Network Working Group
Internet-Draft
Expires April 2000

J. Schoenwaelder
TU Braunschweig
22. October 1999

Operation-Types for SMIv2

<draft-irtf-nmrg-smi-ops-00.txt>

Status of this Memo

Copyright Notice

Abstract

This document defines an extension for the SMIv2 which allows to define operations. Operations can among other things be used to define install/remove operations on conceptual MIB tables. This can result in simpler and more efficient implementations of configuration management systems.

Warning

This document has not been written in order to specify a solution. Instead, this document has been written to stimulate (controversial) discussions within the NMRG (and elsewhere).

Table of Contents

1. Introduction

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules. These modules are written using an adapted subset of OSI's Abstract Syntax Notation One, ASN.1 (1988) [ASN1], termed the Structure of Management Information (SMI) [RFC2578].

When designing a MIB module, it is often useful to define operations on collections of related objects. Operations go beyond what is available with normal read and write access to individual objects:

- Operations have a defined name which makes it easier to communicate complex operations on MIB objects.

- Operations have well defined input and output parameters with ordering constraints which reduces variability and simplifies implementations.

- Operations can raise operation specific errors during their invocation.

- Operations define whether they create or delete rows in conceptual tables.

- The defined signatures of operations allow to implement tools that generate APIs and stub procedures for command responder as well as command generator applications.

Operations can be defined on arbitrary collections of objects. However, it is expected that operations will normally be defined only on closely related objects (e.g. objects contained in a single conceptual row) since this simplifies implementation in extensible agent environments.

2. Definitions

```
SNMPv2-OPS DEFINITIONS ::= BEGIN

IMPORTS ObjectName FROM SNMPv2-SMI;

  OPERATION-TYPE MACRO ::=
  BEGIN
    TYPE NOTATION ::=
                  ArgumentsPart
                  ErrorsPart
                  ResultsPart
                  CreatesPart
                  DeletesPart
                  "STATUS" Status
                  "DESCRIPTION" Text
                  ReferPart

    VALUE NOTATION ::=
                  value(VALUE ObjectName)

    ArgumentsPart ::=
                  "ARGUMENTS" "{" Parameters "}"
                | empty

    ErrorsPart ::=
                  "ERRORS" "{" NamedNumbers "}" -- INTEGER enumerations
                | empty

    ResultsPart ::=
                  "RESULTS" "{" Parameters "}"
                | empty

    CreatesPart ::=
                  "CREATES" "{" Rows "}"
                | empty

    DeletesPart ::=
                  "DELETES" "{" Rows "}"
                | empty

    Parameters ::=
                  Parameter
                | Parameters "," Parameter

    Parameter ::=
                  identifier Syntax

    Syntax ::=
```

type

```
                | "BITS" "{" NamedBits "}"

    NamedBits ::=
                  NamedBit
                | NamedBits "," NamedBit

    NamedBit ::=
                  identifier "(" number ")" -- number is nonnegative

    NamedNumbers ::=
                  NamedNumber
                | NamedNumbers "," NamedNumber

    NamedNumber ::=
                  identifier "(" number ")"

    Rows ::=
                  Row
                | Rows "," Row
    Row ::=
                  value(ObjectName)

    Status ::=
                  "current"
                | "deprecated"
                | "obsolete"

    ReferPart ::=
                  "REFERENCE" Text
                | empty

    -- a character string as defined in [RFC2578]
    Text ::= value(IA5String)
  END
END
```

3. Mapping of the OPERATION-TYPE macro

The OPERATION-TYPE macro is used to define an operation. An operation has a defined signature which consists of the operation name, the types and names of the arguments, the types and names of the results and the operation specific errors that can occur while invoking the operation. Operations can create or delete rows in conceptual tables. The optional CREATES and DELETES clauses of the OPERATION-TYPE macro identify the tables in which rows are created or deleted. It should be noted that the expansion of the OPERATION-TYPE macro is something which conceptually happens during implementation and not during run-time.

3.1. Mapping of the ARGUMENTS clause

The ARGUMENTS clause, which need not be present, defines an ordered sequence of the types and names that form the arguments of the operation. The operation's DESCRIPTION clause must specify the information/meaning conveyed by each argument listed in the ARGUMENTS clause.

The name of an argument must consist of one or more letters or digits, and its initial character must be a lower-case letter. Argument names must be unique across all arguments and results defined in a single invocation of an OPERATION-TYPE macro.

3.2. Mapping of the ERRORS clause

The ERRORS clause, which need not be present, defines the operation specific errors that can occur while invoking the operation. Errors are represented as integer-valued named-number enumerations. Note that although it is recommended that error values start at 1 and be numbered contiguously, any valid value for an INTEGER in the range (1 to 2147483647 decimal) is allowed for an error value and, further, error values need not be contiguously assigned. The no error case does not need to be enumerated.

A label for a named-number enumeration within the ERRORS clause must consist of one or more letters or digits, up to a maximum of 64 characters, and the initial character must be a lower-case letter. (However, labels longer than 32 characters are not recommended.) Note that hyphens are not allowed.

3.3. Mapping of the RESULTS clause

The RESULTS clause, which need not be present, defines an ordered sequence of the types and names that form the results of the operation. The operation's DESCRIPTION clause must specify the information/meaning conveyed by each result listed in the RESULTS clause.

The name of a result parameter must consist of one or more letters or digits, and its initial character must be a lower-case letter. Result names must be unique across all arguments and results defined in a single invocation of an OPERATION-TYPE macro.

3.4. Mapping of the CREATES clause

The CREATES clause, which need not be present, defines the conceptual

table rows that will be created by invoking the operation. Rows are

identified by the descriptor which refers to a conceptual row, i.e. has a syntax which resolves to a SEQUENCE containing columnar objects. The operation's DESCRIPTION clause must specify how an implementation determines the instance identifier(s) of the row(s) created by the operation.

3.5. Mapping of the DELETES clause

The DELETES clause, which need not be present, defines the conceptual table rows that will be deleted by invoking the operation. Rows are identified by the descriptor which refers to a conceptual row, i.e. has a syntax which resolves to a SEQUENCE containing columnar objects. The operation's DESCRIPTION clause must specify how an implementation determines the instance identifier(s) of the row(s) deleted by the operation.

3.6. Mapping of the STATUS clause

The STATUS clause, which must be present, indicates whether this definition is current or historic.

The value "current" means that the definition is current and valid. The value "obsolete" means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value "deprecated" also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

3.7. Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present, contains a textual definition of the operation type which provides all semantic definitions necessary for implementation and use, and should embody any information which would otherwise be communicated in any ASN.1 commentary annotations associated with the object.

3.8. Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a textual cross-reference to some other document, either another information module or some other document which provides additional information relevant to this definition.

3.9. Mapping of the OPERATION-TYPE value

The value of an invocation of the OPERATION-TYPE macro is the name of the operation type, which is an OBJECT IDENTIFIER, an administratively assigned name.

3.10. Usage Examples

The first example shows an operation which creates or allocates a new entry in the vacmSecurityToGroupTable [RFC2575].

```
vacmCreateSTGEntry OPERATION-TYPE
    ARGUMENTS   {
                  securityModel SnmpSecurityModel (1..2147483647),
                  securityName  SnmpAdminString (SIZE(1..32)),
                  groupName     SnmpAdminString (SIZE(1..32)),
                  storageType   StorageType {
                                  volatile(2),
                                  nonVolatile(3)
                                }
                }
    ERRORS      { rowAlreadyExists(1) }
    CREATES     { vacmSecurityToGroupEntry }
    DESCRIPTION
        "This operation installs and activates a new entry in
         the vacmSecurityToGroupTable. The new entry is identified
         by the securityModel and securityName parameters.

         Implementations that do not support the creation of
         new rows but which have permanent rows that are
         currently notInService are expected to allocate
         and activate one of these notInService rows. The
         value of the storageType parameter is ignored in
         this case.

         The rowAlreadyExists(1) error is returned if the row
         identified by securityModel and securityName already
         exists."
    ::= { vacmOperations 1 }
```

The second example shows the definition of an operation that can be used to remove a row from the vacmSecurityToGroupTable.

```
vacmRemoveSTGEntry OPERATION-TYPE
    ARGUMENTS   {
                  securityModel SnmpSecurityModel (1..2147483647),
                  securityName  SnmpAdminString (SIZE(1..32))
                }
    ERRORS      { rowDoesNotExist(1), readOnlyStorageType(2) }
    DELETES     { vacmSecurityToGroupEntry }
    DESCRIPTION
        "This operation removes an entry identified by the
         securityModel and securityName parameters from the
         vacmSecurityToGroupTable.
```

Entries which can not be removed (e.g. the value

```
            of vacmSecurityToGroupStorageType is permanent)
            will be disabled by changing the RowStatus column
            vacmSecurityToGroupStatus to notInService.

            The rowDoesNotExist(1) error is returned if the row
            identified by the securityModel and securityName parameters
            does not exist. The readOnlyStorageType(2) error is returned
            if the row cannot be removed or disabled since it is stored
            in ROM."
        ::= { vacmOperations 2 }
```

The third example shows the definition of an operation which deletes zero or more entries from the vacmSecurityToGroupTable. The argument of the operation contains a pattern which is matched against all vacmGroupName instances.

```
vacmRemoveSTGEntryByGroupName OPERATION-TYPE
    ARGUMENTS   {
                  pattern SnmpAdminString (SIZE(1..32))
                }
    RESULTS     { numberRemoved Unsigned32 }
    DELETES     { vacmSecurityToGroupEntry }
    DESCRIPTION
        "This operation removes all entries from the
         vacmSecurityToGroupTable where the vacmGroupName
         matches the pattern parameter. The comparison is an
         exact match where each byte must match exactly (no
         wildcarding).

         Entries that can not be removed (e.g. the value
         of vacmSecurityToGroupStorageType is permanent)
         will be disabled by changing the RowStatus column
         vacmSecurityToGroupStatus to notInService.

         Entries that can not be disabled (e.g. the value
         of vacmSecurityToGroupStorageType is readOnly)
         will be ignored.

         The operation returns the number of entries that
         were actually removed from or disabled in the
         vacmSecurityToGroupTable."
    ::= { vacmOperations 3 }
```

4. Revising Operation Type Definitions

An operation definition may be revised in any of the following ways:

(1) The ERRORS clause may have new enumerations added.

(2) A STATUS clause value of "current" may be revised as "deprecated" or "obsolete". Similarly, a STATUS clause value of "deprecated" may be revised as "obsolete". When making such a change, the DESCRIPTION clause should be updated to explain the rationale.

(3) A REFERENCE clause may be added or updated.

(4) Clarifications and additional information may be included in the DESCRIPTION clause.

(5) Entirely new operations may be defined, named with previously unassigned OBJECT IDENTIFIER values.

Otherwise, if the semantics of any previously defined operation are changed (i.e., if a non-editorial change is made to any clause other than those specifically allowed above), then the OBJECT IDENTIFIER value associated with that operation must also be changed.

Note that changing the descriptor associated with an existing operation is considered a semantic change, as these strings may be used in fielded implementations and APIs derived from the operation type definition.

5. Open Issues

1. The current version supports three different ways to define a parameter. There are valid reasons for all three alternatives.

   First, there is a need to have input parameters for operations that do not exactly match an existing object. It would be a burdon to require that all parameters be defined as objects since many would end up with a write-only access (which does not exist) anyway.

   Second, there is a need to refer to types that are implicitely defined as part of an object definition. A good example is ifAdminStatus, which is an implicitly defined enumeration without a proper name.

   Third, it is necessary to assign names to parameters in order to solve problems when a single type of implicit type shows up in a arguments or result clause more than once.

2. It is currently not allowed to return vectors as a result. There are several issues to consider in this case. The most important reason to allow vectors is that you can have an operation which

itself returns a sequence of operations to recreate the current

state. But the details how this would work are not yet fully worked out.

6. Security Considerations

This document defines the means to define operations on collections of MIB objects. The definition of operations has no direct security impact on the Internet.

7. Authors' Address

Juergen Schoenwaelder
TU Braunschweig
Bueltenweg 74/75
38106 Braunschweig
Germany

Phone: +49 531 391-3283
EMail: schoenw@ibr.cs.tu-bs.de

8. References

[ASN1] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8824, December, 1987

[RFC2575] Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999

[RFC2578] 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

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999

9. Full Copyright Statement