SMX - Script MIB Extensibility Protocol Version 1.0

J. Schönwälder, M. Bolz, S. Mertens
Department of Computer Science
TU Braunschweig, Germany

J. Quittek, A. Kind, J. Nicklisch
NEC C&C Research Laboratories
Berlin, Germany

Informatik-Bericht 98-07, October 6, 1998

Abstract

The IETF Script MIB defines a standard MIB interface for the delegation of management functions based on the Internet management framework. A management script is a set of instructions that are executed by a language specific runtime system. The Script MIB Extensibility protocol (SMX) defined in this memo allows to separate language specific runtime systems from an language independent MIB implementation.

Contents

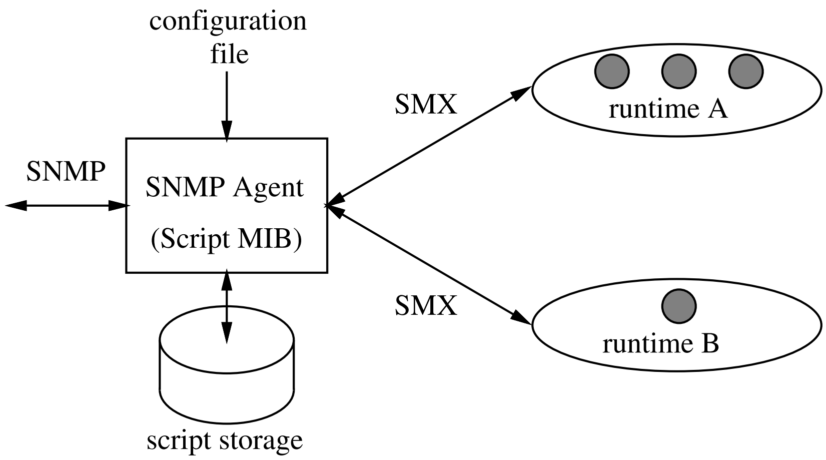
1 Introduction

The Script MIB [2] defines a standard interface for the delegation of management functions based on the Internet management framework. In particular, it provides the following capabilities:

1. Capabilities to transfer management scripts to a mid-level manager.
2. Capabilities for initiating, suspending, resuming and terminating management scripts.
3. Capabilities to transfer arguments for management scripts.
4. Capabilities to monitor and control running management scripts.
5. Capabilities to transfer the results produced by running management scripts.

A management script is a set of instructions written in an executable language. Scripts are executed by a language specific runtime system. The MIB does not prescribe a specific language. Instead, it allows to control scripts written in different languages concurrently.

The purpose of the Script MIB Extensibility protocol (SMX) defined in this memo is to separate language specific runtime systems from the MIB implementation. The goal is to define a lightweight protocol which allows to plugin different runtime systems without any changes to the Script MIB implementation. Figure 1 shows the overall structure.



Examples of language runtime systems considered during the design of SMX are the Java virtual machine [4] and the Tool Command Language interpreter [5, 3]. Other languages with comparable features should be easy to integrate as well.

2 Process Model

The SNMP agent starts one or more runtime systems. A runtime system may be capable to execute multiple scripts simultaneously (multi-threading) or it may be limited to exactly one script. The protocol between the SNMP agent and the runtime system must support multi-threaded runtime system.

Some operating systems have a notion of a operating system security profiles that can be used to restrict the access of a running process to the resources controlled by the operating system. In UNIX systems, the operating system security profile is associated with the effective user id of a process. The SNMP agent should allow to map SNMP user identities (owner strings) to operating system security profiles. This means that two invocations of the same single script might be executed by two separate runtime systems that are running under different operating system security profiles. It is the job of the SNMP agent to assign script invocations to a suitable runtime system and to start or stop runtime systems as needed.

3 Start of Runtime Systems

The SNMP agent starts runtime systems based on the static properties of the runtime system (multi-threaded or single-threaded) and the operating system security profiles obtained from a local configuration file and the SNMP identity. The SNMP agent might in addition choose to create a special process environment for the runtime systems (e.g. using chroot to restrict access to the local file system).

The SNMP agent passes information to the runtime system by means of environment variables. The information is needed to establish a communication channel between the SNMP agent and a runtime system and to pass security cookies around. The security assumption here is that access to the process environment is protected by the operation system, which is the case on a reasonably maintained UNIX system.

4 Connection Establishment

The SNMP agent creates a listening TCP socket which accepts connections from runtime systems. It is the responsibility of the runtime system to establish a connection to this TCP socket. The port number of the listening TCP socket is obtained from the environment variable named `SMX_PORT`.

The SNMP agent must ensure that only authorized runtime systems establish a connection. The following rules are used for this purpose:

- The TCP connection must come from the local host.
- The SNMP agent queries the runtime system for a security cookie and closes the TCP connection if no valid response is received within a given time interval. The security cookie is a random number generated by the SNMP agent and passed to the runtime system as part of its environment. The cookie is found in the variable `SMX_COOKIE`.

Alternate transports (e.g. UNIX domain sockets) are possible but not supported at this point in time. The reason to choose TCP connections was that TCP connections are supported by all runtime systems under discussion so far, while other transports are not universally available.

5 Messages

There are two classes of messages. The first class consists of messages that are directed to the runtime system as a whole. The second class consists of messages that are directed to a particular running script within a runtime system.

The message formats described below are defined using the Augmented BNF defined in RFC 2234 [1]. The definitions for `ALPHA`, `DIGIT`, `HEXDIG`, `WSP`, `CRLF`, `CR`, `LF`, `HTAB` and `DQUOTE` are imported from appendix A of RFC 2234 and not repeated here.

5.1 General Definitions

The following definitions are used below to define the protocol messages.

```
Zero = %x30
AlNum = DIGIT / ALPHA / %x2D-2F ; digits, alphas plus '-', '.', '/'
QuotedString = DQUOTE (VCHAR / WSP) DQUOTE
HexString = 1*(HEXDIG HEXDIG)   ; sequence of pairs of hexdigits
Id = 1*DIGIT                    ; identifier for an SMX transaction pair
Script = QuotedString           ; a script file name
RunId = 1*DIGIT                 ; a number identifying a running script
Profile = 1*AlNum               ; a security profile name
ExitCode = 1*DIGIT              ; see MIB definition of smRunExitCode
RunState = 1*DIGIT              ; see MIB definition of smRunState
Cookie = HexString              ; authentication cookie
VersionString = "SMX/1.0"       ; current version of the SMX protocol
```

```
Argument = HexString / QuotedString
Result = HexString / QuotedString
ErrorMsg = HexString / QuotedString
```

The definition of `QuotedString` requires further explanation. A quoted string may contain special character sequences, all starting with the backslash character (`%x5C`). The interpretation of these sequences is as follows:

`\\`	Backslash character (`%x5C`)
`\t`	Tab character (`HTAB`)
`\n`	Newline character (`LF`)
`\r`	Carriage-return character (`CR`)
`\"`	Quote character (`DQUOTE`)

In all cases not listed above, the backslash is dropped and the following character is treated as an ordinary character.

The `Argument` and the `Result` are either a `QuotedString` or a binary string where each byte is encoded as two hexadecimal digits. The MIB definition defines script arguments and results as arbitrary octet strings. However, it is likely that printable argument and result strings will be used frequently. The quoted string format allows a human readable representation in this case. However, an implementation must be able to handle both formats.

The `Cookie` is a sequence of bytes where each byte is encoded as two hexadecimal digits. The cookie does not carry any semantics other than being a random sequence of bytes. It is therefore not necessary to have a human readable representation.

`ExitCode` and `RunState` are restricted to the values defined in the Script MIB [2]. The table below lists the enumerated values together with the associated label:

RunState	
1	initializing
2	executing
3	suspending
4	suspended
5	resuming
6	aborting
7	terminated

ExitCode	
1	noError
2	halted
3	lifeTimeExceeded
4	noResourcesLeft
5	languageError
6	runtimeError
7	invalidArgument
8	securityViolation
9	genericError

5.2 Commands

The following list of commands can be send from the SNMP agent to a runtime system.

```
Command =  "hello"   WSP Id CRLF
Command =/ "start"   WSP Id WSP RunId WSP Script WSP Profile WSP Argument CRLF
Command =/ "suspend" WSP Id WSP RunId CRLF
Command =/ "resume"  WSP Id WSP RunId CRLF
Command =/ "abort"   WSP Id WSP RunId CRLF
Command =/ "status"  WSP Id WSP RunId CRLF
```

The `hello` command is always the first command send over a connection. It is used to identify and authenticate the runtime system. The `load` command implies to load a script into the runtime system. There is no separate compile command as it is the responsibility of the SNMP agent to perform any compilation steps as needed. There is no separate command to shutdown a runtime system. Closing the connection must be interpreted as a request to shutdown the runtime system.

5.3 Replies

Every reply message starts with a three digit reply code and ends with CRLF. The three digits in a reply code have a special significance. The first digit denotes whether the response is good, bad or an asynchronous notification. The following classes exist:

1yz	Transient Positive Response
2yz	Permanent Positive Response
3yz	Transient Negative Response
4yz	Permanent Negative Response
5yz	Asynchronous Notification

The second digit encodes the specific category. The following categories exist:

x0z	Syntax errors that don't fit any other category.
x1z	Replies for commands that are targeted at the whole runtime system.
x2z	Replies for commands that are targeted at scripts.
x3z	Replies for commands that are targeted at running scripts.

The third digit gives a finer gradation of meaning in each category specified by the second digit. Below is the complete list of all reply codes:

211	Identification of the runtime system.
231	Status of a running script.
232	Abort of a running script.
401	Syntax error in command.
402	Unknown command.
421	Unknown or illegal script name.
431	Unknown or illegal running script id.
432	Unknown or illegal profile name.
433	Illegal script argument.
434	Unable to change the status of a running script.
511	An arbitrary message send from the runtime system.
531	Notification of asynchronous running script status change.
532	Notification of intermediate script results.
533	Notification of intermediate script results that trigger an event report.
534	Notification of asynchronous normal script termination.
535	Notification of asynchronous abnormal script termination.

The ABNF definition of all reply messages and codes:

```
Reply =  "211" WSP Id WSP VersionString WSP Cookie CRLF
Reply =/ "231" WSP Id WSP RunState CRLF
Reply =/ "232" WSP Id CRLF
Reply =/ "401" WSP Id CRLF
Reply =/ "402" WSP Id CRLF
Reply =/ "421" WSP Id CRLF
Reply =/ "431" WSP Id CRLF
Reply =/ "432" WSP Id CRLF
Reply =/ "433" WSP Id CRLF
Reply =/ "434" WSP Id CRLF
Reply =/ "511" WSP Zero WSP QuotedString CRLF
Reply =/ "531" WSP Zero WSP RunId WSP RunState CRLF
Reply =/ "532" WSP Zero WSP RunId WSP RunState WSP Result CRLF
Reply =/ "533" WSP Zero WSP RunId WSP RunState WSP Result CRLF
Reply =/ "534" WSP Zero WSP RunId WSP Result CRLF
Reply =/ "535" WSP Zero WSP RunId WSP ExitCode WSP ErrorMsg CRLF
```

6 Elements of Procedure

This section describes in more detail the processing steps performed by the SNMP agent and the runtime system with regard to the SMX protocol.

6.1 SMX Message Processing on the Runtime Systems

This section describes the responses of the runtime engine on SMX commands (6.1.1 - 6.1.6) and the conditions under which asynchronous notifications are generated by the runtime system (6.1.7).

When the runtime system receives a message, it first tries to recognize a command consisting of command string and transaction identifier. The message is discarded if the runtime system is not able to extract the command string and transaction identifier. Otherwise, the command string is checked to be valid, i.e. to be one of the strings `hello`, `start`, `suspend`, `resume`, `abort`, and `status`. If the string is not valid, a `402` reply is sent. Otherwise, if a syntax error in the message is detected, a `401` reply is sent and processing of the message is stopped. If a valid command has been detected, further processing of the message depends on the command. The subsequent actions performed by the runtime system are listed by the command.

6.1.1 Processing the 'hello' Command

When the runtime system receives a `hello` command, it processes it as follows:

1. The runtime system obtains the security cookie from the process environment.
2. The runtime system sends a `211` reply, which contains the security cookie.

6.1.2 Processing the 'start' Command

When the runtime system receives a `start` command, it processes it as follows:

1. The syntax of the arguments of the `start` command is checked. The four checks listed in the following can be executed in arbitrary order.
   - The syntax of the run identifier is checked and a `431` reply is sent, if any syntax error is detected.
   - The syntax of the script name is checked and a `421` reply is sent, if any syntax error is detected.
   - The syntax of the security profile is checked and a `432` reply is sent, if any syntax error is detected.
   - If syntax of the script argument is checked and a `433` reply is sent, if any syntax error is detected.
2. The runtime system checks, whether the new `RunId` is already in use. If yes, a `431` reply is sent and processing stops.
3. The runtime system checks, whether the `Script` parameter is the name of a file on a local file system, that can be read. A `421` reply is sent and processing stops, if not.
4. The runtime system checks, whether the security profile is known and sends a `432` reply and stops processing if not.
5. The runtime engine starts the script given by the script name. When the script has been started, a `231` reply is sent including the current state of the run.

   Processing of the `start` command stops, when the script reaches the state `running`. For each required change of the state of the running script, a `531` reply is sent. Processing of the `start` command is also stopped, if an error occurs before the state `running` is reached. In this case, the run is aborted and a `535` reply is generated.

   If an `abort` command or a `suspend` command for the running script is received before processing of the `start` command is finished, processing of the `start` command may be stopped before the state `running` is reached. In this case, the resulting status of the running script is given by the respective reply to the `abort` command or a `suspend` command.

6.1.3 Processing the 'suspend' Command

When the runtime system receives a 'suspend' command, it processes it as follows:

1. If there is a syntax error in the running script identifier or if there is no running script matching the identifier, a `431` reply is sent and processing of the command is stopped.
2. If the running script is already in the state `suspended`, a `231` reply is sent and processing of the command is stopped.

3. If the running script is in the state `running`, it is suspended and a `231` reply is sent after suspending. If suspending fails, a `434` reply is sent and processing of the command is stopped.
4. If the running script has not yet reached the state `running` (the `start` command still being processed), it may reach the state `suspended` without having been in the state `running`. After reaching the state `suspended`, a `231` reply is sent.
5. If the running script is already aborted, a `434` reply is sent.

6.1.4 Processing the 'resume' Command

When the runtime system receives a 'resume' command, it processes it as follows:

1. If there is a syntax error in the running script identifier or if there is no running script matching the identifier, a `431` reply is sent and processing of the command is stopped.
2. If the running script is already in the state `running`, a `231` reply is sent and processing of the command is stopped.
3. If the running script is in the state `suspended`, it is resumed and a `231` reply is sent after resuming. If resuming fails, a `434` reply is sent and processing of the command is stopped.
4. If the `start` command is still being processed for the script, a `231` reply is sent, when the state `running` has been reached.
5. If the running script is already aborted, a `434` reply is sent.

6.1.5 Processing the 'abort' Command

When the runtime system receives a `abort` command, it processes it as follows:

1. If there is a syntax error in the running script identifier or if there is no running script matching the identifier, a `431` reply is sent and processing of the command is stopped.
2. If the running script is already aborted, a `232` reply is sent and processing of the command is stopped.
3. The running script is aborted and a `232` reply is sent after aborting. If aborting fails, a `434` reply is sent and processing is stopped.

6.1.6 Processing the 'status' Command

When the runtime system receives a 'status' command, it processes it as follows:

1. If there is a syntax error in the running script identifier or if there is no running script matching the identifier, a `431` reply is sent and processing of the command is stopped.
2. The status of the script is obtained and a `231` reply is sent.

6.1.7 Generation of Asynchronous Notifications

The runtime system generates or may generate the following notifications:

- If a change of the status of a running script is observed by the runtime system, a `531` reply is sent.
- A `534` reply is sent if a running script terminates normally.
- A `535` reply is sent if a running script terminates abnormally.
- If a script generates an intermediate result without changing its state, a `532` reply is sent.
- If a script requests a notification to be send to SNMP manager(s), a `533` reply is sent.
- Besides the notifications mentioned above, the runtime system may generate arbitrary `511` replies, which are logged or displayed by the SNMP agent.

6.2 SMX Message Generation and Processing on the SNMP Agent

This section describes the conditions under which an SNMP agent implementing the Script MIB generates SMX commands. It also describes how the SNMP agent processes replies to SMX commands.

6.2.1 Creating a Runtime System

New runtime systems are started by the SNMP agent while processing set-requests for a `smLaunchStart` variable. The SNMP agent first searches for an already running runtime systems which matches the security profiles associated with `smLaunchStart`. If no suitable runtime system is available, a new runtime system is started by preparing the environment for the new runtime system and starting the executable for the runtime system in a new process which conforms to the operating system security profile. The SNMP agent prepares to accept a connection from the new runtime system. The `smRunState` of all scripts that should be executed in this new runtime system is set to `initializing`.

6.2.2 Generating the 'hello' Command

The `hello` command is generated once a connection request from a runtime system has been accepted. The SNMP agent sends the `hello` command as defined in section 5.2. The SNMP agent then expects a reply from the runtime system within a reasonable timeout interval.

1. If the timeout expires before the SNMP agent received a reply, the connection is closed and all data associated with it is deleted.
2. If the received message can not be analyzed because it does not have the required format, then the connection is closed and all data associated with it is deleted. Any scripts that should be running in this runtime system are aborted, the `smRunExitCode` is set to `genericError` and `smRunResult` is modified to contain a human readable message describing the error situation.
3. If the received message is a `211` reply, then the `Id` is checked whether it matches the `Id` used in the `hello` command. If the `Id` matches, then the `VersionString` is checked. If the `VersionString` matches a supported SMX protocol version, then the `Cookie` is checked whether it matches the cookie passed to the runtime system. If any of these tests fails, then the connection is closed and all data associated with this runtime system is deleted.
4. Received messages are discarded if none of the previous rules applies.

6.2.3 Generating the 'start' Command

The `start` command is generated while processing set-requests for a `smLaunchStart` variable. The `start` command assumes that the SNMP agent already determined a runtime system suitable to execute the script associated with the `smLaunchStart` variable. The SNMP agent sends the `start` command as defined in section 5.2 to the selected runtime system. The SNMP agent then expects a reply from the runtime system within a reasonable timeout interval.

1. If the timeout expires before the SNMP agent received a reply, then the SNMP agent sends an `abort` command to abort the running script and sets the `smRunState` of the running script to `terminated`, the `smRunExitCode` to `genericError` and `smRunResult` is modified to contain a human readable message describing the timeout situation.
2. If the received message can not be analyzed because it does not have the required format, then the message is ignored. The SNMP agent continues to wait for a valid reply message until the timeout expires.
3. If the received message is a 4yz reply, then the SNMP agent assumes that the script can not be started. The `smRunState` of the running script is set to `terminated`, the `smRunExitCode` to `genericError` and the `smRunResult` is modified to contain a human readable message describing the error situation.

4. If the received message is a 231 reply and the Id matches an Id of a previous start command, then the smRunState variable of the running script is updated.
5. Received messages are discarded if none of the previous rules applies.

6.2.4 Generating the 'suspend' Command

The suspend command is generated while processing set-requests for the smLaunchControl and smRunControl variables which change the value to suspend. The SNMP agent sets the smRunState variable to suspending and sends the suspend command as defined in section 5.2. The SNMP agent then expects a reply from the runtime system within a reasonable timeout interval.

1. If the timeout expires before the SNMP agent received a reply, then the SNMP agent sends an abort command to abort the running script and sets the smRunState of the running script to terminated, the smRunExitCode to genericError and smRunResult is modified to contain a human readable message describing the timeout situation.
2. If the received message can not be analyzed because it does not have the required format, then the message is ignored. The SNMP agent continues to wait for a valid reply message until the timeout expires.
3. If the received message is a 401, 402 or a 431 reply, then the runtime systems is assumed to not provide the suspend/resume capability and processing of the suspend command stops.
4. If the received message is a 231 reply and the Id matches an Id of a previous suspend command, then the smRunState variable of the running script is updated.
5. Received messages are discarded if none of the previous rules applies.

6.2.5 Generating the 'resume' Command

The resume command is generated while processing set-requests for the smLaunchControl and smRunControl variables which change the value to resume. The SNMP agent sets the smRunState variable to resuming and sends the resume command as defined in section 5.2. The SNMP agent then expects a reply from the runtime system within a reasonable timeout interval.

1. If the timeout expires before the SNMP agent received a reply, then the SNMP agent sends an abort command to abort the running script and sets the smRunState of the running script to terminated, the smRunExitCode to genericError and smRunResult is modified to contain a human readable message describing the timeout situation.
2. If the received message can not be analyzed because it does not have the required format, then the message is ignored. The SNMP agent continues to wait for a valid reply message until the timeout expires.
3. If the received message is a 401, 402 or a 431 reply, then the runtime systems is assumed to not provide the suspend/resume capability and processing of the resume command stops.
4. If the received message is a 231 reply and the Id matches an Id of a previous resume command, then the smRunState variable of the running script is updated.
5. Received messages are discarded if none of the previous rules applies.

6.2.6 Generating the 'abort' Command

The abort command is generated while processing set-requests for the smLaunchControl and smRunControl variables which change the value to abort. In addition, the abort command is also generated if the smRunLifeTime variable reaches the value 0. The SNMP agent sends the abort command as defined in section 5.2. The SNMP agent then expects a reply from the runtime system within a reasonable timeout interval.

1. If the timeout expires before the SNMP agent received a reply, then the SNMP agent sets the smRunState of the running script to terminated, the smRunExitCode to genericError and smRunResult is modified to contain a human readable message describing the timeout situation.
2. If the received message can not be analyzed because it does not have the required format, then the message is ignored. The SNMP agent continues to wait for a valid reply message until the timeout expires.
3. If the received message is a 4yz reply, then the SNMP agent assumes that the script can not be aborted. The smRunState of the running script is set to terminated, the smRunExitCode to genericError and the smRunResult is modified to contain a human readable message describing the error situation.
4. If the received message is a 232 reply and the Id matches an Id of a previous abort command, then the smRunExitCode variable of the terminated script is changed to either halted (when processing a set-request for the smLaunchControl and smRunControl variables) or lifeTimeExceeded (if the abort command was generated because the smRunLifeTime variable reached the value 0). The smRunState variable is changed to the value terminated.
5. Received messages are discarded if none of the previous rules applies.

6.2.7 Generating the 'status' Command

The status command is generated either periodically or on demand by the SNMP agent in order to get current status information about the running scripts. The SNMP agent sends the status command as defined in section 5.2. The SNMP agent then expects a reply from the runtime system within a reasonable timeout interval.

1. If the timeout expires before the SNMP agent received a reply, then the SNMP agent sends an abort command to abort the running script and sets the smRunState of the running script to terminated, the smRunExitCode to genericError and smRunResult is modified to contain a human readable message describing the timeout situation.
2. If the received message can not be analyzed because it does not have the required format, then the message is ignored. The SNMP agent continues to wait for a valid reply message until the timeout expires.
3. If the received message is a 4xy reply, then the SNMP agent assumes that the script status can not be read, which is a fatal error condition. The SNMP agent sends an abort command to abort the running script. The smRunState of the running script is set to terminated, the smRunExitCode to genericError and the smRunResult is modified to contain a human readable message describing the error situation.
4. If the received message is a 231 reply and the Id matches an Id of a previous status command, then the smRunState variable of the running script is updated.
5. Received messages are discarded if none of the previous rules applies.

6.2.8 Processing asynchronous notifications

The runtime system can send asynchronous status change notifications. These 5yz replies are processed as described below.

1. If the received message is a 511 reply, then the message is displayed or logged appropriately and processing stops.
2. If the received message is a 531 reply, then the SNMP agent checks whether a running script with the given RunId exists in the runtime system. Processing of the notification stops if there is no running script with the smRunId. Otherwise, smExitCode and smRunState are updated. The smRunResult variable is updated if the optional Result field is present in the 531 reply.
3. If the received message is a 532 reply, then the SNMP agent checks whether a running script with the given RunId exists in the runtime system. Processing of the notification stops if there is no running script with the smRunId. Otherwise, smExitCode, smRunState and smRunResult are updated.

4. If the received message is a `533` reply, then the SNMP agent checks whether a running script with the given `RunId` exists in the runtime system. Processing of the notification stops if there is no running script with the `smRunId`. Otherwise, `smExitCode`, `smRunState` and `smRunResult` are updated and an SNMP `smScriptResult` notification is generated.
5. If the received message is a `534` reply, then the SNMP agent checks whether a running script with the given `RunId` exists in the runtime system. Processing of the notification stops if there is no running script with the `smRunId`. Otherwise, `smExitCode` is set to `noError`, `smRunState` is set to `terminated` and the `smRunResult` is updated.
6. If the received message is a `535` reply, then the SNMP agent checks whether a running script with the given `RunId` exists in the runtime system. Processing of the notification stops if there is no running script with the `smRunId`. Otherwise, `smRunState` is set to `terminated` and `smExitCode` and `smRunError` are updated.

7 An Example SMX Message Flow

Below is an example which shows how SMX message exchanges might happen in practice. Messages send from the SNMP agent are marked with > while replies send from the runtime system are marked with <. The line terminators (`CRLF`) are not shown in order to make the example more readable.

```
> hello 1
< 211 1 SMX/1.0 0AF0BAED6F877FBC
> start 2 42 "/var/snmp/scripts/foo.jar" untrusted ""
> start 5 44 "/var/snmp/scripts/bar.jar" trusted "www.ietf.org"
< 231 2 2
> start 12 48 "/var/snmp/scripts/foo.jar" funny ""
< 231 5 2
< 532 0 44 2 "waiting for response"
> status 18 42
> status 19 44
< 432 12
< 231 19 2
< 231 18 2
> hello 578
< 211 578 SMX/1.0 0AF0BAED6F877FBC
> suspend 581 42
< 231 581 4
< 534 0 44 "test completed"
> abort 611 42
< 232 611
```

8 Security Considerations

The security aspects of a scripting engine need to be split into different views in order to handle them. We will first discuss the security aspects of the SMX protocol followed by a discussion of security profiles.

8.1 Security of the SMX Protocol

The SMX protocol runs on top of local TCP connections. Protocol messages therefore never leave the local system. It is therefore not possible to attack the message exchanges, if the underlying operating system protects local TCP connections from users on the local machine.

The only critical part is connection establishment phase. The rules defined in section 4 ensure that only local connections are accepted. A runtime system has to identify itself with a security cookie generated by the SNMP agent and passed to the runtime system process as part of its environment. This rule ensures that scripts will only be executed on authorized runtime systems. This scheme relies on the protection of process environments by the operating system. Well maintained UNIX operating systems have this capability.

8.2 Security Profiles

Security profiles control what a running script is allowed to do:

- The *SNMP security profile* specifies the set of SNMP targets known to the running script and the SNMP security credentials that can be used by a running script. It is currently unclear whether the DISMAN working group will define a standard way to describe SNMP security profiles or whether this will be implementation dependent and part of the runtime security profile.
- The *operating system security profile* specifies the set of operating system services that can be used by the operating system level process which executes a script. Under UNIX, this maps to the effective UNIX user identity for the running process. In addition, some UNIX versions allow to set other resource limit parameters, like the number of open files or the maximum stack sizes. An additional security mechanism in UNIX allows to change the file system root for the runtime system, which allows to hide system files. It is suggested to make use of all applicable operating system security mechanism in order to protect the operating system from malicious scripts.
- Secure runtime systems allow to further control the security properties by specifying which set of services of the runtime system can be used at a particular point of script execution. The *runtime security profile* is runtime system dependent and allows much finer granularity than the operating system security profile. For a Java virtual machine, the runtime security profile maps to the Java SecurityManager class. For Tcl, the runtime security profile maps to the interpreter's security profile.

A reasonable secure implementation of the Script MIB must support the operating system security profile and the runtime security profile by providing mappings from the SNMP access control function to these profiles. Configuration directives must be defined which map the script owner or the script invoker to the security profiles.

References

[1] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 2234, Internet Mail Consortium, Demon Internet Ltd., November 1997.

[2] D. B. Levi and J. Schönwälder. Definitions of Managed Objects for the Delegation of Management Scripts. Internet Draft <draft-ietf-disman-script-mib-06.txt>, SNMP Research, TU Braunschweig, October 1998.

[3] J. Y. Levy, L. Demailly, J. K. Ousterhout, and B. Welch. The Safe-Tcl Security Model. In *Proc. USENIX Annual Technical Conference*, June 1998.

[4] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification.* Addison Wesley, 1997.

[5] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, April 1994.