The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS
VOLUME 7, NUMBER 2 NOVEMBER, 1999

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 as an online journal in HTML, PDF and PostScript. New issues are announced via an electronic mailing list. For information on subscriptions, see page 18.

Editorial

Aiko Pras, University of Twente
Jürgen Schönwälder, TU Braunschweig

One of the interesting additions to the Internet management RFCs are the specifications that define Distributed Management (DISMAN). The idea to distribute management functionality over multiple managers is relatively old and can already be found in the early SNMPv2 drafts, which defined the Manager to Manager (M2M) MIB to facilitate the delegation of polling tasks to intermediate level managers. Since this MIB could only be used for simplistic management tasks, it was generally seen as a starting point for distributed management, and not as the final solution. To develop more powerful distributed management approaches, a new IETF DISMAN working group was therefore formed, and responsibility for the M2M MIB moved to this group.

Currently the DISMAN group is working on three different approaches. The first approach can be seen as the further development of the M2M MIB, and is described in three Internet Drafts: the Expression MIB, the Event MIB and the Notification Log MIB. The second approach is based on scripts and is defined in two RFCs: the Script MIB and the Scheduling MIB. The third approach focuses on MIBs for specific management functions that can be invoked on remote devices. Such MIBs are defined in the Remote Operations Internet Draft. It should be noted that these approaches should not be considered as competitors, but as complements; for a certain task it may be better to use one approach, and for a different task another approach. Unfortunately it is not easy to integrate the three approaches and use, for example, the Scheduling MIB in combination with the Event MIB.

This and an upcoming issue of *The Simple Times* provide an overview of the various DISMAN approaches. In this issue we will focus on the Schedule and Script MIB; in a future issue we will discuss the Event, Expression and Notification Log MIB, as well as the Remote Operations MIB. This issue starts with an article by Alan Luchuk, who discusses the Schedule MIB and gives his experiences while implementing this MIB. The other articles focus on the Script MIB. First David Levi provides an overview of this MIB. Next Éamonn McManus describes his Script MIB implementation. The third article is by Frank Strauß, who analysed the performance of another Script MIB implementation. Finally Jürgen Quittek and Cornelia Kappler describe some practical experiences with Script MIB applications. As usual, this issue also contains a standards summary, as well as the calendar and announcement section.

The issue you are now reading is already the twentieth issue of *The Simple Times*. The first issue appeared in March 1992 and, including this issue, we have published more than 360 pages of text. Recently a number of people asked whether it would be possible to bundle all issues, find a publisher and print it as book. Such a book could provide an extensive index and would serve as a nice reference. To find out if people would be interested in buying such book, we have added a small survey to our homepage; be sure to complete this survey if you would like to see the first twenty issues of *The Simple Times* as a reference book in your bookshelf.

Schedule MIB

Alan Luchuk, SNMP Research Incorporated

There is a well-known need to perform network management operations at periodic intervals or at scheduled times. For example, a network manager may need to enable or disable network interfaces at certain times of day. The `DISMAN-SCHEDULE-MIB` was developed to address this need. This article provides an overview of the `DISMAN-SCHEDULE-MIB` and discusses SNMP Research's implementation experience.

RFC 2591, currently at Proposed Standard status, details the `DISMAN-SCHEDULE-MIB`. RFC 2591 is a product of the Distributed Management (DISMAN) working group, within the Operations and Management Area of the IETF.

MIB Overview

The `DISMAN-SCHEDULE-MIB` consists of a single scalar (`schedLocalTime`) and a single table (`schedTable`). The `schedLocalTime` MIB object specifies the agent's notion of the current (real) time. This object is implemented as an 11-octet `DateAndTime` textual convention.

Each row (`schedEntry`) in the `schedTable` specifies a single scheduled event. Each row is indexed by an owner string (`schedOwner`) and an event name string (`schedName`). The `schedOwner` allows multiple users to schedule events in an agent. The `schedName` allows each user to schedule multiple events. The `schedDescr` provides a human-readable description of the scheduled event.

In the `schedTable`, the row indixes are each prefixed by the length of the index string. For example if the `schedOwner` is `P`, a one-character string, with an ASCII value of 80 (in decimal), the owner index field becomes 1.80. Similarly, if the `schedName` is `Q`, a one-character string, with an ASCII value of 81 (in decimal), the name index field becomes 1.81. In this example, the complete index for this row would be 1.80.1.81. While making it more difficult to configure, this double-indexing facilitates access control using the view-based access control model defined in RFC 2575.

The `schedRowStatus` is an object with the syntax of the `RowStatus` textual convention. It lets the user manage the status of a table row - the creation, editing, and deletion of rows - in the `schedTable`. The `schedStorageType` object lets the user specify whether the row is saved in non-volatile storage. The `schedAdminStatus` object specifies whether the desired status of the scheduled event is enabled or disabled. The `schedOperStatus` object reports the actual status of the scheduled event.

Setting Objects at the Trigger Time

The `schedVariable` object specifies which MIB object should be set at the specified time. This object is an OID, and it must specify completely the object to be set, including all required instancing information. The `schedValue` object specifies the `Integer32` value set at the scheduled time. If needed, the `schedContextName` object specifies the SNMP context where `schedVariable` occurs.

The `DISMAN-SCHEDULE-MIB` can set only a single `Integer32` MIB object at the trigger time. If the capability to set multiple `Integer32` objects at the trigger time is needed, multiple events can be scheduled for the same trigger time. If the capability to set non-`Integer32` objects is required, the `DISMAN-SCHEDULE-MIB` can be coupled with another facility such as the `DISMAN-SCRIPT-MIB`.

The `DISMAN-SCHEDULE-MIB` supports setting MIB objects in the same SNMP agent in which the `DISMAN-SCHEDULE-MIB` is implemented. It does not support setting MIB objects in another SNMP agent on another IP host.

If a scheduled set request fails, the `schedLastFailure` object specifies the error status returned by the most recent set operation. The `schedLastFailed` object, a `DateAndTime` object, reports the date and time of the most recent set failure. The `schedFailures` object reports a count of the number of set failures for this scheduled event.

Types of Scheduling

The `schedType` object specifies the type of scheduled event; it may have the values `periodic(1)`, `calendar(2)`, or `oneshot(3)`. The `periodic(1)` value specifies that the set request be performed repeatedly at regular timed intervals. The `calendar(2)` value specifies the set request be performed at specific dates and times. The `oneshot(3)` value specifies that a set request is performed once at a specific date and time, then the event is finished.

1. Periodic Scheduling:

   For periodic events, the `schedInterval` value specifies the minimum number of seconds between an event performed repeatedly.

2. Calendar Scheduling:

   For calendar-scheduled events, the `schedWeekDay` `BITS` object lets the user specify the day(s) of the week when the event should occur. Events may be scheduled on multiple days of the week by enabling multiple bits in the object. To ignore the day of the week for scheduled events, `schedWeekDay` must be set with all bits enabled. (Enabling a bit means setting that bit to a value of 1.)

   The `schedMonth` `BITS` object lets the user specify the month(s) of the year when the event should occur. Events may be scheduled in multiple months of the year by enabling multiple bits in the object. To ignore the month, the `schedMonth` object must be set with all bits enabled.

   The `schedDay` `BITS` object lets the user specify the day(s) of the month when the event should occur. It also allows the user to specify that the event be triggered on a specific day from the end of the month (e.g., on the 7th day from the end of the month). Events may be scheduled on multiple days of the month by setting multiple bits in the object. To ignore the day of the month, `schedDay` object must be set with all bits enabled.

   The `schedHour` `BITS` object lets the user specify the hour(s) of the day when the event should occur. Events may be scheduled to occur on multiple hours of the day by enabling multiple bits in the object. To ignore the hour, the `schedHour` object must be set with all bits enabled.

   The `schedMinute` `BITS` object lets the user specify the minute(s) of the hour when the event should occur. Events may be scheduled to occur on multiple minutes of the hour by enabling multiple bits in the object. To ignore the minute, the `schedMinute` object must be set with all bits enabled.

3. One-Shot Scheduling:

   One-shot scheduling is almost identical to calendar scheduling, with one difference. The agent will repeat calendar-scheduled events, but the agent will execute a one-shot event only a single time, then the event is finished. Because one-shot scheduling is almost identical to calendar scheduling, all of the MIB objects that configure calendar scheduling (`schedWeekDay`, `schedMonth`, `schedDay`, `schedHour`, `schedMinute`) also configure one-shot scheduling.

The `DISMAN-SCHEDULE-MIB` can be configured to send notifications (traps) under certain circumstances. The `schedActionFailure` notification is generated when the invocation of a scheduled action fails.

Implementation Experience

During implementation, we found RFC 2591 clear and complete. However, RFC 2591 contains two issues that we believe make the `DISMAN-SCHEDULE-MIB` harder-to-use or less versatile than it could be otherwise.

The rows in the `schedTable` are doubly-indexed with `SnmpAdminString` indexes. This was a design decision by the DISMAN working group to support view-based access control to the different scheduled events. We strongly agree with the design for view-based access control. However, we believe there are other ways of providing equivalent support of view-based access control that also provide greater end-user simplicity for SNMP exact-instance get and set requests than two `SnmpAdminString` row indexes.

Generic MIB browser tools typically do not permit octet string table row indices to be entered in a human-friendly format. Thus, requiring `SnmpAdminString` row indexes makes specific-instance configuration of the `DISMAN-SCHEDULE-MIB` difficult using these generic tools. This difficulty either

- encourages `DISMAN-SCHEDULE-MIB` users to use very simple `schedOwner`/`schedName` strings (like "A" and "B"); or
- requires the use of specialized configuration tools that support entering the octet string table row indices in a human-friendly format.

We suspect that in production-oriented network management environments, the ability to do ad-hoc SNMP exact-instance get and set requests easily (using network management tools like HP OpenView) will overshadow the utility of meaningful owner and event names in the row indexes. We suspect network managers will opt to simplify ad-hoc SNMP requests by using very short row indexes.

Second, the DISMAN-SCHEDULE-MIB only supports setting a single Integer32 value at the scheduled time. This somewhat restricts the stand-alone utility of the DISMAN-SCHEDULE-MIB. It would add utility if it supported setting multiple objects and different data-types at the scheduled time. This enhancement would make it possible, for example, to create and activate a table row at the scheduled time.

Currently, our implementation exists as a sub-agent in a master-agent/sub-agent architecture, but it is easily portable into a monolithic agent. Our implementation supports all MIB objects and notifications in RFC 2591, as well as periodic, calendar, and one-shot scheduled events. In addition, for computing platforms without real-time clocks, conditional compilation macros support the lower compliance level described in RFC 2591.

Although our implementation proceeded smoothly, we did have difficulties sufficiently understanding two issues. RFC 2591 hints at these issues, but more explanation or examples of these issues would help ensure correct future implementations. Fortunately for us, one of the MIB authors, David Levi, helped us understand these issues.

First, as described in RFC 2591, calendar scheduled events should be triggered as soon as possible on, or after, the minute they occur. When a scheduled event is triggered, its completion time may be significant, perhaps due to excessive system load or other causes. If a series of events is scheduled to occur during a given minute, the execution delays may accumulate. However, the agent must be properly written so that cumulative processing delays do not cause it to "miss" events scheduled for triggering in subsequent minutes.

As an example, suppose 250 events are scheduled to occur at midnight on the first day of the month. The elapsed time to trigger the 250 events may take five minutes. Upon exit from the event trigger loop, the DISMAN-SCHEDULE-MIB agent may query the system's real-time, and find the real-time now is 00:05. The agent must properly handle this possibility; it must not skip events scheduled to occur during the intervening minutes. Upon finding the real-time is 00:05, the agent must trigger all events scheduled for the minutes 00:01 through 00:05.

Second, during the daylight savings time changes, the agent must properly handle "nonexistent times" or "ambiguous times." For example, in the spring season when clocks are set forward an hour, events scheduled for the missing hour must be triggered immediately after the time change. Similarly, in the fall season when clocks are set backward an hour, events scheduled for the duplicate hour should be triggered only once. We found that thoroughly testing and debugging this capability was time consuming (no pun intended).

Operational Experience

We ran, tested, and productized our implementation of the DISMAN-SCHEDULE-MIB. We demonstrated it at the two most recent U.S. Networld+Interop trade shows. In addition, we tested and demonstrated our DISMAN-SCHEDULE-MIB sub-agent launching scripts in our DISMAN-SCRIPT-MIB sub-agent, and found that this scenario works as expected. Because (to the best of our knowledge) we have the only DISMAN-SCHEDULE-MIB implementation, we have not tested its interoperability with others.

During testing, we stress-tested our implementation by scheduling several high-frequency periodic sets. Because our initial implementation uses a master-agent/sub-agent architecture, the DISMAN-SCHEDULE-MIB sub-agent executes independently from other sub-agents that execute the triggered set requests. This means that our DISMAN-SCHEDULE-MIB sub-agent may trigger set requests faster than the set requests can be completed. Not surprisingly, we re-discovered the basic queuing theory concept: If the mean time between periodic sets is shorter than the mean time to service the sets, infinite queues can result. A cautionary note in RFC 2591 about possible undesirable side effects of high-frequency periodic sets may help future implementors and users avoid surprises.

Summary

All in all, RFC 2591 is implementable, workable, and useful. The DISMAN-SCHEDULE-MIB is fairly simple and has utility apart from other MIBs (e.g., the DISMAN-SCRIPT-MIB or the DISMAN-EVENT-MIB). Although we believe changes to RFC 2591 would make the DISMAN-SCHEDULE-MIB easier to use and increase its utility, we believe RFC 2591 can be advanced further in the standards process. We can see no compelling reason to radically alter RFC 2591 or merge its functionality with other MIBs.

Introduction to the Script MIB

David Levi, Nortel Networks

This article provides an overview of the Script MIB which is a product of the IETF Distributed Management (DISMAN) working group. The `DISMAN-SCRIPT-MIB` is documented in RFC 2592.

History

The `DISMAN-SCRIPT-MIB` was partly derived from a MIB designed by myself and others at SNMP Research, Inc. This MIB was submitted as a potential starting point when the DISMAN working group was being formed. At that time, there were ongoing discussions about the approach that should be taken by the new working group.

These discussions involved topics such as whether the working group should focus on MIB design and SNMP-based approaches to distributed management, or whether other non-SNMP-based approaches should be considered. In addition, the discussions considered the question of whether the MIBs designed by the working group should attempt to provide very specific functionality for distributing management tasks, or whether they should provide more general mechanisms.

Ultimately, the working group decided to focus exclusively on MIB design, and to follow both approaches of providing MIBs with specific functionality, and MIBs with more general functionality. In particular, the working group decided to design several special-purpose MIBs, among which is the `DISMAN-SCHEDULE-MIB`, and to design a general purpose MIB for distributing scripts, the `DISMAN-SCRIPT-MIB`. This approach has resulted in a good start in infrastructure for distributed management (consisting of the various special-purpose MIBs), as well as a general purpose mechanism for implementing additional distributed management functionality not yet addressed by the disman working group.

MIB Overview

The `DISMAN-SCRIPT-MIB` provides mechanisms for distributing scripts which perform arbitrary management tasks to remote devices, which can execute these scripts. Executing scripts in remote devices can reduce the processing load on a central management station. But more importantly, it provides a mechanism for keeping polling local.

The traditional model in SNMP of having a single management station managing many agents runs into scalability problems when a network grows too large. This is because the amount of data that the management station must retrieve and process grows rapidly as the size of the network grows.

A solution to this problem is for the management station to distribute the collection and processing of data to other devices in the network. The `DISMAN-SCRIPT-MIB` provides one mechanism to accomplish this.

The Nature of Scripts

The term "script" as used in the `DISMAN-SCRIPT-MIB` is a very broad term. The document imposes very few restrictions on what can be considered a script, only that a script contains some type of executable code which can be run by the device which implements the MIB. This means that an implementor of the MIB may choose the programming language(s) in which a script must be implemented. Currently, there are implementations which permit scripts written using various Unix shell languages, Java, and even proprietary languages designed specifically for the purpose of manipulating SNMP MIB objects.

The `DISMAN-SCRIPT-MIB` provides a pair of MIB tables, the `smLangTable` and the `smExtsnTable`, which allow an implementation of the MIB to advertise the script languages (and versions) which it supports, as well as any supported extensions to those languages. A management station which wishes to distribute scripts may read these tables to evaluate whether a particular device which implements the `DISMAN-SCRIPT-MIB` meets its needs.

Note that there are currently no requirements as to what languages must be supported by a `DISMAN-SCRIPT-MIB` implementation. This has been a somewhat contentious issue within the DISMAN working group, as everybody has his own favorite language. In the future there may be a minimum requirement that all implementations support a particular language, but at the moment, implementors are free to choose whichever languages they wish. There are potential interoperability issues related to these choices. However, a discussion of these issues is beyond the scope of this article.

Distribution of Scripts

The `DISMAN-SCRIPT-MIB` provides two mechanisms for distributing scripts. These mechanisms are referred to as the pull-model and the push-model. Both of these models use a MIB table, the `smScriptTable`, to specify the names and attributes of scripts which are to be distributed to a device.

The pull-model uses a Uniform Resource Locator (URL) to specify where a script is stored and how it should be retrieved. An entity which supports the MIB is responsible for retrieving a script configured using a URL.

The push-model uses a MIB table, the `smCodeTable`, to allow a management station to push a script to a device using SNMP SetRequest PDUs. In order to use the `smCodeTable`, a script must be representable by a simple string of binary data. The management station simply creates entries in the `smCodeTable` which contain this array of binary data, broken into pieces that fit in SNMP packets, and stored as SNMP `OCTET STRING` data. (The manner in which a script is split into `OCTET STRING` data could be determined by the particular script language being used.)

Execution of Scripts

The execution of scripts is controlled by a MIB table, the `smLaunchTable`. This table provides a mechanism for specifying arguments for scripts, as well as other attributes such as the maximum time that a script may run, the maximum number of concurrent invocations of a script from a launch table entry, and the maximum amount of time to keep the result of a script after termination. The table also contains a control object, `smLaunchStart`, which is used to actually initiate execution of a script.

Scripts which are executing or which have completed execution are represented in the `smRunTable`. This table contains information such as the time the script started and finished execution, the reason the script finished executing, the result of the script, the remaining time the script is allowed to run, and the remaining time until the entry in the `smRunTable` is aged out.

Sharing of Scripts

The `DISMAN-SCRIPT-MIB` provides a model for sharing scripts. The model provides conventions for specifying the owner of a script, and for providing one owner the privilege of running another owner's scripts. These mechanisms work in conjunction with the view-based access control model specified in RFC 2575. The model specifies whether one owner can run another owner's script based on whether the script is read-accessible as defined by RFC 2575.

The model provides two mechanisms for sharing scripts, which are roughly analogous to the concept in Unix systems for providing executable privileges and set-uid privileges on a file. The first mechanism allows an `smLaunchTable` entry created by one owner to refer to a script distributed by another owner. In this case, the script runs with the privileges of the owner who created the `smLaunchTable` entry, and the corresponding `smScriptTable` entry must also be accessible to that user. The second mechanism simply allows an owner to create an `smLaunchTable` entry whose `smLaunchStart` object is accessible to other owners. These other owners may set `smLaunchStart` in order to initiate execution of the script, and the script runs with the privileges of the owner who created the `smLaunchTable` entry.

Security

The `DISMAN-SCRIPT-MIB` provides a mechanism for running arbitrary applications on remote devices in a network. This can have serious implications on security in a network. The `DISMAN-SCRIPT-MIB` discusses these issues briefly. However, many of these issues are dependent on the systems on which the MIB is implemented. The security implications of deploying this MIB should be carefully considered by both implementors and by those deploying devices which implement this MIB.

Script MIB Implementation Experience

Éamonn McManus, Silicomp Research Institute

This article describes how we at the Silicomp Research Insitute implemented the Script MIB (RFC 2592) for a small embedded network device. Work on the implementation was concurrent with the later draft stages of the MIB and provided some useful feedback for the final version.

Target Device

The implementation runs on a small networked commodity device. This device has permanent storage only in the form of flash memory, and it has no real-time clock. It runs an embedded operating system with a complete TCP/IP network stack and an SNMP agent. It can be configured with a Java Virtual Machine (JVM) and we used this configuration both to write the code for the various Script MIB objects and to run the scripts themselves, which are Java programs in the JAR format. The presence of a JVM meant that we had support for URLs, needed to fetch scripts using the `smScriptSource` object.

Why use the Script MIB?

The scripts that can be downloaded to our implementation are rather device-specific so we were not particularly interested in third-party scripts or in porting scripts written for other Script MIB implementations. The main advantages for us of the Script MIB as opposed to a proprietary MIB were, first, that we did not have to do the work of designing the MIB, and second, that any management tool for the Script MIB can be used to download our scripts to the device. In other words, we were interested in protocol interoperability but not script interoperability.

Another important advantage of the Script MIB for us was that there was a reference implementation developed jointly by the Technical University of Braunschweig and NEC C&C Research that was permanently accessible over the Internet. During development we could check our client tools against this implementation both to make sure that they were correct and to see what the reference implementation did in cases we were unsure about.

Consequences of a Small System

A number of features of the Script MIB seem better fitted to bigger machines running interactive operating systems like Unix than to embedded devices.

Our device has no notion of a "user" or "owner." Some tables, such as the `smScriptTable`, are indexed by owner and name strings. As far as we are concerned, these are just two arbitrary strings that are combined to form the index. But with a better implementation of SNMP security, the owner string could gain a real meaning.

The device has a clock but, by default, no notion of real time. So the Script MIB objects that refer to start and end times actually give times relative to the Java base date of the beginning of 1970. It is possible to create and download a script that gets real time from somewhere (NTP, for instance) and sets the device's notion of time according to that. It has been proposed that device time be readable and, optionally, writable with a standard SNMP object something like `sysDateAndTime`, which would provide a useful alternative.

We left out every optional part of the MIB that we did not need. This includes the `smCodeTable`, the `smExtsnTable`, and the ability to suspend and resume scripts. We also did not implement the notifications. At least `smScriptAbort` is required by RFC 2592, so in this respect we are not compliant.

Some Extensions

We found that the Script MIB provided nearly all of the functionality that we needed. There were a few areas where it did not provide an explicit way to do what we wanted, but we were able to find ways to contort the implementation slightly to add the functionality without introducing major changes regarding the Script MIB specification.

We wanted it to be possible for scripts to be launched automatically every time the device booted. Scripts that provide network services fall into this category, for instance. Our solution was simply to decree that any launch button whose name begins with a "*" is automatically "pushed" when the system starts. This uses the existing mechanisms to define the parameters of the start-up script, such as its arguments and its allowed life-time. It also means that the same script can be launched any number of times on startup, perhaps with different arguments, for instance a network service that listens on several different ports.

We were also concerned that there was no way to launch a script without specifying an expire time for it. The maximum specifiable expire time is 2147483647 centiseconds, about 248 days, which was a little too finite for our liking. So when this maximum value is specified our implementation treats it as infinite.

An extension we considered was to include more script meta-data in the MIB. In our implementation, all scripts share the same Java namespace (with Java's notion of packages being used to avoid clashes) so one script can in theory use code from another script. This means that it could be useful to provide information about interscript dependencies, so that for instance we do not delete a script when there are other scripts present that depend on it. The way we considered doing this was to encode the information in the `smScriptDescr` object. In the end we decided that dependencies were better handled by the manager program, even though this means that it has to keep information about every script that has ever been downloaded in the past.

We also considered adding version information in the `smScriptName` object. This would mean that we could not look a script up in the `smScriptTable` just by using indexing, but would be forced to traverse part of the table to find what we were looking for. By putting the version string after the script name and ensuring that it is always the same length, we could traverse a sub-tree of the OID namespace to find any versions of a script. So far versioning has not been a problem and we have not dealt with it.

Problems and Solutions

1. Scanning for Java:
   In discussions on the DISMAN mailing list, some people have found that the way script languages are specified (by a target-dependent index into the `smLangTable`) is clumsy. We did not find this to be a problem, though. Just before downloading a script you scan through the `smLangTable` to find the index you are looking for. The table is unlikely to be big (it has exactly one entry in our implementation) so this does not take long.

2. Setting `smLaunchStart` to 0:
   The Script MIB specifies that if the `smLaunchStart` object in a launch button is set to 0, a script instance is created in the `smRunTable` with any available

smRunIndex. We implemented this functionality but do not use it. Since SNMP running over UDP is unreliable, if you do not get an acknowledgment to the set of smLaunchStart you have no way of knowing if the script actually was started. We have no use for unreliable script launches; others may, for instance in conjunction with the DISMAN-SCHEDULE-MIB.

3. Lost acknowledgment to smLaunchStart:
   If you set smLaunchStart in order to start a script but do not get an acknowledgment, then you do not know whether it was your request or the corresponding reply that was lost. It is important not to use a blind SNMP retransmit strategy in this case. If you resend the set request and it was actually the reply that was lost, then you will most likely get back an error because you are trying to reuse an existing smRunIndex. The right thing to do is to look in the smRunTable to see if the script you tried to launch is there. If not, it was indeed the request that was lost and you must retransmit it.

The smCodeTable

The only area where we found the design of the Script MIB unsatisfactory was the smCodeTable. We did not implement this optional part of the MIB, but we would have liked to have something like it. The advantage would be that a manager who has a script in a file somewhere can download it directly to the SNMP client without having to set up access through an FTP or HTTP server. SNMP is not very well adapted for bulk data transfer, especially in the manager-to-agent direction, so this would be convenient but somewhat slow. But the specification of the smCodeTable implied much too much extra code and data for us to implement it.

The problem with the smCodeTable is that it has been constructed to make it possible to edit a script "in place," that is, to issue SNMP set requests that modify parts of the text of a script that has been downloaded at an earlier time. An agent does not have to provide the smCodeTable, but if it does then it must also provide this editing capability.

It is not clear that being able to read back a script previously loaded into the device is useful, but one can certainly imagine circumstances where one would want to know exactly what is running in the managed system.

But it is very hard to imagine circumstances where one would want to make modifications to part of a script. A manager that is a program would never do this: it would read the script out, modify it, then send it back; or it would just push a new version of the script.

So it looks as if the only use for in-place modification is to allow humans to tweak scripts by issuing low-level SNMP commands. Most humans will not want to do that - they will want to edit the script on a workstation and send back the edited version.

Even humans who do want to edit scripts directly within the agent will be severely limited in what they can do. Scripts are divided into "lines" in the smCodeTable; these lines are indexed by line numbers that are assigned by the manager when the script is downloaded and cannot subsequently be changed. You will only be able to insert a new line if there is a gap in the line numbering, which means this must have been anticipated when the script was downloaded. And you can only insert as many new lines as the gap allows. People were prepared to deal with this sort of environment on microcomputers twenty years ago, but they are not now.

The point here is that providing this editable smCodeTable is not without cost. An agent has to keep the whole table for as long as the script is present, or at least it has to keep the smCodeIndex values and the mapping from them into offsets in the script. Small-system developers that want to provide code pushing will resent this overhead. "Scripts" are not necessarily 20-line one-off Tcl programs: they can be substantial binaries, or big Java packages, that will already be pushing the limits of the machine without adding extra permanent overhead that is only used during download. (I acknowledge that big down-loads are better done by pulling, or yet another protocol, rather than lock-step SNMP packets.)

Here is how we think code pushing should work:

- Entries can only be added to the smCodeTable before the corresponding smScriptAdminStatus is "enabled."

- The first entry to be added to the smCodeTable must have smCodeIndex 1, and every subsequent entry must have smCodeIndex one more than the current largest value. Each entry must have its smCodeRowStatus set to "active" before any subsequent entry can be added. Entries once added to the smCodeTable cannot be deleted.

  These constraints mean that during download the agent does not have to store smCodeRowStatus or smCodeIndex values anywhere. It does have to remember where each smCodeText object is within the script, but only while the script is being pushed.

- Once the script has been pushed, the manager sets the smScriptAdminStatus to "enabled." As at present, the fact that the code comes from the smCodeTable is indicated by an empty smScriptSource and an smScriptStorageType that is "volatile."

- When the smScriptRowStatus is set to "active," an agent may choose to delete the smCodeTable, or it may reorganize it so that the same code is presented using a different division into smCodeText objects. In particular it may decide to chop up the presented text into fixed-length smCodeText objects so that it does not have to remember the offsets corresponding to smCodeIndex values. This means that the smCodeTable takes up no storage: the agent can construct replies to SNMP get requests on the fly.

  If the smScriptStorageType is then made non-volatile, the script text will be written to non-volatile storage.

- The only way to modify the text of a script after it has been downloaded is to delete the smScriptEntry and create a new one.

Conclusions

The Script MIB is well adapted to bigger machines such as workstations and servers. Our experience has shown that it is also quite well adapted to smaller devices, with few exceptions.

Script MIB Performance Analysis

Frank Strauß, TU Braunschweig

This article presents an analysis of time and memory consumptions of the *Java Script MIB Implementation* (Jasmin), developed in a joint project between the Technical University of Braunschweig and NEC C&C Research Laboratories Berlin. We have evaluated some usage examples from RFC 2592 and their time contingent spent in the Jasmin specific SMX operations, that are documented in RFC 2593. We also present some scalability studies under increasing load of concurrently running scripts.

Jasmin Architecture

The purpose of the Jasmin implementation is to evaluate the IETF Script MIB. Jasmin has a modular architecture to increase flexibility and extensibility as shown in Figure 1.

The left side in Figure 1 shows the part which depends on the SNMP toolkit in use. The current Jasmin version uses the EMANATE SNMP agent toolkit from SNMP Research. A future version will add support for the UCD SNMP toolkit. The Script MIB interface is realized as a separate sub-agent. The central part of Figure 1 shows the toolkit independent Jasmin kernel and related parts. The Jasmin kernel communicates with the runtime engines (right side of Figure 1) through the *Script MIB Extensibility Protocol* defined in the Experimental RFC 2593. This lightweight ASCII protocol allows to separate language specific runtime engine(s) from the Jasmin kernel, so that new runtime engines can be added easily. Currently, there is only a single runtime engine which is based on a Java virtual machine and executes management scripts written in Java. For more information about the Jasmin prototype, visit http://www.ibr.cs.tu-bs.de/projects/jasmin/.

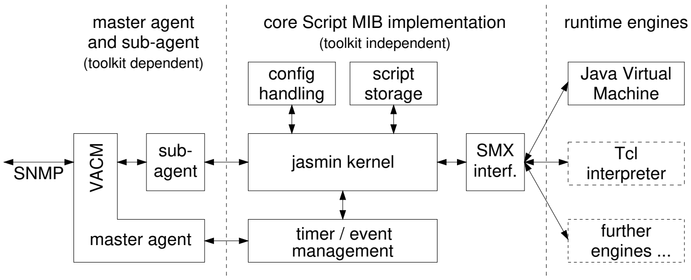

Figure 1: Jasmin Architecture.

Measurement Environment and Goals

There are many parameters in a Script MIB aware management environment that have a substantial impact on different performance aspects. Since only some characteristics of the Jasmin agent are of interest for this analysis, a few parameters were kept fixed during all measurement scenarios. All measurements have been made on a Sun Ultra-1 Model 140 agent host with 128 MBytes of physical memory running Solaris 2.5.1. An equivalent machine ran a set of manager side Tcl scripts to stimulate the agent. Both machines were otherwise unloaded and connected to a lightly loaded 100 MBit/s Ethernet segment.

Our major interests in this analysis have been:

- the time spent on typical operations on Script MIB tables, e.g. installing a script, starting a script, or retrieving the result of a running script,
- the time spent in single SMX commands sent by the Jasmin sub-agent to the Java runtime engine, processed by the runtime engine, and finally answered by the runtime engine with an SMX response,
- the amount of memory consumed by the Jasmin sub-agent and the Java runtime engine while many scripts are running concurrently.

The response times of Script MIB operations have been measured within the stimulating manager script. The SMX operations have been monitored by instrumenting the Jasmin code near the read/write system calls. Finally the memory consumptions of the Jasmin agent and the Java runtime engine have been monitored by regular ps(1) output while another Tcl script stimulated the agent. In all cases, very short Java scripts have been used that do nearly nothing other than sleep because we did not want to stress test the Java engine, but rather the Jasmin Script MIB implementation.

Script MIB SNMP Operations

The first analysis is concerned with the times spent for typical operations on the Script MIB tables. These operations are documented as usage examples in chapter 7 of RFC 2592. We now call them "procedures" to avoid confusion with single SNMP operations, since some of these procedures require a number of SNMP operations and/or polling a particular variable until a certain value indicates the termination of asynchronous actions at the agent side.

The following procedures have been analyzed. The most interesting and frequently used procedures are shown in Figure 2.

- Installing a Script:
  This procedure issues three SNMP SetRequests to fill up a new row in the `smScriptTable` and to initiate the retrieval of a script from a given URL. Then the manager has to poll the `smScriptOperStatus` object until the value of `enabled` indicates that the script has been fetched successfully. This procedure is documented in section 7.2 of RFC 2592. The time spent in this procedure mainly depends on the HTTP or FTP infrastructure that is used to fetch scripts. This is the reason why we do not present this time in Figure 2. However, on our 100 MBit/s Ethernet LAN where a fairly loaded Sun Ultra-30 Model 295 serves the HTTP requests, we get installation times of approximately 230 ms for a very small script of 2.7 kBytes.

- Deleting a Script:
  This procedure first sets the `smScriptAdminStatus` to `disabled`. Then it polls `smScriptOperStatus` until it reflects this change. Finally, the row gets deleted by setting `smScriptRowStatus` to `destroy`. This procedure is documented in section 7.4 of RFC 2592. It takes approximately 88 ms to complete.

- Creating a Launch Button:
  This procedure creates, fills, and enables an entry in the `smLaunchTable` by sending three SNMP SetRequests. Then it has to poll `smLaunchOperStatus` until a value of `enabled` indicates the successful creation of the new launch button. This procedure takes approximately 95 ms to complete. It is documented in section 7.5 of RFC 2592.

- Deleting a Launch Button:
  This procedure sets the `smLaunchAdminStatus` to `disabled` and then polls `smLaunchOperStatus` until it reflects the change. Finally, the row gets deleted by setting `smLaunchRowStatus` to `destroy`. This procedure is documented in section 7.8 of RFC 2592 and takes approximately 19 ms to complete.

- Starting a Script:
  To initiate the start of a script, the manager sends a single SetRequest on an `smLaunchStart` object. We do not retrieve a free `smRunIndex` beforehand, as it is suggested as an alternative in section 7.6. However, afterwards we poll the `smRunState` object until a value of `executing` reflects that the script has been started successfully. This is required, because the SNMP response to the SetRequest just signals the successful creation of an `smRunTable` row. However, a script might not have been started due to a runtime system error. Note that this `smRunState` polling is not (yet) suggested in section 7.6 of RFC 2592. This procedure takes on average 1130 ms when the first script is launched, while subsequent scripts take approximately 74 ms to start. This reflects the fact that the Jasmin agent has to start the Java runtime engine along with the first script.

- Aborting, Suspending, and Resuming a Script:
  To abort a running script, the manager sets the `smRunControl` object to `abort`. Then it polls the `smRunState` object until the value of `terminated` reflects that the script has been aborted. Similarly, the manager writes the values `suspend` or `resume` to the `smRunControl` object and polls the `smRunState` object until the value of `suspended` or `executing` indicates the successful state change. Again, note that the polling, which is used to ensure that the procedure has been successful, is not (yet) documented in RFC 2592, section 7.7 on aborting a script. Suspending and resuming are not described at all in chapter 7. The time spent for these three procedures is around 40 ms.

- Retrieving State and Result of a Running Script:
  To retrieve the state and result of a running script, the manager simply issues a single GetRequest on `smRunState`, or `smRunResult`, respectively. This does not involve the running script in the Java runtime

engine. Instead, the Jasmin kernel just returns the values from its local copies of the script state and result variables. Hence, these procedures are very fast. They take around 6 ms and represent the minimum time needed for an SNMP operation on the Jasmin agent.

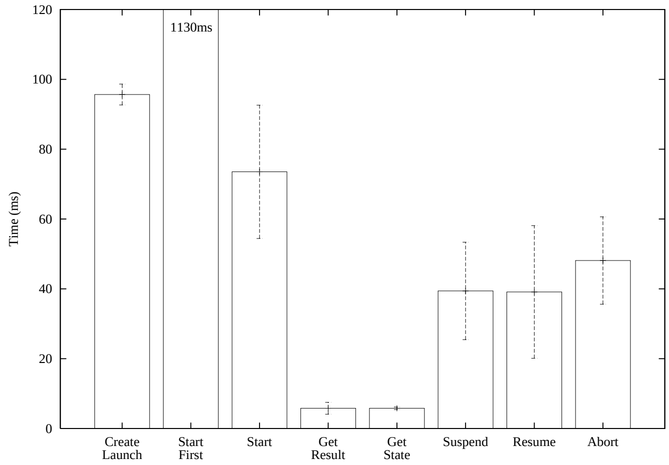

Figure 2: Duration of some procedures on the Script MIB. Error bars indicate the standard derivation.

SMX Operations on the Java Runtime Engine

The following analysis allows us to get an impression of the times spent in the Java runtime engine's operations by measuring the time between sending an SMX command and receiving the appropriate response back from the runtime engine. Furthermore, it gives us an estimate for the overhead induced by the SMX protocol. The results are shown in Figure 3.

- `hello` - 211: The `hello` command must be answered by the runtime engine with a 211 response to authenticate itself. This command is issued once at the startup of every runtime engine, so that its costs do not matter a lot. This command takes about 65 ms to complete.

- `start` - 231: The Jasmin agent uses the `start` command to launch a new script in the runtime engine. On success, the runtime engine sends back a 231 response. The time spent for this command is on average 42 ms.

- `suspend/resume` - 231: These commands can be used by the Jasmin kernel to suspend and resume a running script within the runtime engine. Both commands are answered with a 231 status response. They take about 17 ms to complete.

- `abort` - 232: To terminate a running script, the Jasmin kernel may send an `abort` command. On success, the runtime engine sends back a 232 response. This command takes on average 28 ms.

Intermediate status change and result information is sent asynchronously from the runtime engine to the Jasmin kernel, so that there are no command round-trip times to measure in this case.

Note that SMX commands are processed asynchronously by the Java runtime engine. Therefore, our test script paused between the stimulating operations, so that the runtime engine had to process at most one SMX command at any time.

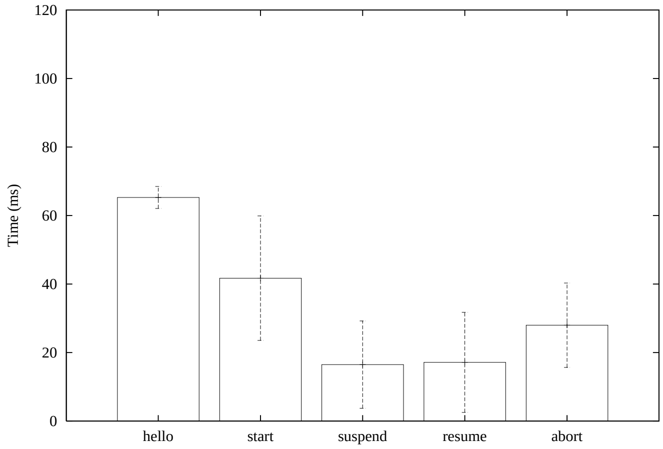

Figure 3: Duration of SMX operations on the Java runtime engine. Error bars indicate the standard derivation.

Scalability

In a final analysis we installed 200 instances of a single script in the `smScriptTable`, created 200 launch buttons in the `smLaunchTable` and started 200 instances to appear in the `smRunTable`. Although, these numbers might be of questionable significance in many environments, they uncovered two interesting points.

First, we have seen that the current Jasmin implementation does not scale well at high numbers of concurrently running scripts: The Jasmin kernel queries the runtime engine(s) for the status of every script at regular intervals. At about 130 scripts (that do nothing but sleep), the Java runtime engine was completely busy with answering SMX `status` commands. SMX commands trying to start further scripts timed out.

Second, the memory allocated by the Jasmin subagent did not increase significantly while the Script MIB tables have been filled up. It grew to not more than 2.5 MBytes. In constrast to the sub-agent, the Java runtime engine allocates approximately 6 MBytes at startup. Additionally, for every running script it allocates approximately 50 kBytes, even for our small test script that contains no notable variables or other runtime state information.

Conclusions

The most frequently used operations on the Jasmin Script MIB implementation perform reasonably well.

The startup time of the first script of any language is significantly longer than for subsequent scripts of the same language. This might be reduced by starting the runtime engine(s) at agent startup or at creation of a script or launch button of that language. Anyway, this is considered to be less important in most cases.

The status polling between the Jasmin sub-agent and the runtime system(s) for running scripts does not scale well. It should be considered to change the polling interval rules or to completely drop SMX status polling, since irregular situations can be signaled asynchronously by the runtime engine, anyway.

The runtime engine's memory consumption might be optimized for concurrently running instances of the same scripts.

In the usage examples of RFC 2592 some additional procedures and operations in procedures have been identified that would be of value to management application developers. They have been submitted to the DISMAN working group.

Practical Experiences with Script MIB Applications

Jürgen Quittek, NEC Europe Ltd.
Cornelia Kappler, NEC Europe Ltd.

This article reports on some experiences gained while building simple management applications with the IETF Script MIB. We discuss three applications with scripts acting as mid-level managers. Mid-level managers act in two roles: acting as agent they provide information to higher-level managers; acting as manager they access other agents. We built our applications with the Jasmin Script MIB implementation described already in the previous article in this issue of *The Simple Times*. This implementation provides a Java runtime environment for scripts, so all our scripts are written in Java.

Distributed Monitoring

Distributed monitoring is a fundamental application of distributed management. When the number of nodes to be monitored becomes too large to be handled by a single central manager, you distribute the work of monitoring among a set of mid-level managers, each of them monitoring a different subset of the nodes. For coordination and central processing of the monitoring results, each mid-level manager communicates with a top-level manager.

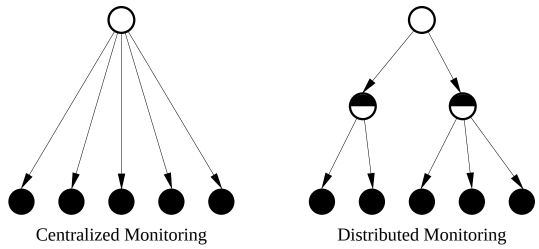

Figure 4: Centralized versus distributed monitoring.

This way, the scalability of a monitoring application can be increased significantly. We studied distributed monitoring with the Script MIB by writing a small test application. The intention was to monitor the system load on managed nodes in several subnets of a LAN. We installed one monitoring script in each subnet.

Our first observation was that, although we considered the problem to be small, our script was not. The problem we encountered was that the script needs to send SNMP requests to the set of nodes it is expected to monitor. Since Java itself does not have support for SNMP, the script needed to carry its own implementation of an SNMP protocol engine, if no specific support is offered by the runtime environment. This is not a problem for the developer, because appropriate class libraries are available, but the resulting size of the script is about 500 kB. We considered this size rather large, especially if you plan to run more than one script on the same node.

Our recommendation resulting from this experience is that a script runtime environment should provide an SNMP protocol stack which can be used by all scripts. The Script MIB already contains a table, the `smExtsnTable`, which informs the user about installed libraries or language extensions, such as a Java SNMP class library or the Scotty extension for Tcl. Our distributed monitoring script shrank from 500 kB to 5 kB after extending the runtime environment by an SNMP class library.

The second observation concerns the limitation of the communication facilities between the script and its manager. The Script MIB supports notifications, a result string, and an exit code. The result string can be written during script execution and at script termination. Each write action overwrites the previous value.

As output, our script produces a comma-separated list of host addresses and system load values, which is written to the result string once per monitoring interval. The top-level manager can poll this String whenever it wishes to do so. This procedure was acceptable for monitoring just one managed object per node. However, if we are going to extend monitoring to more objects, other communication facilities would be quiet helpful.

An obvious solution would be providing the monitoring results as a table of managed objects, but this requires either an SNMP agent implementation by the script itself, or other support, such as AgentX (RFC 2257). However, even if an SNMP stack is provided by the script runtime environment, we would prefer AgentX, because this results in less complex scripts.

Decentralized Active Service Testing

With a decentralized active service testing application we tested availability, response time and other parameters of a network service from a user's perspective. Different to tests performed at the server or at a network management station these tests show the parameters (quality) of the service as they are delivered to the user and not as they are available somewhere else. In order to do such tests, test routines must be executed at or close to the user's location, e.g. at an ISP's point of presence. We call service testing active when it simulates user behaviour rather than passively monitors transactions.

Together with the Technical University of Braunschweig, we developed a demonstrator illustrating a service testing scenario. This demonstrator measures the access time to a particular web server from several hosts in different subnets. At each host, a service testing script periodically measures the time it takes to download a specified page from the web server. The measured time is written to the result string in the Script MIB. The central manager regularly polls this string at all hosts and visualizes it on a map together with the location of the hosts and the web server.

As with distributed monitoring described above, we are limited by the fact that the Script MIB allows only for a single string as result of a script. When measuring a single parameter of a single service, the result string is sufficient, but we expect a real world service tester to perform a more complex task and deliver more complex results. Although it would be possible to start a single script for each parameter of each service, we would not recommend this solution for overcoming the limitations of the result string. Writing each result in an individual MIB object accessible via AgentX appears to be more appropriate. Therefore, we would recommend for SNMP agents implementing the Script MIB also to implement the AgentX protocol.

Decentralized active service testing can be applied in several other scenarios. We intend to examine some of them in the real world, collaborating with an ISP:

- An ISP offering Internet access might install scripts for testing the time needed for downloading a particular web page. This scenario is already prototyped by our demonstrator.
- On-line stores might ask their ISP to guarantee that a maximum processing time for customer's input will not be exceeded, in order to prevent the customers from clicking away to competitors. A script could test the current processing time from the user's perspective.
- An enterprise might reserve a VPN between local sales agents and a central data base containing customer and product information. It requires a guaranteed access time to this data base from all locations, that can be supervised by a script.
- Customers may want to measure and supervise the QoS they receive themselves. They could be provided with a GUI for starting scripts monitoring their QoS.

Distributed Service Management

This application deals with installing, starting, monitoring, and updating services on a managed node. Our test application manages the installation and configuration of an MPLS (Multiprotocol Label Switching, RFC 2702) multicast service over ATM. In our network, each MPLS core switch consists of an ATM switch and an attached Linux system. The MPLS service is realized by a GSMP control process running at the Linux system. Via the GSMP (General Switch Management Protocol) Version 1.1 (RFC 1987) the process controls the switch fabric of the ATM switch. All native ATM signaling is disabled.

MPLS standardization is not yet completed, although there are already MPLS product deployed commercially. The consequence is a need to update your system, whenever new standardization efforts result in a new release of the MPLS control software. Then you have to stop all running control processes, update and reconfigure the installation, and start the process again. These steps have to be made in a very similar way on all MPLS switches.

We tried to automate these steps. In a first stage of automation we developed shell scripts which performed the required steps. The software to be installed as well as the required configuration data was provided by a web server in the switches' control network. After having developed the scripts, updating the MPLS service was done by running

- a script checking the version currently installed,
- a stop script for the running MPLS control process,
- a de-install script for the old software release,
- an install script for the new release,
- a configure script for the new release,
- and a start script for the new MPLS control process.

In the second stage we automated running all these scripts in a coordinated way on all MPLS switches concerned. For this purpose we use the Script MIB. A central management application (written in Java) controls the execution of the scripts via a control network. The application allows the network operator to start the control processes on all switches by entering a single command. Similarly, all other actions described above can be executed simultaneously on all switches.

An inconvenience of this application was the restriction of our Script MIB implementation to Java as script language. We had to convert all our shell scripts to Java which was not a hard job. However, it appeared to be unnecessarily complicated, because for this purpose shell scripts are the better choice. We considered developing the scripts in Java in the first stage, but this appeared to be even more complicated than porting the scripts to Java. Testing this kind of scripts is less convenient in Java, because you have to compile them after each change and you have to wrap the commands by one or more `Runtime.exec()` calls. Please note that these inconveniences with Java appear when dealing with this kind of scripts. For other scripts, particularly more complex ones, Java might be a very good choice.

A strange problem we ran into when automating the software installation was the need to reboot the system, when the installation included kernel modifications. In such a situation we have to poll the booting system until it returnes to normal operation, and then continue with configuring or starting the service. Here, we would like to have a possibility to install a script such that it automatically starts its operation when the SNMP agent starts. Such a script could be installed just before reboot and complete the tasks without polling in between.

For this application, the Script MIB's result string was sufficient for reporting about successful completion of a script or for forwarding an error message. This is due to the nature of the tasks which - originally coded as shell scripts - do not return more than a string.

Conclusion

Summarizing our experiences, we found that there are several application scenarios in which the Script MIB is a valuable tool for distributing management tasks. Developing scripts in Java went very smoothly, although we did not like porting shell scripts. For future Script MIB implementations we would recommend supporting AgentX and providing an SNMP protocol stack in the script runtime environment. With AgentX support it will be possible to investigate also another kind of Script MIB application, namely agent extensions.

References

[1] White, M., Gudur, S., *An Overview of the AgentX Protocol*, *The Simple Times* 6(1), March 1998.

[2] Quittek, J., Kappler, C. *Remote Service Deployment on Programmable Switches with the IETF SNMP Script MIB*, Proc. DSOM '99, Zürich, pp. 135-147, Springer Verlag, 1999.

Standards Summary

Please consult the latest version of *Internet Official Protocol Standards*. As of this writing, the latest version is RFC 2500.

SMIv1 Data Definition Language

Full Standards:

- RFC 1155 - Structure of Management Information
- RFC 1212 - Concise MIB Definitions

Informational:

- RFC 1215 - A Convention for Defining Traps

SMIv2 Data Definition Language

Full Standards:

- RFC 2578 - Structure of Management Information
- RFC 2579 - Textual Conventions
- RFC 2580 - Conformance Statements

SNMPv1 Protocol

Full Standards:

- RFC 1157 - Simple Network Management Protocol

Proposed Standards:

- RFC 1418 - SNMP over OSI
- RFC 1419 - SNMP over AppleTalk
- RFC 1420 - SNMP over IPX

SNMPv2 Protocol

Draft Standards:

- RFC 1905 - Protocol Operations for SNMPv2
- RFC 1906 - Transport Mappings for SNMPv2
- RFC 1907 - MIB for SNMPv2
- RFC 1908 - SNMPv1 and SNMPv2 Coexistence

Experimental:

- RFC 1901 - Community-based SNMPv2
- RFC 1909 - Administrative Infrastructure
- RFC 1910 - User-based Security Model

SNMPv3 Protocol

Draft Standards:

- RFC 2571 - Architecture for SNMP Frameworks
- RFC 2572 - Message Processing and Dispatching
- RFC 2573 - SNMPv3 Applications
- RFC 2574 - User-based Security Model
- RFC 2575 - View-based Access Control Model
- RFC 1905 - Protocol Operations for SNMPv2
- RFC 1906 - Transport Mappings for SNMPv2
- RFC 1907 - MIB for SNMPv2

Informational:

- RFC 2570 - Introduction to SNMPv3

SNMP Agent Extensibility

Proposed Standards:

- RFC 2257 - AgentX Protocol Version 1

SMIv1 MIB Modules

Full Standards:

- RFC 1213 - Management Information Base II
- RFC 1643 - Ethernet-Like Interface Types MIB

Draft Standards:

- RFC 1493 - Bridge MIB
- RFC 1559 - DECnet phase IV MIB
- RFC 1757 - Remote Network Monitoring MIB

Proposed Standards:

- RFC 1285 - FDDI Interface Type (SMT 6.2) MIB
- RFC 1381 - X.25 LAPB MIB
- RFC 1382 - X.25 Packet Layer MIB
- RFC 1414 - Identification MIB
- RFC 1461 - X.25 Multiprotocol Interconnect MIB
- RFC 1471 - PPP Link Control Protocol MIB
- RFC 1472 - PPP Security Protocols MIB
- RFC 1473 - PPP IP NCP MIB
- RFC 1474 - PPP Bridge NCP MIB
- RFC 1512 - FDDI Interface Type (SMT 7.3) MIB
- RFC 1513 - RMON Token Ring Extensions MIB
- RFC 1514 - Host Resources MIB
- RFC 1515 - IEEE 802.3 MAU MIB
- RFC 1525 - Source Routing Bridge MIB
- RFC 1742 - AppleTalk MIB

SMIv2 MIB Modules

Draft Standards:

- 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 Type MIB
- RFC 1724 - RIP version 2 MIB

- RFC 1748 - IEEE 802.5 Interface Type MIB
- RFC 1850 - OSPF version 2 MIB
- RFC 1907 - SNMPv2 MIB
- RFC 2115 - Frame Relay DTE Interface Type MIB

Proposed Standards:

- RFC 1567 - X.500 Directory Monitoring MIB
- RFC 1604 - Frame Relay Service MIB
- RFC 1611 - DNS Server MIB
- RFC 1612 - DNS Resolver MIB
- RFC 1666 - SNA NAU MIB
- RFC 1696 - Modem MIB
- RFC 1697 - RDBMS MIB
- RFC 1747 - SNA Data Link Control MIB
- RFC 1749 - 802.5 Station Source Routing MIB
- RFC 1759 - Printer MIB
- RFC 2006 - Internet Protocol Mobility MIB
- RFC 2011 - Internet Protocol MIB
- RFC 2012 - Transmission Control Protocol MIB
- RFC 2013 - User Datagram Protocol MIB
- RFC 2020 - IEEE 802.12 Interfaces MIB
- RFC 2021 - RMON Version 2 MIB
- RFC 2024 - Data Link Switching MIB
- RFC 2037 - Entity MIB
- RFC 2051 - APPC MIB
- RFC 2074 - RMON Protocol Identifier
- RFC 2096 - IP Forwarding Table MIB
- RFC 2108 - IEEE 802.3 Repeater MIB
- RFC 2127 - ISDN MIB
- RFC 2128 - Dial Control MIB
- RFC 2206 - Resource Reservation Protocol MIB
- RFC 2213 - Integrated Services MIB
- RFC 2214 - Guaranteed Service MIB
- RFC 2232 - Dependent LU Requester MIB
- RFC 2233 - Interfaces Group MIB
- RFC 2238 - High Performance Routing MIB
- RFC 2248 - Network Services Monitoring MIB
- RFC 2249 - Mail Monitoring MIB
- RFC 2266 - IEEE 802.12 Repeater MIB
- RFC 2287 - System-Level Application Mgmt MIB
- RFC 2320 - Classical IP and ARP over ATM MIB
- RFC 2366 - Multicast over UNI 3.0/3.1 / ATM MIB
- RFC 2452 - IPv6 UDP MIB
- RFC 2454 - IPv6 TCP MIB
- RFC 2455 - APPN MIB
- RFC 2456 - APPN Trap MIB
- RFC 2457 - APPN Extended Border Node MIB
- RFC 2465 - IPv6 Textual Conventions and MIB
- RFC 2466 - ICMPv6 MIB
- RFC 2493 - 15 Minute Performance History TCs
- RFC 2494 - DS0, DS0 Bundle Interface Type MIB
- RFC 2495 - DS1, E1, DS2, E2 Interface Type MIB
- RFC 2496 - DS3/E3 Interface Type MIB
- RFC 2512 - Accounting MIB for ATM Networks
- RFC 2513 - Accounting Control MIB
- RFC 2514 - ATM Textual Conventions and OIDs
- RFC 2515 - ATM MIB
- RFC 2558 - SONET/SDH Interface Type MIB
- RFC 2561 - TN3270E MIB
- RFC 2562 - TN3270E Response Time MIB
- RFC 2564 - Application Management MIB
- RFC 2571 - SNMP Framework MIB
- RFC 2572 - SNMPv3 MPD MIB
- RFC 2573 - SNMP Applications MIBs
- RFC 2574 - SNMPv3 USM MIB
- RFC 2575 - SNMP VACM MIB

- RFC 2584 - APPN/HPR in IP Networks
- RFC 2591 - DISMAN Scheduling MIB
- RFC 2592 - DISMAN Script MIB
- RFC 2594 - WWW Services MIB
- RFC 2605 - Directory Server MIB
- RFC 2613 - RMON for Switched Networks MIB
- RFC 2618 - RADIUS Authentication Client MIB
- RFC 2619 - RADIUS Authentication Server MIB
- RFC 2667 - IP Tunnel MIB
- RFC 2662 - ADSL Line MIB
- RFC 2665 - Ethernet-Like Interface Types MIB
- RFC 2668 - IEEE 802.3 MAU MIB
- RFC 2669 - DOCSIS Cable Device MIB
- RFC 2670 - DOCSIS RF Interface MIB
- RFC 2677 - Next Hop Resolution Protocol MIB
- RFC 2720 - Traffic Flow Measurement Meter MIB

Informational:

- RFC 1628 - Uninterruptible Power Supply MIB
- RFC 2620 - RADIUS Accounting Client MIB
- RFC 2621 - RADIUS Accounting Server MIB
- RFC 2666 - Ethernet Chip Set Identifiers
- RFC 2707 - Print Job Monitoring MIB

IANA Maintained MIB Modules

- Interface Type Textual Convention
  ftp://ftp.iana.org/mib/ianaiftype.mib
- Address Family Numbers Textual Convention
  ftp://ftp.iana.org/mib/ianaaddressfamilynumbers.mib
- TN3270E Textual Conventions
  ftp://ftp.iana.org/mib/ianatn3270etc.mib
- Language Identifiers
  ftp://ftp.iana.org/mib/ianalanguage.mib

Related Documents

Informational:

- RFC 1270 - SNMP Communication Services
- RFC 1321 - MD5 Message-Digest Algorithm
- RFC 1470 - Network Management Tool Catalog
- RFC 2039 - Applicability of Standard MIBs to WWW Server Management
- RFC 2089 - Mapping SNMPv2 onto SNMPv1 within a bi-lingual SNMP agent

Experimental:

- RFC 1187 - Bulk Table Retrieval with the SNMP
- RFC 1224 - Techniques for Managing Asynchronously Generated Alerts
- RFC 1238 - CLNS MIB
- RFC 1592 - SNMP Distributed Program Interface
- RFC 1792 - TCP/IPX Connection MIB Specification
- RFC 2593 - Script MIB Extensibility Protocol

Calendar and Announcements

IETF Meetings:

- 46th Meeting of the IETF
  November 8-12, 1999, Washington, DC, USA
- 47th Meeting of the IETF
  March 27-31, 2000, Adelaide, Australia
- 48th Meeting of the IETF
  July 31- Aug 4, 2000, Pittsburgh, PA, USA
- 49th Meeting of the IETF
  December 11-15, 2000, San Diego, CA, USA
- 50th Meeting of the IETF
  March 19-23, 2001, Minneapolis, MN, USA

Conferences and Workshops:

- Network Operations and Management Symposium (NOMS 2000)
  April 10-14, 2000, Honolulu, Hawaii, USA
- Workshop on IP-oriented Operations & Management (IPOM 2000)
  September 4-6, 2000, Cracow, Poland
- Workshop on Distributed Systems Operations & Management 2000 (DSOM 2000)
  Austin, Texas, USA
- Integrated Network Management (IM 2001)
  May 14-18, 2001, Seattle, WA, USA

Exhibitions and Trade Shows:

- NetWorld + Interop Sydney
  November 15-19, 1999, Sydney, Australia
- NetWorld + Interop Las Vegas
  May 8-12, 2000, Las Vegas, USA
- NetWorld + Interop Atlanta
  September 25-29, 2000, Atlanta, USA

Publication Information

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 version 1.0. 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 HTML, PDF and PostScript. New issues are announced via an electronic mailing list. Send electronic mail to

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

with

```
subscribe simple-times
```

in the body if you want to subscribe to this list. Back issues are available via *The Simple Times* Web server:

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