual Conventions for SNMPv2 (see RFC 1903);
- RFC 1444 - Conformance Statements for SNMPv2 (see RFC 1904);
- RFC 1445 - Administrative Model for SNMPv2;
- RFC 1446 - Security Protocols for SNMPv2;
- RFC 1447 - Party MIB for SNMPv2;
- RFC 1448 - Protocol Operations for SNMPv2 (see RFC 1905);
- RFC 1449 - Transport Mappings for SNMPv2 (see RFC 1906);
- RFC 1450 - MIB for SNMPv2 (see RFC 1907);
- RFC 1451 - Manager-to-Manager MIB;
- RFC 1452 - Coexistence between SNMPv1 and SNMPv2 (see RFC 1908);
- RFC 1596 - Frame Relay Service MIB (see RFC 1604);
- RFC 1623 - Ether-Like Interface Type MIB (see RFC 1643); and,
- RFC 1665 - SNA NAU MIB (see RFC 1666).

Subscribing to SNMP-related Working Groups

- 100VG-AnyLAN MIB Working Group
  `<vgmib-request@hprnd.rose.hp.com>`
- Application MIB Working Group
  `<applmib-request@emi-summit.com>`
- AToM MIB Working Group
  `<atommib-request@thumper.bellcore.com>`
- BGP Working Group
  `<iwg-request@ans.net>`
- Bridge MIB Working Group
  `<bridge-mib-request@pa.dec.com>`
- Character MIB Working Group
  `<char-mib-request@decwrl.dec.com>`
- Data Link Switching MIB Working Group
  `<aiw-dlsw-mib@networking.raleigh.ibm.com>`
- DECnet Phase IV MIB Working Group
  `<phiv-mib-request@jove.pa.dec.com>`
- Entity MIB Working Group
  `<entmib-request@cisco.com>`
- FDDI MIB Working Group
  `<fddi-mib-request@cs.utk.edu>`
- Frame Relay Service MIB Working Group
  `<frftc-request@nsco.network.com>`
- Host Resources MIB Working Group
  `<hostmib-request@andrew.cmu.edu>`
- IEEE 802.3 Hub MIB Working Group
  `<hubmib-request@hprnd.rose.hp.com>`
- IDR Working Group
  `<bgp@ans.edu>`
- Interfaces MIB Working Group
  `<if-mib-request@dtl.labs.tek.com>`
- IP over AppleTalk Working Group
  `<apple-ip-request@cayman.com>`
- IPLPDN Working Group
  `<iplpdn-request@nri.reston.va.us>`
- IPv6 MIB Working Group
  `<ip6mib-request@research.ftp.com>`
- ISDN MIB Working Group
  `<isdn-mib-request@combinet.com>`
- IS-IS for IP Internets Working Group
  `<isis-request@merit.edu>`
- Mail and Directory Management Working Group
  `<ietf-madman-request@innosoft.com>`
- Modem Management Working Group
  `<modemmgt-request@telebit.com>`
- NOCtools Working Group
  `<noctools-request@merit.edu>`
- OSPF IGP Working Group
  `<ospf-request@gated.cornell.edu>`
- PPP Extensions Working Group
  `<ietf-ppp-request@merit.edu>`
- RIP Working Group
  `<ietf-rip-request@xylogics.com>`

- Remote Network Monitoring Working Group
  `<rmonmib-request@cisco.com>`
- Routing over Large Clouds Working Group
  `<rolc-request@nexen.com>`
- SNA DLC Services MIB Working Group
  `<snadlcmib-request@cisco.com>`
- SNA NAU Services MIB Working Group
  `<snanaumib-request@cisco.com>`
- SNMP Agent Extensibility Working Group
  `<agentx-request@fv.com>`
- SNMPv2 Working Group
  `<snmpv2-request@tis.com>`
- TCP Client Identity Protocol
  `<ident-request@nri.reston.va.us>`
- DS1/DS3 MIB Working Group
  `<trunk-mib-request@cisco.com>`
- Uninterruptible Power Supply Working Group
  `<ups-mib-request@cs.utk.edu>`
- X.25 MIB Working Group
  `<x25mib-request@dg-rtp.dg.com>`

Internet Resources

Automated Services

Automated services are available in the Internet, provided "as is" with no express or implied warranty. Each service accepts a MIB module in the body of a message.
MIB module checking:

- Emissary `<mib-checker@epilogue.com>`
- mosy `<mosy@simple-times.org>`

MIB module conversion:

- convert SNMPv2 module to SNMPv1
  `<mib-v2tov1@simple-times.org>`
- convert MIB module to HTML
  `<mib-2html@simple-times.org>`

Source Implementations

Source implementations are available in the Internet, provided under various no-fee licensing terms.
Agents:

- Beholder: an RMON agent for UNIX
  `ftp://dnpap.et.tudelft.nl/pub/btng/`
- CMU SNMP: an SNMPv2u agent for UNIX
  `ftp://ftp.cisco.com/ftp/kzm/cmusnmp.tar.gz`
- UT-snmpV2: an SNMPv2 agent for SPARCs
  `http://snmp.cs.utwente.nl/`
- WILMA: an SNMP agent for UNIX
  `http://www.ldv.e-technik.tu-muenchen.de/dist/INDEX.html`

Compilers:

- mosy: a MIB compiler
  `ftp://ftp.cisco.com/ftp/kzm/snmptcl.tar.gz`
- SMIC: a MIB compiler
  `<dperkins@scruznet.com>`
- snacc: an ASN.1 compiler
  `ftp://ftp.cs.ubc.ca/pub/local/src/snacc/`

Platforms:

- NOCOL: a network monitoring package for UNIX
  `ftp://ftp.navya.com/pub/vikas/`
- Scotty: a Tcl-based environment for management applications
  `http://www.cs.tu-bs.de/ibr/projects/nm/`
- snmptcl: a Tcl-based environment for management applications
  `ftp://ftp.cisco.com/ftp/kzm/snmptcl.tar.gz`
- SNMPY: a Python-based environment for management applications
  `http://www.rdt.monash.edu.au/\~{}anthony/snmpy/`
- Tricklet: a Perl-based environment for management applications
  `ftp://dnpap.et.tudelft.nl/pub/btng/`
- WILMA: an X-based monitoring package for UNIX
  `http://www.ldv.e-technik.tu-muenchen.de/dist/INDEX.html`

Other Resources

- IETF Home Page
  `http://www.ietf.cnri.reston.va.us/`
- SNMP Testing FAQ
  `http://www.iwl.com/faq.html`
- User-based Security Model (USEC) Resources
  `http://www.simple-times.org/pub/simple-times/usec/`

Announcements

SNMP Test Summit III

InterWorking Labs announces the SNMP Test Summit III from June 3-7, 1996 at the Winsock Labs in San Jose, California (US). Engineers who wish to find and fix bugs in their SNMPv2 (including c, usec, and star) implementations should attend.

For more information: +1 408 459 9817.

Publication Information

Featured Columnists

Keith McCloghrie	Cisco Systems
Marshall T. Rose	Dover Beach Consulting
Kaj Tesink	Bell Communications Research

Contact Information

E-mail	`st-editorial@simple-times.org`
ISSN	1060–6068

Submissions

The Simple Times solicits high-quality articles of technology and comment. Technical articles are refereed to ensure that the content is marketing-free. By definition, commentaries reflect opinion and, as such, are reviewed only to the extent required to ensure commonly-accepted publication norms.

The Simple Times also solicits terse announcements of products and services, publications, and events. These contributions are reviewed only to the extent required to ensure commonly-accepted publication norms.

Submissions are accepted only via electronic mail, and must be formatted in HTML version1.0. Each submission must include the author's full name, title, affiliation, postal and electronic mail addresses, telephone, and fax numbers. Note that by initiating this process, the submitting party agrees to place the contribution into the public domain.

Subscriptions

The Simple Times is available in three editions: HTML, ASCII, and PostScript. For more information, send a message to

```
st-subscriptions@simple-times.org
```

with a `Subject:` line of

```
help
```

Back issues are available via either the Web or FTP, i.e.,

```
http://www.simple-times.org
ftp://ftp.simple-times.org
```

look under `/pub/simple-times/issues/`. In addition, ***The Simple Times*** has several hard copy distribution outlets. Contact your favorite SNMP vendor and see if they carry it.