Network Working Group
Request for Comments: 2592
Category: Standards Track

D. Levi
Nortel Networks
J. Schoenwaelder
TU Braunschweig

May 1999

Definitions of Managed Objects for the Delegation of Management Scripts

Status of this Memo

Copyright Notice

Abstract

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes a set of managed objects that allow the delegation of management scripts to distributed managers.

Table of Contents

1. Introduction

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes a set of managed objects that allow the delegation of management scripts to distributed managers.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [21].

2. The SNMP Management Framework

The SNMP Management Framework presently consists of five major components:

- An overall architecture, described in RFC 2271 [1].
- Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16, RFC 1155 [2], STD 16, RFC 1212 [3] and RFC 1215 [4]. The second version, called SMIv2, is described in STD 58, RFC 2578 [5], RFC 2579 [6] and RFC 2580 [7].

- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [8]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [9] and RFC 1906 [10]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [10], RFC 2272 [11] and RFC 2274 [12].

- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [8]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [13].

- A set of fundamental applications described in RFC 2273 [14] and the view-based access control mechanism described in RFC 2275 [15].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

MIB conforming to the SMIv1 can be produced through the appropriate translations. The resulting translated MIB must be semantically equivalent, except where objects or events are omitted because no translation is possible (use of Counter64). Some machine readable information in SMIv2 will be converted into textual descriptions in SMIv1 during the translation process. However, this loss of machine readable information is not considered to change the semantics of the MIB.

3. Overview

The Script MIB module defined in this memo can be used to delegate management functions to distributed managers. Management functions are defined as management scripts written in a management scripting language. This MIB makes no assumptions about the language itself and even allows distribution of compiled native code, if an implementation is able to execute native code under the control of this MIB.

The Script MIB 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 distributed 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.

This memo does not address any additional topics like the generation of notifications or how to address remote agents from a Script MIB implementation.

3.1. Terms

This section defines the terms used throughout this memo.

- A ‘distributed manager’ is a processing entity which is capable of performing network management functions. For the scope of this memo, a distributed manager is assumed to implement the Script MIB.

- A ‘higher-level manager’, or just ‘manager’, is a processing entity or human who initiates and controls the operations performed by one or more distributed managers.

- A ‘management script’ is a set of instructions written in an executable language which implements a management function.

- A ‘management scripting language’ is a language used to write management scripts. Note, the term scripting language does not imply that the language must have the characteristics of scripting languages (e.g. string orientation, interpretation, weak typing). The MIB defined in this memo also allows to control management scripts written in arbitrary compiled system programming languages.

- A ‘distributed manager’ can be decomposed into an ‘SNMP entity’ which implements the Script MIB defined in this memo and the ‘runtime system’ that executes scripts. The Script MIB sees the runtime system as the managed resource which is controlled by the MIB.

  The runtime system can act as an SNMP application, according to the SNMP architecture defined in RFC 2271 [1]. For example, a runtime system which sends SNMP requests to other SNMP entities will act as a command generator application. The SNMP

applications in the runtime system may use the same SNMP engine which also serves the command responder application used to implement the Script MIB, but they are not required to do so.

- A ‘launch button’ is the conceptual button used to start the execution of a management script. It assignes control parameters to a management script. In particular, it defines the ownership of the scripts started from a launch button. The ownership can be used by the language runtime system to enforce security profiles on a running management script.

4. Requirements and Design Issues

This section discusses some general requirements that have influenced the design of the Script MIB.

- The Script MIB must not make any assumptions about specific languages or runtime systems.
- The Script MIB must provide mechanisms that help to avoid new management problems (e.g. script version problems).
- The Script MIB must provide SNMP interfaces to all functions required to delegate management scripts. However, other protocols might be used in addition if they provide a significant improvement in terms of convenience for implementation or performance.
- The Script MIB must be organized so that access can be controlled effectively by using view-based access control [15].

The following sections discuss some design issues in more detail.

4.1. Script Languages

The Script MIB defined in this memo makes no assumption about the script language. This MIB can therefore be used in combination with different languages (such as Tcl or Java) and/or different versions of the same language. No assumptions are made about the format in which management scripts are transferred.

The Script MIB provides access to information about the language versions supported by a Script MIB implementation so that a manager can learn about the capabilities provided by an implementation. Languages and language versions are identified as follows:

1. The language is identified by an object identifier. Object identifier for well-known languages will be registered by the Internet Assigned Numbers Authority (IANA). Enterprise specific languages can also be registered in the enterprise specific OID subtree.

2. A particular version of a language is identified by a language version number. The combination of a language object identifier and a language version is in most cases sufficient to decide whether a script can be executed or not.

3. Different implementations of the same language version might have differences due to ambiguities in the language definition or additional language features provided by an implementor. An additional object identifier value is provided which identifies the organization which provides the implementation of a language. This might be used by scripts that require a particular implementation of a language.

4. Finally, there might be different versions of a language implementation. A version number for the language implementation is provided so that the manager can also distinguish between different implementations from the same organization of a particular language version.

The version numbers can either be used by a manager to select the language version required to execute a particular script or to select a script that fits the language versions supported by a particular Script MIB implementation.

An additional table lists language extensions that provide features not provided by the core language. Language extensions are usually required to turn a general purpose language into a management language. In many cases, language extensions will come in the form of libraries that provide capabilities like sending SNMP requests to remote SNMP agents or accessing the local MIB instrumentation. Every extension is associated with a language and carries its own version numbers.

4.2. Script Transfer

There are two different ways to transfer management scripts to a distributed manager. The first approach requires that the manager pushes the script to the distributed manager. This is therefore called the ‘push model’. The second approach is the ‘pull model’ where the manager tells the distributed manager the location of the script and the distributed manager retrieves the script itself.

The MIB defined in this memo supports both models. The ‘push model’ is realized by a table which allows a manager to write scripts by sending a sequence of SNMP set requests. The script can be split into several fragments in order to deal with SNMP message size limitations.

The ‘pull model’ is realized by the use of Uniform Resource Locators (URLs) [17] that point to the script source. The manager writes the URL which points to the script source to the distributed manager by sending an SNMP set request. The distributed manager is then responsible for retrieving the document using the protocol specified in the URL. This allows the use of protocols like FTP [18] or HTTP [19] to transfer large management scripts efficiently.

The Script MIB also allows management scripts that are hard-wired into the Script MIB implementation. Built-in scripts can either be implemented in a language runtime system, or they can be built natively into the Script MIB implementation. The implementation of the ‘push model’ or the ‘pull model’ is not required.

Scripts can be stored in non-volatile storage. This allows a distributed manager to restart scripts if it is restarted (off-line restart). A manager is not required to push scripts back into the distributed manager after a restart if the script is backed up in non-volatile storage.

Every script is identified by an administratively assigned name. This name may be used to derive the name which is used to access the script in non-volatile storage. This mapping is implementation specific. However, the mapping must ensure that the Script MIB implementation can handle scripts with the same administrative name owned by different managers. One way to achieve this is to use the script owner in addition to the script name in order to derive the internal name used to refer to a particular script in non-volatile storage.

4.3. Script Execution

The Script MIB permits execution of several instances of the same or different management scripts. Script arguments are passed as OCTET STRING values. Scripts return a single result value which is also an OCTET STRING value. The semantic interpretation of result values is left to the invoking manager or other management scripts. A script invoker must understand the format and semantics of both the arguments and the results of the scripts that it invokes.

Scripts can also export complex results through a MIB interface. This allows a management application to access and use script results in the same manner as it processes any other MIB data. However, the Script MIB does not provide any special support for the implementation of MIBs through scripts.

Runtime errors terminate active scripts. An exit code and a human readable error message is left in the MIB. A notification containing the exit code, the error message and a timestamp is generated when a script terminates with an error exit code.

Script arguments and results do not have any size limitations other than the limits imposed by the SMI and the SNMP protocol. However, implementations of this MIB might have further restrictions. A script designer might therefore choose to return the results via other mechanisms if the script results can be very large. One possibility is to return a URL as a script result which points to the file containing the script output.

Executing scripts have a status object attached which allows script execution to be suspended, resumed, or aborted. The precise semantics of the suspend and resume operations are language and runtime system dependent. Some runtime systems may choose to not implement the suspend/resume operations.

A history of finished scripts is kept in the MIB. A script invoker can collect results at a later point in time (offline operation). Control objects can be used to control how entries in the history are aged out if the table fills up.

5. The Structure of the MIB

This section presents the structure of the MIB. The objects are arranged into the following groups:

- language group (smLanguageGroup)
- script group (smScriptGroup)
- script code group (smCodeGroup)
- script launch group (smLaunchGroup)
- running script group (smRunGroup)

5.1. The smLanguageGroup

The smLanguageGroup is used to provide information about the languages and the language extensions supported by a Script MIB implementation. This group includes two tables. The smLangTable lists all languages supported by a Script MIB implementation and the smExtsnTable lists the extensions that are available for a given language.

5.2. The smScriptGroup

The smScriptGroup consists of a single table, called the smScriptTable. The smScriptTable lists all scripts known to a Script MIB implementation. The smScriptTable contains objects that allow the following operations:

- download scripts from a URL (pull model)
- read scripts from local non-volatile storage
- store scripts in local non-volatile storage
- delete scripts from local non-volatile storage
- list permanent scripts (that can not be changed or removed)
- read and modify the script status (enabled, disabled, editing)

A status object called smScriptOperStatus allows a manager to obtain the current status of a script. It is also used to provide an error indication if an attempt to invoke one of the operations listed above fails. The status change of a script can be requested by modifying the associated smScriptAdminStatus object.

The source of a script is defined by the smScriptSource object. This object may contain a URL pointing to a remote location which provides access to the management script. The script source is read from the smCodeTable (described below) or from non-volatile storage if the smScriptSource object contains an empty URL. The smScriptStorageType object is used to distinguish between scripts read from non-volatile storage and scripts read from the smCodeTable.

Scripts are automatically loaded once the smScriptAdminStatus object is set to ‘enabled’. Loading a script includes retrieving the script (probably from a remote location), compiling the script for languages that require a compilation step, and making the code available to the runtime system. The smScriptOperStatus object is used to indicate the status of the loading process. This object will start in the

state ‘retrieving’, switch to the state ‘compiling’ and finally reach the state ‘enabled’. Errors during the retrieval or compilation phase will result in an error state such as ‘compilationFailed’.

5.3. The smCodeGroup

The smCodeGroup consists of a single table, called the smCodeTable, which provides the ability to transfer and modify scripts via SNMP set requests. In particular, the smCodeTable allows the following operations:

- download scripts via SNMP (push model)
- modify scripts via SNMP (editing)

The smCodeTable lists the code of a script. A script can be fragmented over multiple rows of the smCodeTable in order to handle SNMP message size limitations. Modifications of the smCodeTable are only possible if the associated smScriptOperStatus object has the value ‘editing’. The Script MIB implementation reloads the modified script code once the smScriptOperStatus changes to ‘enabled’ again.

The implementation of the smCodeGroup is optional.

5.4. The smLaunchGroup

The smLaunchGroup contains a single table, the smLaunchTable. An entry in the smLaunchTable represents a launch button which can be used to start a script. The smLaunchTable allows the following operations:

- associate a script with an owner used during script execution
- provide arguments and parameters for script invocation
- invoke scripts with a single set operation

The smLaunchTable describes scripts and their parameters that are ready to be launched. An entry in the smLaunchTable attaches an argument to a script and control values which, for example, define the maximum number of times that a script invoked from a particular row in the smLaunchTable may be running concurrently.

An entry in the smLaunchTable also defines the owner which will be used to associate permissions with the script execution.

5.5. The smRunGroup

The smRunGroup contains a single table, called the smRunTable, which lists all scripts that are currently running or have terminated recently. The smRunTable contains objects that allow the following operations:

- retrieve status information from running scripts
- control running scripts (suspend, resume, abort)
- retrieve results from recently terminated scripts
- control the remaining maximum lifetime of a running script
- control how long script results are accessible

Every row in the smRunTable contains the argument passed during script invocation, the result produced by the script and the script exit code. The smRunTable also provides information about the current run state as well as start and end time-stamps. There are three writable objects in the smRunTable. The smRunLifeTime object defines the maximum time a running script may run before it is terminated by the Script MIB implementation. The smRunExpireTime object defines the time that a completed script can stay in the smRunTable before it is aged out. The smRunControl object allows running scripts to be suspended, resumed, or aborted.

6. Definitions

```
DISMAN-SCRIPT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
    Integer32, Unsigned32, mib-2
        FROM SNMPv2-SMI

    RowStatus, TimeInterval, DateAndTime, StorageType, DisplayString
        FROM SNMPv2-TC

    MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
        FROM SNMPv2-CONF

    SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB;

scriptMIB MODULE-IDENTITY
    LAST-UPDATED "9902221800Z"
```

```
        ORGANIZATION "IETF Distributed Management Working Group"
        CONTACT-INFO
            "David B. Levi
             Nortel Networks
             4401 Great America Parkway
             Santa Clara, CA 95052-8185
             U.S.A.
             Tel: +1 423 686 0432
             E-mail: dlevi@nortelnetworks.com

             Juergen Schoenwaelder
             TU Braunschweig
             Bueltenweg 74/75
             38106 Braunschweig
             Germany
             Tel: +49 531 391-3283
             E-mail: schoenw@ibr.cs.tu-bs.de"
        DESCRIPTION
            "This MIB module defines a set of objects that allow to
             delegate management scripts to distributed managers."
        ::= { mib-2 64 }

    --
    -- The groups defined within this MIB module:
    --

    smObjects       OBJECT IDENTIFIER ::= { scriptMIB 1 }
    smNotifications OBJECT IDENTIFIER ::= { scriptMIB 2 }
    smConformance   OBJECT IDENTIFIER ::= { scriptMIB 3 }

    --
    -- Script language and language extensions.
    --
    -- This group defines tables which list the languages and the
    -- language extensions supported by a script MIB implementation.
    -- Languages are uniquely identified by object identifier values.
    --

    smLangTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SmLangEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table lists supported script languages."
        ::= { smObjects 1 }

    smLangEntry OBJECT-TYPE
        SYNTAX      SmLangEntry
```

```
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "An entry describing a particular language."
        INDEX { smLangIndex }
        ::= { smLangTable 1 }

    SmLangEntry ::= SEQUENCE {
        smLangIndex         Integer32,
        smLangLanguage      OBJECT IDENTIFIER,
        smLangVersion       SnmpAdminString,
        smLangVendor        OBJECT IDENTIFIER,
        smLangRevision      SnmpAdminString,
        smLangDescr         SnmpAdminString
    }

    smLangIndex OBJECT-TYPE
        SYNTAX      Integer32 (1..2147483647)
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The locally arbitrary, but unique identifier associated
             with this language entry.

             The value is expected to remain constant at least from one
             re-initialization of the entity's network management system
             to the next re-initialization.

             Note, the data type and the range of this object must be
             consistent with the definition of smScriptLanguage."
        ::= { smLangEntry 1 }

    smLangLanguage OBJECT-TYPE
        SYNTAX      OBJECT IDENTIFIER
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The globally unique identification of the language."
        ::= { smLangEntry 2 }

    smLangVersion OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (0..32))
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The version number of the language. The zero-length string
             shall be used if the language does not have a version
             number.
```

```
            It is suggested that the version number consist of one or
            more decimal numbers separated by dots, where the first
            number is called the major version number."
        ::= { smLangEntry 3 }

    smLangVendor OBJECT-TYPE
        SYNTAX      OBJECT IDENTIFIER
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "An object identifer which identifies the vendor who
             provides the implementation of the language. This object
             identifer SHALL point to the object identifier directly
             below the enterprise object identifier {1 3 6 1 4 1}
             allocated for the vendor. The value must be the object
             identifier {0 0} if the vendor is not known."
        ::= { smLangEntry 4 }

    smLangRevision OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (0..32))
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The version number of the language implementation.
             The value of this object must be an empty string if
             version number of the implementation is unknown.

             It is suggested that the value consist of one or more
             decimal numbers separated by dots, where the first
             number is called the major version number."
        ::= { smLangEntry 5 }

    smLangDescr OBJECT-TYPE
        SYNTAX      SnmpAdminString
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "A textual description of the language."
        ::= { smLangEntry 6 }


    smExtsnTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SmExtsnEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table lists supported language extensions."
        ::= { smObjects 2 }
```

```
   smExtsnEntry OBJECT-TYPE
       SYNTAX      SmExtsnEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "An entry describing a particular language extension."
       INDEX { smLangIndex, smExtsnIndex }
       ::= { smExtsnTable 1 }

   SmExtsnEntry ::= SEQUENCE {
       smExtsnIndex        Integer32,
       smExtsnExtension    OBJECT IDENTIFIER,
       smExtsnVersion      SnmpAdminString,
       smExtsnVendor       OBJECT IDENTIFIER,
       smExtsnRevision     SnmpAdminString,
       smExtsnDescr        SnmpAdminString
   }

   smExtsnIndex OBJECT-TYPE
       SYNTAX      Integer32 (1..2147483647)
       MAX-ACCESS  not-accessible
       STATUS      current

       DESCRIPTION
           "The locally arbitrary, but unique identifier associated
            with this language extension entry.

            The value is expected to remain constant at least from one
            re-initialization of the entity's network management system
            to the next re-initialization."
       ::= { smExtsnEntry 1}

   smExtsnExtension OBJECT-TYPE
       SYNTAX      OBJECT IDENTIFIER
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The globally unique identification of the language
            extension."
       ::= { smExtsnEntry 2 }

   smExtsnVersion OBJECT-TYPE
       SYNTAX      SnmpAdminString (SIZE (0..32))
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The version number of the language extension.
```

```
            It is suggested that the version number consist of one or
            more decimal numbers separated by dots, where the first
            number is called the major version number."
        ::= { smExtsnEntry 3 }

    smExtsnVendor OBJECT-TYPE
        SYNTAX      OBJECT IDENTIFIER
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "An object identifer which identifies the vendor who
             provides the implementation of the extension. The
             object identifer value should point to the OID node
             directly below the enterprise OID {1 3 6 1 4 1}
             allocated for the vendor. The value must by the object
             identifier {0 0} if the vendor is not known."
        ::= { smExtsnEntry 4 }

    smExtsnRevision OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (0..32))
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The version number of the extension implementation.
             The value of this object must be an empty string if
             version number of the implementation is unknown.

             It is suggested that the value consist of one or more
             decimal numbers separated by dots, where the first
             number is called the major version number."
        ::= { smExtsnEntry 5 }

    smExtsnDescr OBJECT-TYPE
        SYNTAX      SnmpAdminString
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "A textual description of the language extension."
        ::= { smExtsnEntry 6 }


    --
    -- Scripts known by the Script MIB implementation.
    --
    -- This group defines a table which lists all known scripts.
    -- Scripts can be added and removed through manipulation of the
    -- smScriptTable.
    --
```

```
   smScriptObjects OBJECT IDENTIFIER ::= { smObjects 3 }

   smScriptTable OBJECT-TYPE
       SYNTAX      SEQUENCE OF SmScriptEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "This table lists and describes locally known scripts."
       ::= { smScriptObjects 1 }

   smScriptEntry OBJECT-TYPE
       SYNTAX      SmScriptEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "An entry describing a particular script. Every script that
            is stored in non-volatile memory is required to appear in
            this script table."
       INDEX { smScriptOwner, smScriptName }
       ::= { smScriptTable 1 }
   SmScriptEntry ::= SEQUENCE {
       smScriptOwner       SnmpAdminString,
       smScriptName        SnmpAdminString,
       smScriptDescr       SnmpAdminString,
       smScriptLanguage    Integer32,
       smScriptSource      DisplayString,
       smScriptAdminStatus INTEGER,
       smScriptOperStatus  INTEGER,
       smScriptStorageType StorageType,
       smScriptRowStatus   RowStatus
   }

   smScriptOwner OBJECT-TYPE
       SYNTAX      SnmpAdminString (SIZE (0..32))
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The manager who owns this row in the smScriptTable."
       ::= { smScriptEntry 1 }

   smScriptName OBJECT-TYPE
       SYNTAX      SnmpAdminString
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The locally-unique, administratively assigned name for this
            script. This object allows an smScriptOwner to have multiple
            entries in the smScriptTable.
```

```
            This value of this object may be used to derive the name
            (e.g. a file name) which is used by the Script MIB
            implementation to access the script in non-volatile
            storage. The details of this mapping are implementation
            specific. However, the mapping needs to ensure that scripts
            created by different owners with the same script name do not
            map to the same name in non-volatile storage."
        ::= { smScriptEntry 2 }

    smScriptDescr OBJECT-TYPE
        SYNTAX      SnmpAdminString
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "A description of the purpose of the script."
        ::= { smScriptEntry 3 }

    smScriptLanguage OBJECT-TYPE
        SYNTAX      Integer32 (0..2147483647)
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The value of this object type identifies an entry in the
             smLangTable which is used to execute this script.
             The special value 0 may be used by hard-wired scripts
             that can not be modified and that are executed by
             internal functions.

             Note, the data type and the range of this object must be
             consistent with the definition of smLangIndex."
        ::= { smScriptEntry 4 }

    smScriptSource OBJECT-TYPE
        SYNTAX      DisplayString
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object either contains a reference to the script
             source or an empty string. A reference must be given
             in the form of a Uniform Resource Locator (URL) as
             defined in RFC 2396. The allowed character sets and the
             encoding rules defined in RFC 2396 section 2 apply.

             When the smScriptAdminStatus object is set to 'enabled',
             the Script MIB implementation will 'pull' the script
             source from the URL contained in this object if the URL
             is not empty.
```

```
            An empty URL indicates that the script source is loaded
            from local storage. The script is read from the smCodeTable
            if the value of smScriptStorageType is volatile. Otherwise,
            the script is read from non-volatile storage.

            Note: This document does not mandate implementation of any
            specific URL scheme. A attempt to load a script from a
            nonsupported URL scheme will cause the smScriptOperStatus
            to report an 'unknownProtocol' error.

            Set requests to change this object are invalid if the
            value of smScriptOperStatus is 'enabled', 'editing',
            'retrieving' or 'compiling' and will result in an
            inconsistentValue error."
        DEFVAL { ''H }
        ::= { smScriptEntry 5 }
    smScriptAdminStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        enabled(1),
                        disabled(2),
                        editing(3)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The value of this object indicates the desired status of
            the script. See the definition of smScriptOperStatus for
            a description of the values.

            When the smScriptAdminStatus object is set to 'enabled' and
            the smScriptOperStatus is 'disabled' or one of the error
            states, the Script MIB implementation will 'pull' the script
            source from the URL contained in the smScriptSource object
            if the URL is not empty."
        DEFVAL { disabled }
        ::= { smScriptEntry 6 }

    smScriptOperStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        enabled(1),
                        disabled(2),
                        editing(3),
                        retrieving(4),
                        compiling(5),
                        noSuchScript(6),
                        accessDenied(7),
                        wrongLanguage(8),
                        wrongVersion(9),
```

```
                    compilationFailed(10),
                    noResourcesLeft(11),
                    unknownProtocol(12),
                    protocolFailure(13),
                    genericError(14)
                }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The actual status of the script in the runtime system. The
         value of this object is only meaningful when the value of the
         smScriptRowStatus object is 'active'.

         The smScriptOperStatus object may have the following values:
         - 'enabled' indicates that the script is available and can
            be started by a launch table entry.

         - 'disabled' indicates that the script can not be used.

         - 'editing' indicates that the script can be modified in the
           smCodeTable.

         - 'retrieving' indicates that the script is currently being
           loaded from non-volatile storage or a remote system.

         - 'compiling' indicates that the script is currently being
           compiled by the runtime system.

         - 'noSuchScript' indicates that the script does not exist
           at the smScriptSource.

         - 'accessDenied' indicates that the script can not be loaded
           from the smScriptSource due to a lack of permissions.

         - 'wrongLanguage' indicates that the script can not be loaded
           from the smScriptSource because of a language mismatch.

         - 'wrongVersion' indicates that the script can not be loaded
           from the smScriptSource because of a language version
           mismatch.

         - 'compilationFailed' indicates that the compilation failed.

         - 'noResourcesLeft' indicates that the runtime system does
           not have enough resources to load the script.

         - 'unknownProtocol' indicates that the script could not be
           loaded from the smScriptSource because the requested
```

protocol is not supported.

- ‘protocolFailure’ indicates that the script could not be loaded from the smScriptSource because of a protocol failure.

- ‘genericError’ indicates that the script could not be loaded due to an error condition not listed above.

The ‘retrieving’ and ‘compiling’ states are transient states which will either lead to one of the error states or the ‘enabled’ state. The ‘disabled’ and ‘editing’ states are administrative states which are only reached by explicit management operations.

All launch table entries that refer to this script table entry shall have an smLaunchOperStatus value of ‘disabled’ when the value of this object is not ‘enabled’."
DEFVAL { disabled }
::= { smScriptEntry 7 }

smScriptStorageType OBJECT-TYPE
SYNTAX StorageType
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"This object defines whether this row and the script controlled by this row are kept in volatile storage and lost upon reboot or if this row is backed up by non-volatile or permanent storage.

The script controlled by this row is written into local non-volatile storage if the following condition becomes true:

(a) the URL contained in the smScriptSource object is empty
and
(b) the smScriptStorageType is ‘nonVolatile’
and
(c) the smScriptOperStatus is ‘enabled’

Setting this object to ‘volatile’ removes a script from non-volatile storage if the script controlled by this row has been in non-volatile storage before. Attempts to set this object to permanent will always fail with an inconsistentValue error.

The value of smScriptStorageType is only meaningful if the

```
            value of the corresponding RowStatus object is 'active'.

            If smScriptStorageType has the value permanent(4), then all
            objects whose MAX-ACCESS value is read-create must be
            writable, with the exception of the smScriptStorageType and
            smScriptRowStatus objects, which shall be read-only."
        DEFVAL { volatile }
        ::= { smScriptEntry 8 }

    smScriptRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "A control that allows entries to be added and removed from
            this table.

            Changing the smScriptRowStatus from 'active' to 'notInService'
            will remove the associated script from the runtime system.
            The value of smScriptOperStatus will be reset to 'disabled'.

            Deleting conceptual rows from this table includes the
            deletion of all resources associated with this row. This
            implies that a script stored in non-volatile storage is
            removed from non-volatile storage.

            An entry may not exist in the 'active' state unless all
            required objects in the entry have appropriate values. Rows
            that are not complete or not in service are not known by the
            script runtime system.

            Attempts to 'destroy' a row or to set a row 'notInService'
            while the script is executing will result in an
            inconsistentValue error.

            Attempts to 'destroy' a row or to set a row 'notInService'
            where the value of the smScriptStorageType object is
            'permanent' or 'readOnly' will result in an
            inconsistentValue error."
        ::= { smScriptEntry 9 }


    --
    -- Access to script code via SNMP
    --
    -- The smCodeTable allows script code to be read and modified
    -- via SNMP.
    --
```

```
   smCodeTable OBJECT-TYPE
       SYNTAX      SEQUENCE OF SmCodeEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "This table contains the script code for scripts that are
            written via SNMP write operations."
       ::= { smScriptObjects 2 }

   smCodeEntry OBJECT-TYPE
       SYNTAX      SmCodeEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "An entry describing a particular fragment of a script."
       INDEX { smScriptOwner, smScriptName, smCodeIndex }
       ::= { smCodeTable 1 }

   SmCodeEntry ::= SEQUENCE {
       smCodeIndex         Unsigned32,
       smCodeText          OCTET STRING,
       smCodeRowStatus     RowStatus
   }

   smCodeIndex OBJECT-TYPE
       SYNTAX      Unsigned32 (1..4294967295)
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The index value identifying this code fragment."
       ::= { smCodeEntry 1 }

   smCodeText OBJECT-TYPE
       SYNTAX      OCTET STRING (SIZE (1..1024))
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The code that makes up a fragment of a script. The format
            of this code fragment depends on the script language which
            is identified by the associated smScriptLanguage object."
       ::= { smCodeEntry 2 }

   smCodeRowStatus OBJECT-TYPE
       SYNTAX      RowStatus
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "A control that allows entries to be added and removed from
```

```
            this table."
        ::= { smCodeEntry 3 }

    --
    -- Script execution.
    --
    -- This group defines tables which allow script execution to be
    -- initiated, suspended, resumed, and terminated.  It also provides
    -- a mechanism for keeping a history of recent script executions
    -- and their results.
    --

    smRunObjects OBJECT IDENTIFIER ::= { smObjects 4 }

    smLaunchTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SmLaunchEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table lists and describes scripts that are ready
             to be executed together with their parameters."
        ::= { smRunObjects 1 }

    smLaunchEntry OBJECT-TYPE
        SYNTAX      SmLaunchEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "An entry describing a particular executable script."
        INDEX { smLaunchOwner, smLaunchName }
        ::= { smLaunchTable 1 }

    SmLaunchEntry ::= SEQUENCE {
        smLaunchOwner               SnmpAdminString,
        smLaunchName                SnmpAdminString,
        smLaunchScriptOwner         SnmpAdminString,
        smLaunchScriptName          SnmpAdminString,
        smLaunchArgument            OCTET STRING,
        smLaunchMaxRunning          Unsigned32,
        smLaunchMaxCompleted        Unsigned32,
        smLaunchLifeTime            TimeInterval,
        smLaunchExpireTime          TimeInterval,
        smLaunchStart               Integer32,
        smLaunchControl             INTEGER,
        smLaunchAdminStatus         INTEGER,
        smLaunchOperStatus          INTEGER,
        smLaunchRunIndexNext        Integer32,
        smLaunchStorageType         StorageType,
```

```
        smLaunchRowStatus           RowStatus
    }

    smLaunchOwner OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (0..32))
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION

            "The manager who owns this row in the smLaunchTable. Every
             instance of a running script started from a particular entry
             in the smLaunchTable (i.e. entries in the smRunTable) will be
             owned by the same smLaunchOwner used to index the entry in
             the smLaunchTable. This owner is not necessarily the same as
             the owner of the script itself (smLaunchScriptOwner)."
        ::= { smLaunchEntry 1 }

    smLaunchName OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (1..32))
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The locally-unique, administratively assigned name for this
             launch table entry. This object allows an smLaunchOwner to
             have multiple entries in the smLaunchTable. The smLaunchName
             is an arbitrary name that must be different from any other
             smLaunchTable entries with the same smLaunchOwner but can be
             the same as other entries in the smLaunchTable with different
             smLaunchOwner values. Note that the value of smLaunchName
             is not related in any way to the name of the script being
             launched."
        ::= { smLaunchEntry 2 }

    smLaunchScriptOwner OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (0..32))
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The value of this object in combination with the value of
             smLaunchScriptName identifies the script that can be
             launched from this smLaunchTable entry. Attempts to write
             this object will fail with an inconsistentValue error if
             the value of smLaunchOperStatus is 'enabled'."
        ::= { smLaunchEntry 3 }

    smLaunchScriptName OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE (0..32))
        MAX-ACCESS  read-create
```

```
        STATUS      current
        DESCRIPTION
            "The value of this object in combination with the value of
             the smLaunchScriptOwner identifies the script that can be
             launched from this smLaunchTable entry. Attempts to write
             this objects will fail with an inconsistentValue error if
             the value of smLaunchOperStatus is 'enabled'."
        ::= { smLaunchEntry 4 }

    smLaunchArgument OBJECT-TYPE
        SYNTAX      OCTET STRING
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The argument supplied to the script. When a script is
             invoked, the value of this object is used to initialize
             the smRunArgument object."
        DEFVAL { ''H }
        ::= { smLaunchEntry 5 }

    smLaunchMaxRunning OBJECT-TYPE
        SYNTAX      Unsigned32 (1..4294967295)
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The maximum number of concurrently running scripts that may
             be invoked from this entry in the smLaunchTable. Lowering the
             current value of this object does not affect any scripts that
             are already executing."
        DEFVAL { 1 }
        ::= { smLaunchEntry 6 }

    smLaunchMaxCompleted OBJECT-TYPE
        SYNTAX      Unsigned32 (1..4294967295)
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The maximum number of finished scripts invoked from this
             entry in the smLaunchTable allowed to be retained in the
             smRunTable. Whenever the value of this object is changed
             and whenever a script terminates, entries in the smRunTable
             are deleted if necessary until the number of completed
             scripts is smaller than the value of this object. Scripts
             whose smRunEndTime value indicates the oldest completion
             time are deleted first."
        DEFVAL { 1 }
        ::= { smLaunchEntry 7 }
```

```
   smLaunchLifeTime OBJECT-TYPE
       SYNTAX      TimeInterval
       UNITS       "centi-seconds"
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The default maximum amount of time a script launched
            from this entry may run. The value of this object is used
            to initialize the smRunLifeTime object when a script is
            launched. Changing the value of an smLaunchLifeTime
            instance does not affect scripts previously launched from
            this entry."
       DEFVAL { 360000 }
       ::= { smLaunchEntry 8 }

   smLaunchExpireTime OBJECT-TYPE
       SYNTAX      TimeInterval
       UNITS       "centi-seconds"
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The default maximum amount of time information about a
            script launched from this entry is kept in the smRunTable
            after the script has completed execution.  The value of
            this object is used to initialize the smRunExpireTime
            object when a script is launched. Changing the value of an
            smLaunchExpireTime instance does not affect scripts
            previously launched from this entry."
       DEFVAL { 360000 }
       ::= { smLaunchEntry 9 }

   smLaunchStart OBJECT-TYPE
       SYNTAX      Integer32 (0..2147483647)
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "This object is used to start the execution of scripts.
            When retrieved, the value will be the value of smRunIndex
            for the last script that started execution by manipulating
            this object. The value will be zero if no script started
            execution yet.

            A script is started by setting this object to an unused
            smRunIndex value. A new row in the smRunTable will be
            created which is indexed by the value supplied by the
            set-request in addition to the value of smLaunchOwner and
            smLaunchName. An unused value can be obtained by reading
            the smLaunchRunIndexNext object.
```

Setting this object to the special value 0 will start the script with a self-generated smRunIndex value. The consequence is that the script invoker has no reliable way to determine the smRunIndex value for this script invocation and that the invoker has therefore no way to obtain the results from this script invocation. The special value 0 is however useful for scheduled script invocations.

If this object is set, the following checks must be performed:

1) The value of the smLaunchOperStatus object in this entry of the smLaunchTable must be 'enabled'.
2) The values of smLaunchScriptOwner and smLaunchScriptName of this row must identify an existing entry in the smScriptTable.
3) The value of smScriptOperStatus of this entry must be 'enabled'.
4) The principal performing the set operation must have read access to the script. This must be checked by calling the isAccessAllowed abstract service interface defined in RFC 2271 on the row in the smScriptTable identified by smLaunchScriptOwner and smLaunchScriptName. The isAccessAllowed abstract service interface must be called on all columnar objects in the smScriptTable with a MAX-ACCESS value different than 'not-accessible'. The test fails as soon as a call indicates that access is not allowed.
5) If the value provided by the set operation is not 0, a check must be made that the value is currently not in use. Otherwise, if the value provided by the set operation is 0, a suitable unused value must be generated.
6) The number of currently executing scripts invoked from this smLaunchTable entry must be less than smLaunchMaxRunning.

Attempts to start a script will fail with an inconsistentValue error if one of the checks described above fails.

Otherwise, if all checks have been passed, a new entry in the smRunTable will be created indexed by smLaunchOwner, smLaunchName and the new value for smRunIndex. The value of smLaunchArgument will be copied into smRunArgument, the value of smLaunchLifeTime will be copied to smRunLifeTime, and the value of smLaunchExpireTime

```
            will be copied to smRunExpireTime.

            The smRunStartTime will be set to the current time and
            the smRunState will be set to 'initializing' before the
            script execution is initiated in the appropriate runtime
            system.

            Note, the data type and the range of this object must
            be consistent with the smRunIndex object. Since this
            object might be written from the scheduling MIB, the
            data type Integer32 rather than Unsigned32 is used."
        DEFVAL { 0 }
        ::= { smLaunchEntry 10 }

    smLaunchControl OBJECT-TYPE
        SYNTAX      INTEGER {
                        abort(1),
                        suspend(2),
                        resume(3),
                        nop(4)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object is used to request a state change for all
             running scripts in the smRunTable that were started from
             this row in the smLaunchTable.

             Setting this object to abort(1), suspend(2) or resume(3)
             will set the smRunControl object of all applicable rows
             in the smRunTable to abort(1), suspend(2) or resume(3)
             respectively. The phrase 'applicable rows' means the set of
             rows which were created from this entry in the smLaunchTable
             and whose value of smRunState allows the corresponding
             state change as described in the definition of the
             smRunControl object. Setting this object to nop(4) has no
             effect."
        DEFVAL { nop }
        ::= { smLaunchEntry 11 }

    smLaunchAdminStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        enabled(1),
                        disabled(2)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
```

```
            "The value of this object indicates the desired status of
             this launch table entry."
        DEFVAL { disabled }
        ::= { smLaunchEntry 12 }

    smLaunchOperStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        enabled(1),
                        disabled(2)
                    }
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The value of this object indicates the actual status of
             this launch table entry. An 'enabled' launch table
             entry can be used to start scripts while a 'disabled'
             launch table entry will refuse any attempts to start
             scripts. The value 'enabled' requires that the
             smLaunchRowStatus object is active. The value
             'disabled' requires that there are no entries in the
             smRunTable associated with this smLaunchTable entry."
        DEFVAL { disabled }
        ::= { smLaunchEntry 13 }

    smLaunchRunIndexNext OBJECT-TYPE
        SYNTAX      Integer32 (1..2147483647)
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "This variable is used for creating rows in the smRunTable.
             The value of this variable is a currently unused value
             for smRunIndex, which can be written into the smLaunchStart
             object associated with this row to launch a script.

             The value returned when reading this variable must be unique
             for the smLaunchOwner and smLauchName associated with this
             row. Subsequent attempts to read this variable must return
             different values.

             This variable will return the special value 0 if no new rows
             can be created.

             Note, the data type and the range of this object must be
             consistent with the definition of smRunIndex."
        ::= { smLaunchEntry 14 }

    smLaunchStorageType OBJECT-TYPE
        SYNTAX      StorageType
```

```
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object defines if this row is kept in volatile storage
             and lost upon reboot or if this row is backed up by stable
             storage.

             The value of smLaunchStorageType is only meaningful if the
             value of the corresponding RowStatus object is active.

             If smLaunchStorageType has the value permanent(4), then all
             objects whose MAX-ACCESS value is read-create must be
             writable, with the exception of the smLaunchStorageType and
             smLaunchRowStatus objects, which shall be read-only."
        DEFVAL { volatile }
        ::= { smLaunchEntry 15 }

    smLaunchRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "A control that allows entries to be added and removed from
             this table.

             Attempts to 'destroy' a row or to set a row 'notInService'
             while scripts started from this launch table entry are
             running will result in an inconsistentValue error.

             Attempts to 'destroy' a row or to set a row 'notInService'
             where the value of the smLaunchStorageType object is
             'permanent' or 'readOnly' will result in an
             inconsistentValue error."
        ::= { smLaunchEntry 16 }


    smRunTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SmRunEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table lists and describes scripts that are currently
             running or have been running in the past."
        ::= { smRunObjects 2 }

    smRunEntry OBJECT-TYPE
        SYNTAX      SmRunEntry
        MAX-ACCESS  not-accessible
```

```
        STATUS      current
        DESCRIPTION
            "An entry describing a particular running or finished
             script."
        INDEX { smLaunchOwner, smLaunchName, smRunIndex }
        ::= { smRunTable 1 }

    SmRunEntry ::= SEQUENCE {
        smRunIndex          Integer32,
        smRunArgument       OCTET STRING,
        smRunStartTime      DateAndTime,
        smRunEndTime        DateAndTime,
        smRunLifeTime       TimeInterval,
        smRunExpireTime     TimeInterval,
        smRunExitCode       INTEGER,
        smRunResult         OCTET STRING,
        smRunControl        INTEGER,
        smRunState          INTEGER,
        smRunError          SnmpAdminString
    }

    smRunIndex OBJECT-TYPE
        SYNTAX      Integer32 (1..2147483647)
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The locally arbitrary, but unique identifier associated
             with this running or finished script. This value must be
             unique for all rows in the smRunTable with the same
             smLaunchOwner and smLaunchName.

             Note, the data type and the range of this object must be
             consistent with the definition of smLaunchRunIndexNext
             and smLaunchStart."
        ::= { smRunEntry 1 }

    smRunArgument OBJECT-TYPE
        SYNTAX      OCTET STRING
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The argument supplied to the script when it started."
        DEFVAL { ''H }
        ::= { smRunEntry 2 }

    smRunStartTime OBJECT-TYPE
        SYNTAX      DateAndTime
        MAX-ACCESS  read-only
```

```
        STATUS      current
        DESCRIPTION
            "The date and time when the execution started. The value
             '0000000000000000'H is returned if the script has not
             started yet."
        DEFVAL { '0000000000000000'H }
        ::= { smRunEntry 3 }

    smRunEndTime OBJECT-TYPE
        SYNTAX      DateAndTime
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The date and time when the execution terminated. The value
             '0000000000000000'H is returned if the script has not
             terminated yet."
        DEFVAL { '0000000000000000'H }
        ::= { smRunEntry 4 }

    smRunLifeTime OBJECT-TYPE
        SYNTAX      TimeInterval
        UNITS       "centi-seconds"
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
            "This object specifies how long the script can execute.
             This object returns the remaining time that the script
             may run. The object is initialized with the value of the
             associated smLaunchLifeTime object and ticks backwards.
             The script is aborted immediately when the value reaches 0.

             The value of this object may be set in order to increase or
             reduce the remaining time that the script may run. Setting
             this value to 0 will abort script execution immediately,
             and, if the value of smRunExpireTime is also 0, will remove
             this entry from the smRunTable once it has terminated.

             The value of smRunLifeTime reflects the real-time execution
             time as seen by the outside world. The value of this object
             will always be 0 for a script that finished execution, that
             is smRunState has the value 'terminated'.

             The value of smRunLifeTime does not change while a script
             is suspended, that is smRunState has the value 'suspended'.
             Note, this does not affect set operations. It is legal to
             modify smRunLifeTime via set operations while a script is
             suspended."
        ::= { smRunEntry 5 }
```

```
   smRunExpireTime OBJECT-TYPE
       SYNTAX      TimeInterval
       UNITS       "centi-seconds"
       MAX-ACCESS  read-write
       STATUS      current
       DESCRIPTION
           "This value specifies how long this row can exist in the
            smRunTable after the script has terminated.  This object
            returns the remaining time that the row may exist before it
            is aged out. The object is initialized with the value of the
            associated smLaunchExpireTime object and ticks backwards. The
            entry in the smRunTable is destroyed when the value reaches 0
            and the smRunState has the value 'terminated'.

            The value of this object may be set in order to increase or
            reduce the remaining time that the row may exist.  Setting
            the value to 0 will destroy this entry as soon as the
            smRunState has the value 'terminated'."
       ::= { smRunEntry 6 }

   smRunExitCode OBJECT-TYPE
       SYNTAX      INTEGER {
                       noError(1),
                       halted(2),
                       lifeTimeExceeded(3),
                       noResourcesLeft(4),
                       languageError(5),
                       runtimeError(6),
                       invalidArgument(7),
                       securityViolation(8),
                       genericError(9)
                   }
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "The value of this object indicates the reason why a
            script finished execution. The smRunExitCode code may have
            one of the following values:

            - 'noError', which indicates that the script completed
              successfully without errors;

            - 'halted', which indicates that the script was halted
              by a request from an authorized manager;

            - 'lifeTimeExceeded', which indicates that the script
              exited because a time limit was exceeded;
```

```
            - 'noResourcesLeft', which indicates that the script
              exited because it ran out of resources (e.g. memory);

            - 'languageError', which indicates that the script exited
              because of a language error (e.g. a syntax error in an
              interpreted language);

            - 'runtimeError', which indicates that the script exited
              due to a runtime error (e.g. a division by zero);

            - 'invalidArgument', which indicates that the script could
              not be run because of invalid script arguments;

            - 'securityViolation', which indicates that the script
              exited due to a security violation;

            - 'genericError', which indicates that the script exited
              for an unspecified reason.

            If the script has not yet begun running, or is currently
            running, the value will be 'noError'."
        DEFVAL { noError }
        ::= { smRunEntry 7 }

    smRunResult OBJECT-TYPE
        SYNTAX      OCTET STRING
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The result value produced by the running script. Note that
             the result may change while the script is executing."
        DEFVAL { ''H }
        ::= { smRunEntry 8 }

    smRunControl OBJECT-TYPE
        SYNTAX      INTEGER {
                        abort(1),
                        suspend(2),
                        resume(3),
                        nop(4)
                    }
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
            "The value of this object indicates the desired status of the
             script execution defined by this row.

             Setting this object to 'abort' will abort execution if the
```

```
            value of smRunState is 'initializing', 'executing',
            'suspending', 'suspended' or 'resuming'. Setting this object
            to 'abort' when the value of smRunState is 'aborting' or
            'terminated' will result in an inconsistentValue error.

            Setting this object to 'suspend' will suspend execution
            if the value of smRunState is 'executing'. Setting this
            object to 'suspend' will cause an inconsistentValue error
            if the value of smRunState is not 'executing'.

            Setting this object to 'resume' will resume execution
            if the value of smRunState is 'suspending' or
            'suspended'. Setting this object to 'resume' will cause an
            inconsistentValue error if the value of smRunState is
            not 'suspending' or 'suspended'.

            Setting this object to nop(4) has no effect."
        DEFVAL { nop }
        ::= { smRunEntry 9 }

    smRunState OBJECT-TYPE
        SYNTAX      INTEGER {
                        initializing(1),
                        executing(2),
                        suspending(3),
                        suspended(4),
                        resuming(5),
                        aborting(6),
                        terminated(7)
                    }
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
            "The value of this object indicates the script's execution
             status.  If the script has been invoked but has not yet
             begun execution, the value will be 'initializing'. If the
             script is running, the value will be 'executing'. A script
             which received a request to suspend execution but which
             did not actually suspend execution will be 'suspending'.
             A script which has suspended execution will be 'suspended'.
             A script which received a request to resume execution but
             which is not yet running is 'resuming'. The resuming state
             will finally lead to the 'executing' state. A script which
             received a request to abort execution but which is still
             running is 'aborting'. A script which stopped execution
             is 'terminated'."
        ::= { smRunEntry 10 }
```

```
   smRunError OBJECT-TYPE
       SYNTAX      SnmpAdminString
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
           "This  contains a descriptive error message if the script
            terminates in an abnormally. An implementation must store a
            descriptive error message in this object if the script exits
            with the smRunExitCode 'genericError'.

            The value of this object is the zero-length string as long
            as the smRunExitCode has the value 'noError'"
       DEFVAL { ''H }
       ::= { smRunEntry 11 }

   --
   -- Notifications. The definition of smTraps makes notification
   -- registrations reversible (see STD 58, RFC 2578).
   --

   smTraps OBJECT IDENTIFIER ::= { smNotifications 0 }

   smScriptAbort NOTIFICATION-TYPE
       OBJECTS     { smRunExitCode, smRunEndTime, smRunError }
       STATUS      current
       DESCRIPTION
           "This notification is generated whenever a running script
            terminates with an smRunExitCode unequal to 'noError'."
       ::= { smTraps 1 }

   smScriptResult NOTIFICATION-TYPE
       OBJECTS     { smRunResult }
       STATUS      current
       DESCRIPTION
           "This notification can be used by scripts to notify other
            management applications about script results. It can be
            used to notify managers about a script result.

            This notification is not automatically generated by the
            script MIB implementation. It is the responsibility of
            the executing script to emit this notification where it
            is appropriate to do so."
       ::= { smTraps 2 }

   -- conformance information
   smCompliances OBJECT IDENTIFIER ::= { smConformance 1 }
   smGroups      OBJECT IDENTIFIER ::= { smConformance 2 }
```

```
   -- compliance statements

   smCompliance MODULE-COMPLIANCE
       STATUS      current
       DESCRIPTION
           "The compliance statement for SNMP entities which implement
            the script MIB."
       MODULE      -- this module
       MANDATORY-GROUPS {
               smLanguageGroup, smScriptGroup, smLaunchGroup, smRunGroup
       }
       GROUP   smCodeGroup
       DESCRIPTION
           "The smCodeGroup is mandatory only for those implementations
            that support the downloading of scripts via SNMP."
       OBJECT  smScriptSource
       MIN-ACCESS  read-only
       DESCRIPTION
           "The smScriptSource object is read-only for implementations
            that are not able to download script code from a URL."
       OBJECT smLaunchArgument
       DESCRIPTION
           "A compliant implementation has to support a minimum size
            for smLaunchArgument of 255 octets."
       OBJECT smRunArgument
       DESCRIPTION
           "A compliant implementation has to support a minimum size
            for smRunArgument of 255 octets."
       OBJECT smRunResult
       DESCRIPTION
           "A compliant implementation has to support a minimum size
            for smRunResult of 255 octets."
       OBJECT smRunState
       DESCRIPTION
           "A compliant implementation does not have to support script
            suspension and the smRunState 'suspended'. Such an
            implementation will change into the 'suspending' state
            when the smRunControl is set to 'suspend' and remain in this
            state until smRunControl is set to 'resume' or the script
            terminates."
       ::= { smCompliances 1 }

   smLanguageGroup OBJECT-GROUP
       OBJECTS {
           smLangLanguage,
           smLangVersion,
           smLangVendor,
           smLangRevision,
```

```
            smLangDescr,
            smExtsnExtension,
            smExtsnVersion,
            smExtsnVendor,
            smExtsnRevision,
            smExtsnDescr
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects providing information about the
             capabilities of the scripting engine."
        ::= { smGroups 1 }

    smScriptGroup OBJECT-GROUP
        OBJECTS {
            smScriptDescr,
            smScriptLanguage,
            smScriptSource,
            smScriptAdminStatus,
            smScriptOperStatus,
            smScriptStorageType,
            smScriptRowStatus
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects providing information about
             installed scripts."
        ::= { smGroups 2 }

    smCodeGroup OBJECT-GROUP
        OBJECTS {
            smCodeText,
            smCodeRowStatus
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects used to download or modify scripts
             by using SNMP set requests."
        ::= { smGroups 3 }

    smLaunchGroup OBJECT-GROUP
        OBJECTS {
            smLaunchScriptOwner,
            smLaunchScriptName,
            smLaunchArgument,
            smLaunchMaxRunning,
            smLaunchMaxCompleted,
            smLaunchLifeTime,
```

```
            smLaunchExpireTime,
            smLaunchStart,
            smLaunchControl,
            smLaunchAdminStatus,
            smLaunchOperStatus,
            smLaunchRunIndexNext,
            smLaunchStorageType,
            smLaunchRowStatus
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects providing information about scripts
             that can be launched."
        ::= { smGroups 4 }

    smRunGroup OBJECT-GROUP
        OBJECTS {
            smRunArgument,
            smRunStartTime,
            smRunEndTime,
            smRunLifeTime,
            smRunExpireTime,
            smRunExitCode,
            smRunResult,
            smRunState,
            smRunControl,
            smRunError
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects providing information about running
             scripts."
        ::= { smGroups 5 }

    smNotificationsGroup NOTIFICATION-GROUP
        NOTIFICATIONS {
            smScriptAbort,
            smScriptResult
        }
        STATUS      current
        DESCRIPTION
            "The notifications emitted by the script MIB."
        ::= { smGroups 6 }

    END
```

7. Usage Examples

This section presents some examples that explain how a manager can use the Script MIB defined in this memo. The purpose of these examples is to explain the steps that are normally used to delegate management scripts.

7.1. Pushing a script via SNMP

This example explains the steps performed by a manager to push a script into a distributed manager.

1. The manager first checks the smLanguageTable and the smExtensionTable in order to select the appropriate script or language.

2. The manager creates a row in the smScriptTable by issuing an SNMP set-request. The smScriptRowStatus object is set to 'createAndWait' and the smScriptSource object is set to an empty string. The smScriptLanguage object is set to the language in which the script was written. The smScriptStorageType object is set to 'volatile' to indicate that the script will be loaded via the smCodeTable. The smScriptOwner is set to a string which identifies the principal who owns the new row. The smScriptName defines the administratively assigned unique name for the script.

3. The manager sets the smScriptRowStatus object to 'active' and the smScriptAdminStatus object to 'editing'.

4. The manager pushes the script to the distributed manager by issuing a couple of SNMP set-requests to fill the smCodeTable.

5. Once the whole script has been transferred, the manager sends a set-request to set the smScriptAdminStatus object to 'enabled'. The Script MIB implementation now makes the script accessible to the runtime system. This might include the compilation of the script if the language requires a compilation step.

6. The manager polls the smScriptOperStatus object until the value is either 'enabled' or one of the error status codes. The script can only be used if the value of smScriptOperStatus is 'enabled'.

7. If the manager wants to store the script in local non-volatile storage, it should send a set-request which changes the smScriptStorageType object to 'nonVolatile'.

7.2. Pulling a script from a URL

This example explains the steps performed by a manager to cause a distributed manager to pull a script from a URL.

1. The manager first checks the smLanguageTable and the smExtensionTable in order to select the appropriate script or language.

2. The manager creates a row in the smScriptTable by issuing an SNMP set-request. The smScriptRowStatus object is set to 'createAndWait' and the smScriptSource object is set to the URL which points to the script source. The smScriptLanguage object is set to the language in which the script was written. The smScriptOwner is set to a string which identifies the principal who owns the new row. The smScriptName defines the administratively assigned unique name for the script.

3. The manager sets the smScriptRowStatus object to 'active'.

4. The manager sends a set-request to set the smScriptAdminStatus object to 'enabled'. The Script MIB implementation now makes the script accessible to the runtime system. This causes a retrieval operation to pull the script from the URL stored in smScriptSource. This retrieval operation might be followed by a compile operation if the language requires a compilation step.

5. The manager polls the smScriptOperStatus object until the value is either 'enabled' or one of the error status codes. The script can only be used if the value of smScriptOperStatus is 'enabled'.

6. If the manager wants to store the script in local non-volatile storage, it should send a set-request which changes the smScriptStorageType object to 'nonVolatile'.

7.3. Modifying an existing script

This section explains how a manager can modify a script by sending SNMP set-requests.

1. First, the script is de-activated by setting the smScriptAdminStatus to 'disabled'.

2. The manager polls the smScriptOperStatus object until the value is 'disabled'.

3. The manager sets smScriptSource to an empty string and smScriptAdminStatus to ‘editing’. This makes the script source available in the smCodeTable.

4. The manager polls the smScriptOperStatus object until the value is ‘editing’.

5. The manager sends SNMP set-requests to modify the script in the smCodeTable.

6. The manager sends a set-request to set the smScriptAdminStatus object to ‘enabled’. The Script MIB implementation now makes the script accessible to the runtime system. This might include the compilation of the script if the language requires a compilation step.

7. The manager polls the smScriptOperStatus object until the value is either ‘enabled’ or one of the error status codes. The script can only be used if the value of smScriptOperStatus is ‘enabled’.

7.4. Removing an existing script

This section explains how a manager can remove a script from a distributed manager.

1. First, the manager sets the smScriptAdminStatus to ‘disabled’. This will ensure that no new scripts can be started while running scripts finish their execution.

2. The manager polls the smScriptOperStatus object until the value is ‘disabled’.

3. The manager sends an SNMP set-request to change the smScriptRowStatus object to ‘destroy’. This will remove the row and all associated resources from the Script MIB implementation.

7.5. Creating a launch button

This section explains how a manager can create a launch button for starting a script.

1. The manager, who is identified by an smLaunchOwner value, first chooses a name for the new row in the smLaunchTable. The manager sends an SNMP set-request to set the smLaunchRowStatus object for this smLaunchOwner and smLaunchName to ‘createAndWait’.

2. The manager fills the new smLaunchTable row with all required parameters. The smLaunchScriptOwner and smLaunchScriptName values point to the script that should be started from this launch button.

3. The manager sends a set-request to change smLaunchAdminStatus to 'enabled' once the new smLaunchTable row is complete.

4. The manager polls the smLaunchOperStatus object until the value is 'enabled'.

7.6. Launching a script

This section explains the suggested way to launch a script from a given launch button.

1. The manager first retrieves the value of smLaunchRunIndexNext from the launch button selected to start the script.

2. The manager sends an SNMP set-request to set the smLaunchStart object to the value obtained in step 1. This will launch the script if all necessary pre-conditions are satisfied (see the definition of smLaunchStart for more details). The manager can also provide the smLaunchArgument in the same set-request that is used to start the script. Upon successful start, a new row will be created in the smRunTable indexed by smLaunchOwner, smLaunchName and the value written to smLaunchStart.

Note, the first step is not required. A manager can also try to guess an unused value for smRunIndex if he wants to start script in a single transaction. A manager can also use the special value 0 if he does not care about the results produced by the script.

7.7. Terminating a script

This section explains two ways to terminate a running script. The first approach is as follows:

1. The manager sets the smRunControl object of the running script or the smLaunchControl object of the launch button used to start the running script to 'abort'. Setting smLaunchControl will abort all running scripts started from the launch button while smRunControl will only abort the running script associated with the smRunControl instance.

The second way to terminate a script is to set the smRunLifeTime to zero which causes the runtime system to terminate the script with a 'lifeTimeExceeded' exit code:

1. The manager changes the value of smRunLifeTime to 0. This causes the Script MIB implementation to abort the script because the remaining life time has expired.

Note, changing the smRunLifeTime value can also be used to increase the permitted lifetime of a running script. For example, a manager can choose to set smRunLifeTime to a small fixed time interval and increase the value periodically. This strategy has the nice effect that scripts terminate automatically if the manager loses contact with the Script MIB engine.

7.8. Removing a launch button

This section explains how a manager can remove a launch button from a distributed manager.

1. First, the manager sets the smLaunchAdminStatus to 'disabled'. This will ensure that no new scripts can be started from this launch button while running script will finish their execution.

2. The manager polls the smLaunchOperStatus object until the value is 'disabled'.

3. The manager sends an SNMP set-request to change the smLaunchRowStatus object to 'destroy'. This will remove the row and all associated resources from the Script MIB implementation.

8. VACM Configuration Examples

This section shows how the view-based access control model defined in RFC 2275 [15] can be configured to control access to the script MIB.

8.1. Sandbox for guests

The first example demonstrates how to configure VACM to give the members of the VACM group "guest" limited access to the script MIB. The MIB views defined below give the members of the "guest" group a sandbox where they can install and start their own scripts, but not access any other scripts maintained by the Script MIB implementation.

```
vacmAccessReadView."guest"."".usm.authNoPriv = "guestReadView"
vacmAccessWriteView."guest"."".usm.authNoPriv = "guestWriteView"
```

The guestReadView grants read access to the smLangTable, the smExtsnTable and to all the table entries owned by "guest":

```
guestReadView:
    smLangTable                      (included)
    smExtsnTable                     (included)
    smScriptObjects.*.*.*."guest"    (included)
    smRunObjects.*.*.*."guest"       (included)
```

The guestWriteView grants write access to all the table entries owned by "guest":

```
guestWriteView:
    smScriptObjects.*.*.*."guest"    (included)
    smRunObjects.*.*.*."guest"       (included)
```

8.2. Sharing scripts

This example demonstrates how VACM can be used to share a repository of scripts between the members of the "senior" and the members of the "junior" VACM group:

```
vacmAccessReadView."junior"."".usm.authNoPriv = "juniorReadView"
vacmAccessWriteView."junior"."".usm.authNoPriv = "juniorWriteView"

juniorReadView:
    smLangTable                      (included)
    smExtsnTable                     (included)
    smScriptObjects.*.*.*."junior"   (included)
    smRunObjects.*.*.*."junior"      (included)
    smScriptObjects.*.*.*."utils"    (included)

juniorWriteView:
    smScriptObjects.*.*.*."junior"   (included)
    smRunObjects.*.*.*."junior"      (included)
```

The definitions above allow the members of the "junior" VACM group to start the scripts owned by "utils" in addition to the script the members of the "junior" VACM group installed themself. This is accomplished by giving the members of "junior" read access to scripts in "utils". This allows members of "junior" to create entries in the smLauchTable which refer to scripts in "utils", and to launch those scripts using these entries in the smLaunchTable.

```
vacmAccessReadView."senior"."".usm.authNoPriv = "seniorReadView"
vacmAccessWriteView."senior"."".usm.authNoPriv = "seniorWriteView"

seniorReadView:
    smLangTable                      (included)
    smExtsnTable                     (included)
    smScriptObjects.*.*.*."senior"   (included)
    smRunObjects.*.*.*."senior"      (included)
    smScriptObjects.*.*.*."utils"    (included)

seniorWriteView:
    smScriptObjects.*.*.*."senior"   (included)
    smRunObjects.*.*.*."senior"      (included)
    smScriptObjects.*.*.*."utils"    (included)
```

The definitions for the members of the "senior" VACM group allow to start the scripts owned by "utils" in addition to the script the members of the "senior" VACM group installed themself. The third write access rule in the seniorWriteView also grants the permission to install scripts owned by "utils". The members of the "senior" VACM group therefore have the permissions to install and modify scripts that can be called by the members of the "junior" VACM group.

8.3. Emergency scripts

This example demonstrates how VACM can be used to allow the members of the "junior" VACM group to launch scripts that are executed with the permissions associated with the "emergency" owner. This works by adding the following rules to the juniorReadView and the juniorWriteView:

```
juniorReadView:
    smScriptObjects.*.*.*."emergency" (included)
    smRunObjects.*.*.*."emergency"    (included)

juniorWriteView
    smLaunchStart."emergency"         (included)
    smLaunchArgument."emergency"      (included)
```

The rules added to the juniorReadView grant read access to the scripts, the launch buttons and the results owned by "emergency". The rules added to the juniorWriteView grant write permissions to the smLaunchStart and smLaunchArgument variables ownded by "emergency". Members of the "junior" VACM group can therefore start scripts that will execute under the owner "emergency".

```
seniorReadView:
    smScriptObjects.*.*.*."emergency" (included)
    smRunObjects.*.*.*."emergency"    (included)

seniorWriteView:
    smScriptObjects.*.*.*."emergency" (included)
    smRunObjects.*.*.*."emergency"    (included)
```

The rules added to the seniorReadView and the seniorWriteView will give the members of the "senior" VACM group the rights to install emergency scripts and to configure appropriate launch buttons.

9. IANA Considerations

The Internet Assigned Numbers Authority (IANA) is responsible for maintaining a MIB module which provides OID registrations for well-known languages. The IANA language registry is intented to reduce interoperability problems by providing a single list of well-known languages. However, it is of course still possible to register languages in private OID spaces. Registering languages in private spaces is especially attractive if a language is used for experimentation or if a language is only used in environments where the distribution of MIB modules with the language registration does not cause any maintenance problems.

Any additions or changes to the list of languages registered via IANA require Designated Expert Review as defined in the IANA guidelines [20]. The Designated Expert will be selected by the IESG Area Director for the IETF Operations and Management Area.

10. Security Considerations

This MIB provides the ability to distribute applications written in an arbitrary language to remote systems in a network. The security features of the languages available in a particular implementation should be taken into consideration when deploying an implementation of this MIB.

To facilitate the provisioning of access control by a security administrator using the View-Based Access Control Model (VACM) defined in RFC 2275 [15] for tables in which multiple users may need to independently create or modify entries, the initial index is used as an "owner index". Such an initial index has a syntax of SnmpAdminString, and can thus be trivially mapped to a securityName or groupName as defined in VACM, in accordance with a security policy.

All entries in related tables belonging to a particular user will have the same value for this initial index. For a given user's entries in a particular table, the object identifiers for the information in these entries will have the same subidentifiers (except for the "column" subidentifier) up to the end of the encoded owner index. To configure VACM to permit access to this portion of the table, one would create vacmViewTreeFamilyTable entries with the value of vacmViewTreeFamilySubtree including the owner index portion, and vacmViewTreeFamilyMask "wildcarding" the column subidentifier. More elaborate configurations are possible.

The VACM access control mechanism described above provides control over SNMP access to Script MIB objects. There are a number of other access control issues that are outside of the scope of this MIB. For example, access control on URLs, especially those that use the file scheme, must be realized by the underlying operating system. A mapping of the owner index value to a local operating system security user identity should be used by an implementation of this MIB to control access to operating system resources when resolving URLs or executing scripts.

11. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

12. Acknowledgments

This document was produced by the IETF Distributed Management (DISMAN) working group.

13. References

[1] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2271, January 1998.

[2] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[3] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[4] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[5] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[6] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[7] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[8] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[9] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[10] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[11] Case, J., Harrington D., Presuhn R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2272, January 1998.

[12] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2274, January 1998.

[13] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[14] Levi, D., Meyer, P. and B. Stewart, "SNMPv3 Applications", RFC 2273, January 1998.

[15] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2275, January 1998.

[16] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[17] Berners-Lee, T., Fielding, R. and L. Masinter, " Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[18] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[19] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[20] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[21] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

14. Editors' Addresses

David B. Levi
Nortel Networks
4401 Great America Parkway
Santa Clara, CA 95052-8185
U.S.A.

Phone: +1 423 686 0432
EMail: dlevi@nortelnetworks.com

Juergen Schoenwaelder
TU Braunschweig
Bueltenweg 74/75
38106 Braunschweig
Germany

Phone: +49 531 391-3683
EMail: schoenw@ibr.cs.tu-bs.de

16. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.