INTERNET-DRAFT L. Heintz
Cisco Systems, Inc.
15 June 2001

SNMP Row Operations Extensions

<draft-ietf-eos-snmp-rowops-01.txt>

Status of this Memo

Copyright Notice

Abstract

This document describes a set of extensions (protocol operations and textual conventions) to the existing SNMP framework architecture [RFC2571]. These extensions provide mechanisms for efficient creation, modification, deletion and retrieval of table rows and other information.

Table of Contents

1. The SNMP Network Management Framework

The SNMP Management Framework presently consists of five major components:

- An overall architecture, described in RFC 2571 [RFC2571].

- 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 RFC 1155 [RFC1155], RFC 1212 [RFC1212] and RFC 1215 [RFC1215]. The second version, called SMIv2, is described in RFC 2578 [RFC2578], RFC 2579 [RFC2579] and RFC 2580 [RFC2580].

- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in RFC 1157 [RFC1157]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [RFC1901] and RFC 1906 [RFC1906]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [RFC1906], RFC 2572 [RFC2572] and RFC 2574 [RFC2574].

- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in RFC 1157 [RFC1157]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [RFC1905].

- A set of fundamental applications described in RFC 2573 [RFC2573] and the view-based access control mechanism described in RFC 2575 [RFC2575].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 2570 [RFC2570].

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.

This memo specifies a MIB module that is compliant to the SMIv2. A 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.

2. Overview

This document describes a set of SNMP extensions to current protocol operations [RFC1905] to provide for efficient management operations (i.e. creating, modifying, deleting and retrieving table rows and other MIB data). In addition, a new textual convention, RowState, is defined to replace RowStatus in future MIBs. RowState maintains the ability to report the state of a row, but does not provide a mechanism to create or delete a row.

2.1. Terms

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 [RFC2119].

Terminology such as "leftmost" or "left" indicates a PDU component that is transmitted on the wire prior to other components. Thus, terms such as "rightmost" imply components that have similar, but opposite semantics.

Protocol operation refers to a high-level request, such as a SetRequest or GetRequest (or one of the six new requests defined within this document). Row operation refers to one logical operation that affects one specific table row. A protocol operation may contain one or more row operations. The term RowOp refers to the aggregate parts of a protocol operation that comprise a single row operation.

2.2. Motivations for the Extensions

Experience has shown that current SNMP protocol operations and management structures are not ideally suited to effect configuration changes within managed devices and when retrieving large amounts of information. The extensions described in this document are specifically designed to minimize, or provide opportunities to minimize, the following problems which inhibit the effectiveness of SNMP:

- Requests that contain multiple varbinds that affect one row of a table may contain significant redundancy of information. This is because each varbind contains an object name (i.e. Object Identifier or OID), and these OIDs may differ only in the

single subid that designates a specific column. In cases where strings are used as instance identifiers, for example, UDP maximum-message-size constraints may force multiple SetRequests to be used to construct a new, or modify an existing row in a table. Requests containing redundant data are also more costly to encrypt and decrypt.

- SetRequests may contain multiple varbinds that actually refer to the same MIB object. For example, varbind one may be attempting to set the object, foo, to the value 1, while varbind two may be attempting to set the same object, foo, to the value 2. In such cases, the SNMP protocol indicates that implementations may make independent decisions as to which value will be used.

- SetRequests do not impose any ordering requirements on the varbinds within a single request, even if they affect different objects in the same row of a table. This can cause added complexity in set request processing.

- The RowStatus textual convention [RFC1903] provides a mechanism for row management. RowStatus most often requires the implementation of a rather complicated state machine, many of whose transitions are optional and whose target states are at times non-deterministic. RowStatus also confuses the notion of row status with the notion of row fate, which also tends to complicate both the MIB design and the implementation.

2.3. Design Goals

Several goals were identified when considering the kind and nature of extensions that were needed:

- allow separate row operations (hereafter referred to as RowOps) to be performed in a single protocol operation.

- allow operations on individual scalar and/or tabular objects in conjunction with row operations.

- minimize redundant information in a protocol operation. The extensions should at least make use of OID suppression techniques to meet this goal. Note that OID suppression (largely an issue of how data is stored within a PDU) is not equivalent to OID compression (data compression algorithms). Issues of OID compression are considered out of scope for this document.

- eliminate the need for special MIB objects (e.g. RowStatus) that control the creation and deletion of rows.

- minimize the impact on existing network management and subagent protocols such as SNMPv3, AgentX, and related applications.

- interoperate with legacy MIBs as well as future MIBs.

- operate in existing SNMP networks and not disrupt legacy SNMP-capable devices.

3. The Extensions

Six new protocol operations are described in this document: CreateRowRequest-PDU (aka CreateRow), DeleteRowRequest-PDU (aka DeleteRow), EditRowRequest-PDU (aka EditRow), GetRowRequest-PDU (aka GetRow), GetNextRowRequest-PDU (aka GetNextRow), and GetBulkRowRequest-PDU (aka GetBulkRow). These requests are based on the BulkPDU structure as originally defined in [RFC1905].

Despite the obvious use of the word "row" throughout this document, these new protocol operations can also be used to create, delete, modify and retrieve non-row data (i.e. scalars); specific details are provided later in the document.

For purposes of discussion, the three requests, CreateRow, DeleteRow and EditRow are more generically referred to as SetRow class requests, and GetRow, GetNextRow and GetBulkRow are referred to as RetrieveRow class requests.

The CreateRow request can only succeed if it is used to create an aggregate (e.g. row) that doesn't already exist. EditRow has no such constraints and can be used much in the same way traditional SetRequests are used. One noteworthy feature of the GetBulkRow request is that it can perform head and/or tail functions on a table. In addition, all SetRow requests make use of a three-phase processing model in which a new third phase (retrieve) is added to the traditional test and commit phases discussed in [RFC1905].

Another extension defined in this document is the RowState textual convention, which is intended to replace RowStatus in all future MIB designs. It is not the intention to deprecate RowStatus. RowState serves to reestablish a distinction between SNMP data types and SNMP operations -- a line which is blurred in the current RowStatus definition.

3.1. RowState

The RowState textual convention defined in this document is intended to replace RowStatus in new MIBs and provides several important improvements over RowStatus:

- RowState relaxes some of the row timeout rules that RowStatus suffers from. Such rules inhibit the usefulness of RowStatus as a means of temporarily placing system resources (i.e. table rows) out of service. For example, if an SNMP manager changes a given instance of snmpNotifyRowStatus from Active to NotInService as a means of temporarily disabling one or more notifications, an unintended side-effect of this action on some implementations may be that the row is automatically deleted after some short amount of time has elapsed (typically, a few minutes).

- More importantly, RowState separates the notion of reporting row status and the notion of managing row fate (i.e. creation & deletion). Specifically, the purpose of RowState is to enable reporting of row state, while matters of creating and deleting rows are better served by protocol operations.

RowState provides three states: NotReady, NotInService and Active, which are very similar to the corresponding RowStatus definitions. Unlike RowStatus, RowState does not provide CreateAndWait, CreateAndGo or Destroy.

Any entity providing a RowState column in a table must instantiate an instance of RowState when one or more other column objects in the same row have been created or instantiated (by whatever means). Using the new protocol operations defined in this document, it is usually unnecessary to directly set or reference a RowState instance in order to create and activate a new row.

3.2. Protocol Operations

The new protocol operations are designed to "fit" in the existing BulkPDU structure [RFC1905]. Formal definitions of the new protocol operations are provided in section 6.

This document specifies an evolutionary approach in the way the new protocol operations are encoded or contained within the BulkPDU. The new requests capitalize on OID suppression techniques to minimize information redundancy and therefore minimize PDU size. Note that the traditional SNMP protocol operations [RFC1905] are not being changed, either in their semantics, the way data is encoded within them, or

the way they are processed.

The following general discussion centers on how the varbinds of the new protocol operations and their responses are constructed and generally used. The elements of procedure formally describes how the new requests are processed and how the responses are constructed. The APPENDIX provides high-level examples of protocol exchanges using the extensions.

Each varbind in a request serves one of three purposes: as a Singleton, as a RowIdentifier or as an Operand.

3.2.1. Singletons

A varbind that serves as a Singleton functions more or less like varbinds in a traditional SNMP request (e.g. SetRequest, GetRequest). They are encoded in the same manner, and except for prefix-level inheritance (described below), they do not take advantage of OID suppression techniques. Singletons may exist in any SetRow or RetrieveRow class request or response, but if they do exist in any request, they MUST be the first N varbinds in the request, where N is provided in the non-repeaters field of the request. N may be zero to indicate that Singletons have not been included in the request. If N Singletons are included in a request, then the same N Singletons will also exist in the response, although in a successful response to a GetBulkRow request, additional Singletons may also be present.

The set of Singletons contained in a request SHOULD be ordered so that the name of Singleton i is lexicographically smaller than Singleton i+1, though SNMP applications MUST be prepared to accept unordered Singletons.

Singletons may be included within a CreateRow or a DeleteRow request. Such Singletons indicate a specific object instance within an aggregate (e.g. a row or a group scalar) upon which a creation or deletion operation is performed. By default, the operation takes affect on all object instances within the same aggregate; however, aggregate semantics may be defined (within the relevant description clauses) as to whether only the directly referenced objects in the Singletons are instead affected. Usually, Singletons are not included within CreateRow and DeleteRow requests, but there are cases when they may be needed (one of the protocol exchange examples in the APPENDIX illustrates such a case).

Within any EditRow class request, Singletons function much like individual varbinds within a conventional SetRequest, and the name of any Singleton MUST NOT refer to any other Singleton or to any

existing (or not) object within any RowOp of the same EditRow request.

Within the GetRow request, Singletons function much like individual varbinds within a conventional GetRequest.

Within the GetNextRow request, Singletons function much like individual varbinds within a conventional GetNextRequest.

Within the GetBulkRow request, Singletons function much like the non-repeaters varbinds in the conventional GetBulk request; thus, they function as GetNextRequest varbinds.

The remaining varbinds in any request form one or more independent RowOps with each varbind serving as either a RowIdentifier or as an Operand component of a RowOp.

3.2.2. The RowOp

The new requests defined in this document may generally contain zero or more RowOps. This allows a single CreateRow request, for example, to create one or more new rows in a single protocol operation. Each RowOp corresponds to one attempt to create a row, in this case, or corresponds to one attempt to delete a row in the case of DeleteRow, and so forth.

Note that the three layers in the diagram below do not describe different sections of the request, rather, they each represent the same information and structure (at different levels of abstraction).

```
<CreateRow.............>
<RowOp1><RowOp2><RowOp3>
<vb><vb><vb><vb><vb><vb>
```

Although the above diagram shows a CreateRow request logically containing three RowOps (i.e. create three rows) with two consecutive varbinds per RowOp, in reality, these requests may contain zero to many RowOps, each of which may may be comprised of a differing number of varbinds.

The first varbind in each RowOp is hereafter referred to as the RowIdentifier of a RowOp. The remaining varbinds in a given RowOp provide the individual operands (i.e. the affected row objects), which are hereafter referred to as Operands. In the diagram above, the 1st, 3rd and 5th varbinds are therefore RowIdentifiers and the 2nd, 4th and 6th varbinds are Operands.

3.2.3. The RowIdentifier

The following diagram shows a GetRow request containing a single RowOp that itself is composed of the required RowIdentifier and two Operands. In this case, the GetRow request is seeking to retrieve two specific column objects from a specific row.

```
<GetRow.....................................>
<RowOp......................................>
<vb1-RowIdentifier><vb2-Operand1><vb3-Operand2>
```

A traditional OID for a given object instance in a table can be broken up into these logical components:

```
OID = <TableName><1><ObjectSubid><Instance>
```

ObjectSubid is equivalent to one of the column descriptors. A more formal representation for RowIdentifier is now possible:

```
RowIdentifier = <vb.name=AggName,
                 vb.type=OID,
                 vb.value=1.0.Instance>
```

The component, AggName (Aggregate Name), takes the same value of TableName.

The vb.type MUST specify a type of OID. This is because the instance part of an OID is actually comprised of one or more table index values, each of which resolves to one or more subids.

Because the Instance always resolves to zero or more subids (because there are cases when Instance is optional within a RowIdentifier), and because a valid OID must be composed of at least two subids, Instance is prefixed with the OID value of 1.0. The reason for this choice of prefixes is that every valid table object name contains the value of <1.ObjectSubid> as part of its OID.

The AggName of a RowIdentifier MUST NOT contain partial OIDs. This means that the AggName MUST always exactly correspond to a legal table definition (if the desired results are to be achieved). The motivation behind this is that all RowOps are performed at the row level on a specific table.

The Instance within a GetNextRow is not required to be fully formed except that if the value is non-NULL, it MUST at least contain the 1.0 prefix. Also note that GetNextRow RowOps do not "jump" to the next table. In other words, in the event a GetNextRow RowOp refers to the last row in a given table, the appropriate exception is

returned for that RowOp even if other tables follow that contain retrievable rows. In this sense, GetNextRow RowOps are limited to operate within the subtree of the targeted table(s), though the Singletons within a GetNextRow or GetBulkRow request have no such constraint.

3.2.4. RowIdentifiers in GetBulkRow Requests

GetBulkRow RowIdentifiers have a slightly different construction than the RowIdentifiers for the other five request types. The diagram below shows two new components are included in these RowIdentifiers, InstanceLen and Instance2.

```
RowIdentifier = <vb.name=AggName,
                 vb.type=OID,
                 vb.value=1.0.InstanceLen.Instance.Instance2>
```

The two instance components define the search range of the GetBulkRow request (i.e. the rows from which data may be returned). Unlike the traditional GetBulk request, which may operate across one or more tables and/or scalar objects, each RowOp within a GetBulkRow request is confined to operate only within the table specified by the AggName component, although each RowOp may operate on different tables. Furthermore, each RowOp operates only within the range of rows as specified by the two instance components (inclusive) and upon one or more columns within that row range, as specified by the Operands that are associated with the RowOp.

The InstanceLen component specifies the number of subids that comprise Instance. InstanceLen may be 0, which indicates that Instance is not provided and that the search range begins at the first row (head) of the table and ends with the row specified by Instance2. Instance or Instance2 need not resolve to existing rows.

The number of subids in Instance2 may be 0 or more, and if 0 (i.e. Instance2 is not provided), this indicates that the search range includes any row that may correspond to the row specified by Instance and all those that follow (i.e. to the end of the table).

If Instance2 is lexicographically identical to Instance, the search range is confined to the one row common to both instances, or if identical values do not resolve to an existing row, a NuSuchInstance error is returned. If Instance2 is lexicographically smaller than Instance, a NoSuchInstance error will be returned.

The max-repetitions field in the GetBulkRow request may contain a non-zero value (M) to indicate a head or tail function should be

performed for each RowOp in the request. Otherwise, max-repetitions MUST always be zero (for all request types). If InstanceLen contains the value 0 (i.e. Instance is not provided), at most M rows will be returned from the head of the table. If InstanceLen contains a non-zero value (i.e. Instance is provided), then at most M rows at the tail of the table will be returned. For a head function, all returned rows MUST be lexicographically smaller than or equal to Instance2, if provided. For a tail function, all returned rows MUST be lexicographically greater than or equal to Instance; and Instance2, if provided, is ignored. The Instance2 component of a given RowOp, therefore, SHOULD NOT be provided in a GetBulkRow request whose max-repetitions and InstanceLen values are both non-zero as this unnecessarily increases PDU size.

Whether the GetBulkRow request indicates a head or tail function (or neither), it should be noted that the vb.value can be provided as either 1.0.0 or as 1.0 to indicate that the desired search range includes the entire table. The latter OID is slightly more efficient, and both forms have the same semantics unless a head or tail function is indicated (max-repetitions is non-zero). If max-repetitions is non-zero and the vb.value = 1.0.0, this indicates a head function whose search range is the entire table. If max-repetitions is non-zero and the vb.value = 1.0, this indicates a tail function whose search range is the entire table.

3.2.5. RowIdentifier Inheritance

Several forms of OID inheritance are possible within RowIdentifiers in order to further minimize PDU size: name-level, prefix-level, and instance-level inheritance.

In order to further minimize information redundancy, the OID of 0.0 is permitted to be substituted as the AggName component of any RowIdentifier to indicate that the most recent (left) RowIdentifier whose AggName is dissimilar (not 0.0) contains the OID value intended to be used.

In this example, a simplified notation is used to help illustrate how RowOps (the two middle ones) use the inheritance OID to minimize PDU size. This example shows four RowOps, each comprised of one RowIdentifier and one Operand (op):

```
[<foo><op>] [<0.0><op>] [<0.0><op>] [<fum><op>]
```

The following is logically identical, though it forms a larger PDU:

```
[<foo><op>] [<foo><op>] [<foo><op>] [<fum><op>]
```

In order for an inheritance OID to be correctly used, all RowOps that affect the same table MUST be consecutively placed in the varbind list, and also the first such RowIdentifier MUST NOT contain the inheritance OID. The above is known as name-level inheritance.

If a RowIdentifier of any request type has a NULL varbind value, this indicates that the most recent non-NULL varbind value is to be inherited. If the first RowIdentifier in a request contains a NULL varbind value, the OID of 1.0 will be substituted. This is known as instance-level inheritance.

The third form of inheritance (prefix-level inheritance) occurs whenever the varbind name of any AggName of a RowIdentifier begins with the OID prefix of 0.0 AND four or more subids exist in the OID. In such cases, the OID value of 1.3.6.1 is to be substituted in place of the 0.0 prefix.

The example below shows two RowOps, each comprised of one RowIdentifier and one Operand (op). Name- and prefix-level inheritance are used:

[<0.0.foo><op>] [<0.0><op>]

The following is logically identical, though it forms a larger PDU:

[<1.3.6.1.foo><op>] [<1.3.6.1.foo><op>]

The following OID is not a candidate for prefix-level inheritance because it has less than four subids: 0.0.1

Each RowOp is fully independent of any other despite any inheritance it may use.

3.2.6. The Operands

The remaining varbinds for a given RowOp are referred to as its Operands and are formed as standard request varbinds except that the name of each varbind is an OID whose length is either two or three subids long, as follows:

0.ObjectSubid (where ObjectSubid > 0 and ObjectSubid <= 39)

or

0.1.ObjectSubid (where ObjectSubid == 0 or ObjectSubid > 39)

Each Operand contained in the same RowOp SHOULD be ordered according

to ascending values of the ObjectSubids.

By way of example, if a table, foo, had four columns whose descriptors (ObjectSubids) were 0, 1, 39 and 40, their Operand names would be 0.1.0, 0.1, 0.39 and 0.1.40, respectively. Because the majority of MIB object descriptors are between 1 and 39, typical Operand names will take the form of 0.X.

3.2.7. Extended Error Codes

In addition to the standard error-status and error-index values in a response PDU, and varbind-level exceptions, an implementation MAY provide application-specific (extended) error indications in any response by inserting one new varbind at the end of the varbind list. Only the last varbind MAY contain extended error information.

Extended error information may be included in the PDU whether or not other errors/exceptions are indicated in the response (e.g. error-status).

The name of any varbind containing an extended error code is constructed as follows:

```
0.0.<ErrorInfo><ErrorInfo><...>
```

Each ErrorInfo component is constructed as follows:

```
0.ErrorCode[.<VbList>]
```

The ErrorInfo component may repeat multiple times but MUST have at least one occurrence. The leading subid of 0 in the ErrorInfo component serves as a separator between successive ErrorInfo components.

ErrorCode is a value in the range 1..4294967295.

The optional VbList is one or more subids which indicates a list of varbind positions (one per subid) within the same response PDU that associate to the preceding ErrorCode. The subids MUST be a value in the range of 1..max-bindings [RFC1905].

The value component of a varbind containing an extended error information is always NULL.

A properly formed varbind name will have a prefix of 0.0.0. This prefix allows EOS-enabled SNMP applications to distinguish these varbinds from all other varbind types (i.e. Singletons,

RowIdentifiers or Operands).

The ErrorCode of 3 in the following example indicates the 4th and 5th varbinds raised that error.

```
vb.name = 0.0.0.3.4.5
vb.name = NULL
```

The same varbinds MAY be associated to one or more extended error codes. The following example indicates an error code of 2 (for varbinds 1 and 2), an error code of 5 (no specific varbinds provided), an error code of 8 (for varbinds 2, 6 and 7) and an error code of 9 (for varbind 8). An error code that does not have a VbList (e.g. 5 in the above example) maps to a condition that applies to the entire PDU as opposed to a set of varbinds.

```
vb.name = 0.0.0.2.1.2.0.5.0.8.2.6.7.0.9.8
vb.name = NULL
```

The set of all returned errors and exceptions indicated within a Request-PDU is comprised of the error indicated in error-status (and error-index), if any, the extended errors codes, if any, and the exceptions present in any varbind, if any. Each error and/or exception is independent of any other.

An implementation SHOULD NOT provide extended error information if the desired information can instead be adequately conveyed using the error-status/index fields or varbind-level exceptions. An implementation MAY use extended error codes to convey a more detailed error status than is otherwise possible using traditional error/exception mechanisms. For example, if a wrongValue error is returned for varbind 5, an implementation may also choose to provide an extended error value to indicate that the provided value was "TooLarge".

The presence of extended error codes (or lack thereof) in a response does not guarantee all possible errors in the original request have been identified (or that there are no unknown errors). The presense of extended error information does not determine success/failure of a protocol operation. Rather, the error-status of noError implies a successful operation, whereas all other error-status values imply failure.

The method used to impart meaning to the returned error values is out of scope to this document. See [TBD] for details.

3.2.8. RowOps and Group Scalar Objects.

It is also possible to use the new requests to access or modify groups of scalar objects where each RowOp relates to a specific group. The only exception to this is that GetBulkRow RowOps are limited to operating on tables only (though the Singletons within a GetBulkRow may reference scalar objects).

To form a valid RowOp that refers to a group, the following MUST be done: First, the AggName instead refers to a valid group OID (e.g. the system group OID is 1.3.6.1.2.1.1), second, the RowIdentifier Instance prefix value is always provided as 0.0 (instead of 1.0 as for tables) and third, there is no Instance provided. The following example shows the relevant OIDs within a RowOp which accesses the system group scalars:

```
RowIdentifier = <vb.name=AggName=0.0.2.1.1, -- 1.3.6.1.2.1.1
                 vb.type=OID,
                 vb.value=0.0>              -- not a table
Operand1      = 0.1        -- sysDescr
Operand2      = 0.2        -- sysObjectID
Operand3      = 0.3        -- sysUptime
Operand4      = 0.4        -- sysContact
Operand5      = 0.5        -- sysName
Operand6      = 0.6        -- sysLocation
Operand7      = 0.7        -- sysServices
```

A CreateRow request can be used for group scalars if the affected scalars MAX-ACCESS values are read-create and the objects do not exist.

A DeleteRow request can be used for group scalars if the affected scalars MAX-ACCESS values are read-create.

An EditRow request can be used for group scalars if the affected scalars MAX-ACCESS values are either read-write or read-create.

The GetRow or GetNextRow requests may also be used for group scalars.

The following shows a request with three RowOps using instance-level inheritance for group scalar access:

```
[<system:0.0><op>]
[<snmpTargetObjects:NULL><op>]
[<snmpEngine:NULL><op>]
```

The following is logically identical to the above example, though it forms a larger PDU:

```
[<system:0.0><op>]
[<snmpTargetObjects:0.0><op>]
[<snmpEngine:0.0><op>]
```

3.2.9. Three-Phase Sets

All SetRow requests are processed using a three-phase processing model instead of the two-phase model defined in [RFC1905]. The new third phase (retrieve Operand values) occurs only if the first phase (test) completes successfully. If the test phase results in an error, all Operand values in the response will be set to NULL or appropriate errors and exceptions will be returned. Otherwise, all Operand values in the response will contain the value of the indicated object as it exists at the start of the third-phase (i.e. after the completion of the commit phase) or may contain exceptions if the Operand value is unavailable. In the event any Operand refers to an object which is known to have existed at the start of the first phase but not at the start of the third phase (as is possible with DeleteRow), the value of such Operands will be NULL in the final response.

3.2.10. Response PDUs

The traditional Response-PDU [RFC1905] is used as the standard response to each of the SetRow and RetrieveRow requests, except that the varbinds are constructed using the same OID suppression techniques as described above and, except for a successful GetBulkRow response, the number and type of varbinds (though not necessarily their names and values) returned are identical to the original request. Any response MAY also contain an additional varbind (the last) containing extended error information. The structure and contents of GetBulkRow responses are a bit more unpredictable.

Singletons within a GetBulkRow (or any) request always have corresponding Singletons in the response (i.e. the first N varbinds MAY be Singletons). The RowOps in a GetBulkRow request, however, are not returned in a successful response; instead, new RowOps and/or Singletons are inserted into the response after varbind N. A RowOp will be included only if it contains two or more Operands, whereas a Singleton will be included if a RowOp with only one Operand from a given row would otherwise be returned. Thus, RowOps and Singletons may be interspersed together in the response, although in all cases they are provided in row by row order.

The returned row information MUST be ordered so that RowOp i is lexicographically smaller than or equal to RowOp i+1. RowOps with

similar AggName values MUST also be sub-ordered in lexicographically ascending order based on their Instance values. Any two returned RowOps MUST NOT contain identical AggName and Instance values.

Similarly, for all new Singletons included after varbind N, Singleton i MUST be lexicographically smaller than Singleton i+1 and they MUST be interspersed with RowOps so that all RowIdentifier and Singleton varbinds (after varbind N) are ordered in lexicographically ascending order. In addition, Singletons MUST NOT be inserted between any RowIdentifier and one of its Operands.

If an otherwise successful response to a GetBulkRow request ever fails to include all the data requested (due to PDU size limitations), then the response MAY include an appropriate extended error indicating this fact. In this way, the last RowOp (or Singleton) in the response contains an Instance that can be used to calculate the beginning search range for a subsequent GetBulkRow request.

SNMP applications which generate GetBulkRow requests MUST preserve (i.e. remember) the value of the non-repeaters field for each request where it was non-zero. This allows the application to easily and correctly distinguish the first N (N = non-repeaters) Singletons from subsequent response components.

3.2.11. Determining Varbind Roles

Varbinds function as either Singletons, RowIdentifiers, Operands, or as carriers of extended error information. It is important for an implementation to correctly determine the role of every varbind in any given request or response. The following procedure is used to correctly identify the role of a given varbind, V, where V is a value in the range (1..max-bindings) in a request or response, and where N = non-repeaters in the associated request. The first rule below that applies determines the correct role of the varbind.

- If V <= N, and N > 0, V is a Singleton.
- If V is the last varbind in a Response-PDU, and V has a name prefixed by the value, 0.0.0, V contains extended error information.
- If the name of V is comprised of exactly two subids and this name is the value 0.X (where X > 0 and X <= 39), V is an Operand.

- If the name of V is comprised of exactly three subids and this name is the value 0.1.X (where X = 0, or X > 39), V is an Operand.

- If V is in any request or response other than a successful GetBulkRow response, V is a RowIdentifier.

- If V is in a successful GetBulkRow response and the varbind V+1 exists, and varbind V+1 is an Operand, then V is a RowIdentifier.

- If V is in a successful GetBulkRow response and the varbind V+1 exists, and varbind V+1 is NOT an Operand, then V is a Singleton.

- Otherwise, V is a Singleton

4. Elements of Procedure

4.1. CreateRow Request Processing

TBD

4.2. DeleteRow Request Processing

TBD

4.3. EditRow Request Processing

TBD

4.4. GetRow Request Processing

TBD

4.5. GetNextRow Request Processing

TBD

4.6. GetBulkRow Request Processing

TBD

4.7. Response-PDU Processing

TBD

5. Coexistence and Transition

An essential requirement for these operations is that they must operate seamlessly in existing networks and not disrupt legacy SNMP devices. This is satisfied by the fact that the new protocol operations have new and unique ASN.1 tags, which allow older implementations to efficiently and silently drop these new requests.

Some entities may only support these extensions for certain tables. For example, different AgentX subagents may or may not support these operations. This requires that the requests fail whenever a table is targeted that cannot support the new operation. The elements of procedure indicate the proper exceptions in these cases.

It is also possible that some table implementations may support only some subset of the new operations, for example, the RetrieveRow requests, but not the SetRow requests.

In an ideal world, the extensions herein would be easily translatable to traditional operations. However, this is not the case for the CreateRow, DeleteRow and GetBulkRow requests. On the other hand, it is believed the GetRow, GetNextRow and EditRow requests can be translated into conventional request formats (or subagent operations).

One possible transition strategy is to enable existing SNMP applications to support GetRow, GetNextRow and EditRow requests (with or without the traditional requests). A proxy application, for example, could bridge an EOS network with a non-EOS network by translating those EOS requests into conventional requests (and responses). Similarly, an AgentX master agent could translate this same subset of EOS requests into conventional AgentX requests (and responses). The benefits achieved with this strategy would be to obtain smaller PDU sizes on the network along with a 100% reuse of existing instrumentation methods (and subagent protocols). Subsequent support of the CreateRow, DeleteRow and GetBulkRow requests is always possible at a future date, or on a subagent by subagent basis (assuming a compatible subagent protocol has been adopted).

The procedures to translate EOS requests into conventional operations, where possible, are out of scope of this document.

It is RECOMMENDED, however, that new SNMP implementations support (for all MIB objects) all of the RetrieveRow class of requests. It is further RECOMMENDED that new implementations supporting any of the SetRow class of requests instead support all of the SetRow requests (for all applicable MIB objects).

Also, it is RECOMMENDED that new SNMP MIBs SHOULD use the RowState textual-convention in lieu of RowStatus, if applicable.

Proxy applications wishing to fulfill these recommendations can only serve as EOS request forwarders for the CreateRow, DeleteRow and GetBulkRow requests, and as request translators (or forwarders) for the other requests.

6. Protocol Operations Definitions

```
SNMP-ROWOP-PDUS-MIB DEFINITIONS ::= BEGIN

IMPORTS
    BulkPDU
        SNMPv2-PDU;

    ROWOP-PDUs ::=
        CHOICE {
        create-row-request
            CreateRowRequest-PDU,

        delete-row-request
            DeleteRowRequest-PDU,

        edit-row-request
            EditRowRequest-PDU,

        get-row-request
            GetRowRequest-PDU,

        get-next-row-request
            GetNextRowRequest-PDU

        get-bulk-row-request
            GetBulkRowRequest-PDU
        }

    CreateRowRequest-PDU ::=
```

```
          [9]
              IMPLICIT BulkPDU

      DeleteRowRequest-PDU ::=
         [10]
              IMPLICIT BulkPDU

      EditRowRequest-PDU ::=
         [11]
              IMPLICIT BulkPDU

      GetRowRequest-PDU ::=
         [12]
               IMPLICIT BulkPDU

      GetNextRowRequest-PDU ::=
         [13]
              IMPLICIT BulkPDU

      GetBulkRowRequest-PDU ::=
         [14]
              IMPLICIT BulkPDU

   END
```

7. Managed Object Definitions

```
   SNMP-ROWOP-MIB DEFINITIONS ::= BEGIN

   IMPORTS
       MODULE-IDENTITY, OBJECT-TYPE,
       OBJECT-IDENTITY,
       snmpModules                          FROM SNMPv2-SMI
       TEXTUAL-CONVENTION                   FROM SNMPv2-TC
       MODULE-COMPLIANCE, OBJECT-GROUP      FROM SNMPv2-CONF;

   snmpRowopMIB MODULE-IDENTITY
       LAST-UPDATED "200106151500Z"
       ORGANIZATION "EOS Working Group"
       CONTACT-INFO "WG-EMail:   eos@ops.ietf.org
                     Subscribe:  eos-request@ops.ietf.org

                     Co-Chair:   Dale Francisco
                     EMail:      dfrancisco@acm.org
                     phone:      +1 408-324-1389

                     Co-Chair:   Glenn Waters
```

```
                            Nortel Networks
                     EMail:     gww@nortelnetworks.com

                     Editor:    Lauren Heintz
                                Cisco Systems, Inc.
                     postal:    130 Baytech Drive
                                San Jose, CA 95134
                                USA
                     EMail:     lheintz@cisco.com
                     phone:     +1 408-853-6568
                    "
        DESCRIPTION  "The SNMP Row Operations MIB"
        REVISION     "200106151500Z"
        DESCRIPTION  "The initial version, published in
                      draft-ietf-eos-snmp-rowops-01.txt.
                     "
        ::= { snmpModules TBD }

    -- Textual Conventions

    RowState ::= TEXTUAL-CONVENTION
        STATUS       current
        DESCRIPTION "
               This textual-convention provides a means
               to represent the state of a conceptual row;
               it does not provide a means to manage creation
               and/or destruction of conceptual rows.

               In any row which includes a definition of RowState
               that is also mandatory, that row MUST contain
               an instantiation of a RowState object if any other
               object in the same row is instantiated.

               A status column of this type has three defined
               values:

                  - `active', which indicates that the
                  conceptual row is in use by the managed
                  device and is considered fully operational
                  from a management perspective;

                  - `notInService', which indicates that the
                  conceptual row is available for use by
                  the managed device but is NOT considered
                  fully operational from a management
                  perspective; a row in this state contains
                  all information necessary for a transition to
                  the active state;
```

- `notReady', which indicates that the conceptual row is not available for use by the managed device, perhaps because the row is missing information or the row has been explicitly set to this state;

Any of the three states may be specified in a management protocol set operation. At any time, the value of an existing RowState object may be modified or changed to any other state unless the current value is notReady and the row (after completion of a management set operation) would not be consistent with an active state (i.e. information is missing or inconsistent and as a result the row could not assume an active state until further management set operations were performed). In this case, the value MUST remain at notReady and an inconsistentValue error is returned.

A RowState object whose value is notReady MUST implicitly promote itself from notReady to notInService if any management protocol set operation not referencing the status column explicitly changes the value of any other object in the same row AND the row afterward is now consistent with the active state (i.e all information needed to make the row active is available).

When a row does not exist, and a management protocol set operation causes any object in the row to be created, the status object will also be instantiated and its initial value will be determined by the first of the following rules that are satisfied:

1. if the set operation includes an initial value, the RowState object will take on the highest value (i.e. active is highest) consistent with the state definitions, but no higher than the provided value. For example, if a set operation contains a varbind where RowState=notInService (or active), the initial value will be notReady if the row is in an inconsistent state after the operation is complete; otherwise, it will be notInService (or active).

2. if the set operation does not include an initial

value, but the RowState definition does include a supported DEFVAL, the initial value will be the highest value consistent with the state definitions but no higher than the value specified in the DEFVAL, unless the DEFVAL specifies a value of notReady, which in this case, the DEFVAL is ignored (i.e. case 3 below instead applies).

3. otherwise, the set operation does not include an initial value, and the RowState definition does not include a DEFVAL, and the initial value will be set to the highest value consistent with the state definitions. Thus, the initial state will be active if the new row is consistent with that state, or it will be notReady otherwise.

No constraint is imposed on whether other objects in the same row can be modified, regardless of the value of the associated RowState object. If such constraints are desired, they MUST be explicitly stated in the DESCRIPTION clause of the status column. In the absence of such statements, the managed device MUST allow any object in any row to be modified at any time, notwithstanding the possibility that other MIB objects MAY also impose similar constraints. An inconsistentValue error is returned when an invalid attempt is made to alter a row whose state precludes such an operation.

RowState description clauses, in addition to the DESCRIPTION clauses of associated column objects, SHOULD together describe the general conditions under which a row can be made active. In the absence of such statements, any row SHOULD generally be capable of being made active at any time.

The agent must detect conceptual rows that have been in the notReady state for an abnormally long period of time and remove them. It is the responsibility of the DESCRIPTION clause of the status column to indicate what an abnormally long period of time would be. This period of time should be long enough to allow for human response time (including `think time') between the creation of the conceptual row and the setting of the status to `active'. In the absense of such information in the DESCRIPTION clause, it is suggested that this period be approximately

```
            5 minutes in length. This removal action applies not only
            to newly-created rows, but also to previously active rows
            which are set to, and left in, the notReady state for a
            prolonged period exceeding that which is considered normal
            for such a conceptual row."
        SYNTAX      INTEGER {
                        active(1),
                        notInService(2),
                        notReady(3)
                    }

    END
```

8. IANA Considerations

This document requires IANASnmpExtendedProtocol values to be reserved for allowing command responders to advertise their ability to support the extensions outlined in this document. IANASnmpExtendedProtocol values are administered by IANA. IANASnmpExtendedProtocol is defined in SNMP-X-PROTOCOL-TC.

9. 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.

10. Acknowledgements

This document is the result of the efforts of the Evolution Of SNMP (EOS) Working Group. Some special thanks are in order to the following EOS WG members for their ideas, efforts and asundry contributions:

Dr. Jeff Case
Sharon Chisholm
Dale Francisco
Sandra McLeod
Steve Moulton
David Perkins
Randy Presuhn
Jeurgen Schoenwaelder
Robert Story
Glenn Waters
Matt White

11. Security Considerations

TBD

12. References

[RFC1155] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", STD 16, RFC 1155, May 1990.

[RFC1157] Case, J., M. Fedor, M. Schoffstall and J. Davin, "The Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1212] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[RFC1901] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC2571] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2578] McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

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

[RFC2580] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC-PROTO] Presuhn, R., Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Protocol Operations for the Simple Network Management Protocol", <draft-ietf-snmpv3-update-proto-05.txt>, June 2001.

[RFC-TMM] Presuhn, R., Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Transport Mappings for the Simple Network Management Protocol", <draft-ietf-snmpv3-update-transmap-05.txt>, June 2001.

[RFC-MIB] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for the Simple Network Management Protocol", <draft-ietf-snmpv3-update-mib-05.txt>, June 2001.

[RFC-COEX] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", <draft-ietf-snmpv3-coex-v2-00.txt>, June 2001.

[RFC1909] McCloghrie, K., Editor, "An Administrative Infrastructure for SNMPv2", RFC 1909, February 1996.

[RFC1910] Waters, G., Editor, "User-based Security Model for SNMPv2", RFC 1910, February 1996.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January, 1998.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC2863] McCloghrie, K. and F. Kastenholz. "The Interfaces Group MIB." RFC 2863, June 2000.

[SNMP-MPD] Case, J., Harrington, D., Presuhn, R. and B. Wijnen,

"Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", <draft-ietf-snmpv3-mpd-v2-00.txt>, June 2001.

[SNMP-USM] Blumenthal, U. and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", <draft-ietf-snmpv3-usm-v2-00.txt>, June 2001.

[SNMP-ACM] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", <draft-ietf-snmpv3-vacm-04.txt>, February 1999. <draft-ietf-snmpv3-vacm-v2-00.txt>, June 2001.

[RFC-APPL] Levi, D., Meyer, P. and B. Stewart, "SNMP Applications", <draft-ietf-snmpv3-apps-v2-00.txt>, June 2001.

[RFC2570] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", <draft-ietf-snmpv3-intro-04.txt>, January 1999.

[RFC-COEX] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", <draft-ietf-snmpv3-coex-v2-00.txt>, June 2001.

13. Editor's Addresses

Lauren Heintz
Cisco Systems, Inc.
130 Baytech Drive
San Jose, Ca 95134

Phone: +1 408-853-6568
EMail: lheintz@cisco.com

APPENDIXES

A. Impact to SNMP and other Protocols

A.1. SNMPv3

An issue remains whether a new message processing model MUST be specified as part of the SNMPv3 (or later) standard. Otherwise, it is not seen that these extensions pose any impact to other SNMPv3 architectural components (i.e. USM, VACM) because the new protocol operations and their contents contain sufficient information (along with the information provided in whatever version-specific message wrapper they are contined within) to satisfy the abstract service interfaces for those components.

However, these extensions may not be fully compatible with the SNMPv3 proxy application (or any legacy SNMP application incorporating a message processing module that receives and processes or forwards SNMP messages).

A.2. AgentX

These extensions imply that AgentX will have to evolve in order to support the new protocol operations. For example, AgentX does not provide a delete PDU (to support DeleteRow), and neither does its TestSet PDU provide for a standard way to indicate whether the operation being performed maps to a CreateRow or EditRow request. Furthermore, master agents will need to know how to recognize and process the new protocol operations (i.e. distinguish RowIdentifiers from Operands, logically expand the targeted object OIDs and map them to subtree registrations).

The feature to allow extended error codes to be returned in response PDUs may also require AgentX to provide a means to communicate such information from the subagent(s) to the master agent.

B. Examples of Row Operations

B.1. CreateRow

Create two new rows in the snmpNotifyTable [RFC2573]. This table uses RowStatus, so we choose to explicitly set its value for each row (new impls may allow the request to omit a RowStatus varbind if its

value is createAndGo). The response includes the values of the specified Operands after row creation.

```
   CreateRow
   (
       request-id      = 1
       non-repeaters   = 0 -- 0 Singletons
       max-repetitions = 0

    -- RowOp 1

       -- RowIdentifier
       vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
       vb1.value = 1.0.114.111.119.49 -- "row1"

       vb2.name  = 0.2 -- snmpNotifyTag
       vb2.value = "tag1"

       vb3.name  = 0.3 -- snmpNotifyType
       vb3.value = 1 -- trap

       -- skip snmpNotifyStorageType. Use DEFVAL

       vb4.name  = 0.5 -- snmpNotifyRowStatus
       vb4.value = createAndGo

    -- RowOp 2

       -- RowIdentifier
       vb5.name  = 0.0 -- inherit snmpNotifyTable
       vb5.value = 1.0.114.111.119.50 -- "row2"

       vb6.name  = 0.3 -- snmpNotifyType
       vb6.value = 1 -- trap

       vb7.name  = 0.5 -- snmpNotifyRowStatus
       vb7.value = createAndWait
   )

   ResponsePdu
   (
       request-id   = 1
       error-status = noError

    -- RowOp 1

       -- RowIdentifier
       vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
```

```
          vb1.value = 1.0.114.111.119.49 -- "row1"

          vb2.name  = 0.2 -- snmpNotifyTag
          vb2.value = "tag1"

          vb3.name  = 0.3 -- snmpNotifyType
          vb3.value = 1 -- trap

          vb4.name  = 0.5 -- snmpNotifyRowStatus
          vb4.value = active

       -- RowOp 2

          -- RowIdentifier
          vb5.name  = 0.0 -- inherit snmpNotifyTable
          vb5.value = 1.0.114.111.119.50 -- "row2"

          vb6.name  = 0.3 -- snmpNotifyType
          vb6.value = 1 -- trap

          vb7.name  = 0.5 -- snmpNotifyRowStatus
          vb7.value = notInService
      )
```

This protocol exchange illustrates the use of the CreateRow request to create two new rows in the snmpNotifyTable [RFC2573] except that we pretend here that RowState was used in the design of that table instead of RowStatus.

```
      CreateRow
      (
          request-id      = 2
          non-repeaters   = 0 -- 0 Singletons
          max-repetitions = 0

       -- RowOp 1

          -- RowIdentifier
          vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb1.value = 1.0.114.111.119.49 -- "row1"

          vb2.name  = 0.2 -- snmpNotifyTag
          vb2.value = "tag1"

          vb3.name  = 0.3 -- snmpNotifyType
          vb3.value = 1 -- trap

          -- skip snmpNotifyStorageType. Use DEFVAL
```

```
          -- By omitting a RowState varbind, it is the
          -- same as setting RowState=Active.

       -- RowOp 2

          -- RowIdentifier
          vb4.name  = 0.0 -- inherit snmpNotifyTable
          vb4.value = 1.0.114.111.119.50 -- "row2"

          vb5.name  = 0.3 -- snmpNotifyType
          vb5.value = 1 -- trap

          -- Explicitly set RowState to an initial
          -- value because we don't want to go
          -- active just yet. Even though we hint
          -- for an initial value of NotInService,
          -- it's possible that the result might
          -- show NotReady (if the row as defined
          -- by this RowOp were not ready to go Active).
          -- If a DEFVAL (NotInService) were provided
          -- in the MIB, this varbind could be omitted
          -- from this RowOp.
          vb6.name  = 0.5 -- snmpNotifyRowState
          vb6.value = NotInService
      )

      ResponsePdu
      (
          request-id   = 2
          error-status = noError

       -- RowOp 1

          -- RowIdentifier
          vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb1.value = 1.0.114.111.119.49 -- "row1"

          vb2.name  = 0.2 -- snmpNotifyTag
          vb2.value = "tag1"

          vb3.name  = 0.3 -- snmpNotifyType
          vb3.value = 1 -- trap

       -- RowOp 2

          -- RowIdentifier
          vb4.name  = 0.0 -- inherit snmpNotifyTable
          vb4.value = 1.0.114.111.119.50 -- "row2"
```

```
        vb5.name  = 0.3 -- snmpNotifyType
        vb5.value = 1 -- trap

        vb6.name  = 0.5 -- snmpNotifyRowState
        vb6.value = NotInService
    )
```

B.2. DeleteRow

Delete the two rows created in either of the two previous examples. Note that the RowIdentifier in the second RowOp does not inherit the table OID from the first RowOp. Although this is legal, it also needlessly increases the request PDU size. Also note that the command responder instead chose to use name-level inheritance.

```
    DeleteRow
    (
        request-id      = 3
        non-repeaters   = 0 -- 0 Singletons
        max-repetitions = 0

     -- RowOp 1

        -- RowIdentifier
        vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
        vb1.value = 1.0.114.111.119.49 -- "row1"

     -- RowOp 2

        -- RowIdentifier
        vb2.name  = 1.3.6.1.6.3.13.1.1 -- snmpNotifyTable
        vb2.value = 1.0.114.111.119.50 -- "row2"
    )

    ResponsePdu
    (
        request-id   = 3
        error-status = noError

     -- RowOp 1

        -- RowIdentifier
        vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
        vb1.value = 1.0.114.111.119.49 -- "row1"

     -- RowOp 2
```

```
        -- RowIdentifier
        vb2.name  = 0.0 -- snmpNotifyTable
        vb2.value = 1.0.114.111.119.50 -- "row2"
    )
```

This DeleteRow example illustrates the use of Singletons. The Singletons allow a complete object instance to be provided (whereas the previous example contained no Operands and thus did not have access to the column descriptors, which are needed to form complete OIDs). If the user's VACM view only allowed him/her to delete certain rows in a table, the DeleteRow request would have to contain enough information to satisfy the isAccessAllowed service interface [RFC2573], and the above example does not. This can be accomplished by either providing an Operand within each RowOp or by instead using Singletons, which are more efficient than using RowOps in such cases.

```
    DeleteRow
    (
        request-id      = 4
        non-repeaters   = 2 -- 2 Singletons
        max-repetitions = 0

     -- Singleton 1

        -- delete snmpNotifyTag in row1, which
        -- in this case also deletes all of row1
        vb1.name  = 0.0.6.3.13.1.1.1.2.114.111.119.49
        vb1.value = NULL

     -- Singleton 2

        -- delete snmpNotifyTag in row2, which
        -- in this case also deletes all of row2
        vb1.name  = 0.0.6.3.13.1.1.1.2.114.111.119.50
        vb1.value = NULL
    )
```

B.3. GetRow

This is an example of a protocol exchange for a GetRow request and its response.

```
    GetRow
    (
        request-id      = 5
        non-repeaters   = 0 -- 0 Singletons
        max-repetitions = 0
```

```
       -- RowOp 1

          -- RowIdentifier
          vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb1.value = 1.0.114.111.119.49 -- "row1"

          vb2.name  = 0.2 -- snmpNotifyTag
          vb2.value = NULL

          vb3.name  = 0.3 -- snmpNotifyType
          vb3.value = NULL

      -- RowOp 2

          -- RowIdentifier
          vb4.name  = 0.0 -- inherit snmpNotifyTable
          vb4.value = 1.0.114.111.119.50 -- "row2"

          vb5.name  = 0.5 -- snmpNotifyRowStatus
          vb5.value = NULL

          -- must use the 0.1.X OID form because 999 > 39
          vb6.name  = 0.1.999 -- Should result in NoSuchObject
          vb6.value = NULL
      )

      ResponsePdu
      (
          request-id   = 5
          error-status = noError

       -- RowOp 1

          -- RowIdentifier
          vb1.name  = 0.0.6.3.13.1.1.1 -- snmpNotifyTable
          vb1.value = 1.0.114.111.119.49 -- "row1"

          vb2.name  = 0.2 -- snmpNotifyTag
          vb2.value =  "tag1"

          vb3.name  = 0.3 -- snmpNotifyType
          vb3.value = 1 -- trap

      -- RowOp 2

          -- RowIdentifier
          vb4.name  = 0.0 -- inherit snmpNotifyTable
          vb4.value = 1.0.114.111.119.50 -- "row2"
```

```
        vb5.name  = 0.5 -- snmpNotifyRowStatus
        vb5.value = NotInService

        vb6.name  = 0.1.999
        vb6.value = NoSuchObject
    )
```

This GetRow request includes one Singleton using prefix-level inheritance to retrieve the sysUpTime value along with one RowOp.

```
    GetRow
    (
        request-id      = 6
        non-repeaters   = 1    -- 1 Singleton
        max-repetitions = 0

     -- Singletons

        vb1.name  = 0.0.2.1.1.3.0 -- get(sysUpTime.0)
        vb1.value = NULL

     -- RowOp 1

        -- RowIdentifier
        vb2.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
        vb2.value = 1.0.114.111.119.49 -- "row1"

        vb3.name  = 0.2 -- snmpNotifyTag
        vb3.value = NULL

        vb4.name  = 0.3 -- snmpNotifyType
        vb4.value = NULL
    }

    ResponsePdu
    (
        request-id   = 6
        error-status = noError

     -- Singletons

        -- Singleton 1. The original request
        -- contained non-repeaters = 1. This
        -- count is not transmitted back in the
        -- response, so the originating command generator
        -- must maintain this count and associate it
        -- to the returned request-id.
        vb1.name  = 0.0.2.1.1.3.0 -- sysUpTime.0
```

```
          vb1.value = 123456

       -- RowOp 1

          -- RowIdentifier
          vb2.name  = 0.0.6.3.13.1.1.1 -- snmpNotifyTable
          vb2.value = 1.0.114.111.119.49 -- "row1"

          vb3.name  = 0.2 -- snmpNotifyTag
          vb3.value =  "tag1"

          vb4.name  = 0.3 -- snmpNotifyType
          vb4.value = 1 -- trap
      )
```

B.4. GetNextRow

This is an example of a protocol exchange for a GetNextRow request and its response.

```
   GetNextRow
   (
       request-id      = 7
       non-repeaters   = 1 -- 1 Singleton
       max-repetitions = 0

    -- Singletons

       vb1.name  = 0.0.2.1.1.3 -- getNext(sysUpTime)
       vb1.value = NULL

    -- RowOp 1

       -- RowIdentifier
       vb2.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
       vb2.value = 1.0.114.111.119 -- "row" (partial instance)

       vb3.name  = 0.2 -- snmpNotifyTag
       vb3.value = NULL

       vb4.name  = 0.3 -- snmpNotifyType
       vb4.value = NULL

   -- RowOp 2

       -- RowIdentifier
       vb5.name  = 0.0 -- inherit snmpNotifyTable
```

```
          vb5.value = 1.0

          -- No Operands, this RowOp is logically identical
          -- to  GetNext(snmpNotifyEntry). A bit inefficient
          -- since it produces a response RowOp containing
          -- only one Operand (would be better as a Singleton)..
      )

      ResponsePdu
      (
          request-id   = 7
          error-status = noError

       -- Singletons

          vb1.name  = 0.0.2.1.1.3.0 -- sysUpTime.0
          vb1.value = 123457

       -- RowOp 1

          -- RowIdentifier
          vb2.name  = 0.0.6.3.13.1.1.1 -- snmpNotifyTable
          vb2.value = 1.0.114.111.119.49 -- "row1"

          vb3.name  = 0.2 -- snmpNotifyTag
          vb3.value =  "tag1"

          vb4.name  = 0.3 -- snmpNotifyType
          vb4.value = 1 -- trap

      -- RowOp 2

          -- RowIdentifier
          vb5.name  = 0.0 -- inherit snmpNotifyTable
          vb5.value = 1.0.114.111.119.49 -- "row1"

          vb6.name  = 0.2 -- snmpNotifyTag
          vb6.value = "tag1"
      )
```

The RowOps in this GetNextRow request are logically the same as a getNext(snmpNotifyEntry), getNext(snmpNotifyTag) and getNext(system). The fourth RowOp is illegally constructed and would return a genErr error, because although the RowIdentifier is correctly formed, in this case it contains an Instance without at least one Operand, and without an Operand it is impossible to decide which table object the getNext should be performed on. Note that it would be more efficient to instead provide these types of RowOps as individual Singletons.

```
GetNextRow
(
    request-id      = 8
    non-repeaters   = 0 -- 0 Singletons
    max-repetitions = 0

 -- RowOp 1, same as getNext(snmpNotifyEntry)

    -- RowIdentifier
    vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
    vb1.value = 1.0

 -- RowOp 2, same as getNext(snmpNotifyTag)

    -- RowIdentifier
    vb2.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
    vb2.value = 1.0

    vb3.name  = 0.2 -- snmpNotifyTag
    vb3.value = NULL

 -- RowOp 3, same as getNext(system)

    -- RowIdentifier
    vb4.name  = 0.0.2.1.1 -- system
    vb4.value = 0.0       -- not a table

 -- RowOp 4, same as getNext(snmpNotifyEntry)

    -- RowIdentifier (missing Operand produces genErr)
    vb5.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
    vb5.value = 1.0.114.111.119.49 -- "row1"
}
```

B.5. GetBulkRow

This GetBulkRow request includes one Singleton using prefix-level inheritance to retrieve the sysUpTime value along with two RowOps. Though both of the RowOps in this request are legal, note that RowOp2 returns a subset of RowOp1's desired values. Therefore, RowOp2 could be omitted without affecting the response.

```
GetBulkRow
(
    request-id      = 9
    non-repeaters   = 1 -- 1 Singleton
    max-repetitions = 0 -- head/tail disabled
```

```
       -- Singletons

          vb1.name  = 0.0.2.1.1.3 -- getNext(sysUpTime)
          vb1.value = NULL

       -- RowOp 1

          -- RowIdentifier
          -- Retreive all snmpNotifyTags and
          -- RowStatus objects from
          -- row1 to end of table (Instance2 not provided).
          vb2.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb2.value = 1.0.4.114.111.119.49 -- "row1"

          vb3.name  = 0.2 -- snmpNotifyTag
          vb3.value = NULL

          vb4.name  = 0.5 -- snmpNotifyRowStatus
          vb4.value = NULL

       -- RowOp 2

          -- RowIdentifier
          -- Retreive all snmpNotifyTags and
          -- RowStatus objects from
          -- row1 to row3 (inclusive)
          vb5.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb5.value = 1.0.4.114.111.119.49.114.111.119.51

          vb6.name  = 0.2 -- snmpNotifyTag
          vb6.value = NULL

          vb7.name  = 0.5 -- snmpNotifyRowStatus
          vb7.value = NULL
      }

      ResponsePdu
      (
          request-id   = 9
          error-status = noError

       -- Singletons

          vb1.name  = 0.0.2.1.1.3.0 -- sysUpTime.0
          vb1.value = 123458

       -- RowOp 1
```

```
          -- RowIdentifier
          vb2.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb2.value = 1.0.4.114.111.119.49 -- "row1"

          vb3.name  = 0.2 -- snmpNotifyTag
          vb3.value = "tag1"

          vb4.name  = 0.5 -- snmpNotifyRowStatus
          vb4.value = Active

       -- RowOp 2

          -- RowIdentifier
          vb5.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb5.value = 1.0.4.114.111.119.50 -- "row2"

          vb6.name  = 0.2 -- snmpNotifyTag
          vb6.value = "tag2"

          vb7.name  = 0.5 -- snmpNotifyRowStatus
          vb7.value = Active

       -- RowOp 3

          -- RowIdentifier
          vb8.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb8.value = 1.0.4.114.111.119.51 -- "row3"

          vb9.name  = 0.2 -- snmpNotifyTag
          vb9.value = "tag3"

          vb10.name  = 0.5 -- snmpNotifyRowStatus
          vb10.value = Active

       -- RowOp 4

          -- RowIdentifier
          vb11.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb11.value = 1.0.4.114.111.119.52 -- "row4"

          vb12.name  = 0.2 -- snmpNotifyTag
          vb12.value = "tag4"

          vb13.name  = 0.5 -- snmpNotifyRowStatus
          vb13.value = Active

       -- RowOp 5
```

```
        -- RowIdentifier
        vb14.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
        vb14.value = 1.0.4.114.111.119.53 -- "row5"

        vb15.name  = 0.2 -- snmpNotifyTag
        vb15.value = "tag5"

        vb16.name  = 0.5 -- snmpNotifyRowStatus
        vb16.value = Active
    }
```

This GetBulkRow request is the same as the previous example except that RowOp2 specifies a column not included in RowOp1. The response would be identical to the above response except that RowOps1-3 (row 1-3) would include a Operand/value for snmpNotifyType.

```
    GetBulkRow
    (
        request-id      = 10
        non-repeaters   = 1 -- 1 Singleton
        max-repetitions = 0 -- head/tail disabled

     -- Singletons

        vb1.name  = 0.0.2.1.1.3 -- getNext(sysUpTime)
        vb1.value = NULL

     -- RowOp 1

        -- RowIdentifier
        -- Retreive all snmpNotifyTags and
        -- RowStatus objects from
        -- row1 to end of table (Instance2 not provided).
        vb2.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
        vb2.value = 1.0.4.114.111.119.49 -- "row1"

        vb3.name  = 0.2 -- snmpNotifyTag
        vb3.value = NULL

        vb4.name  = 0.5 -- snmpNotifyRowStatus
        vb4.value = NULL

     -- RowOp 2

        -- RowIdentifier
        -- Retreive all snmpNotifyTypes and
        -- RowStatus objects from
        -- row1 to row3 (inclusive)
```

```
        vb5.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
        vb5.value = 1.0.4.114.111.119.49.114.111.119.51

        vb6.name  = 0.3 -- snmpNotifyType
        vb6.value = NULL

        vb7.name  = 0.5 -- snmpNotifyRowStatus
        vb7.value = NULL
    }
```

This GetBulkRow request includes two RowOps to show how to perform a head and tail function, respectively.

In RowOp1 (head function), the search range is indicated by the OID 1.0.0 (no Instance or Instance2 values provided). That value along with a non-zero max-reps value indicates a head function with a possible search range of the entire table.

In RowOp2 (tail function), the search range is indicated by the the OID 1.0 (no InstanceLen, Instance or Instance2 values provided). That value along with a non-zero max-reps value indicates a tail function with a possible search range of the entire table.

The Response indicates the desired objects in the first and last two rows of the table are returned (five rows in table, so row3 is not returned).

Note that if max-reps were instead 0, the two RowOps would be logically identical.

```
   GetBulkRow
   (
       request-id      = 11
       non-repeaters   = 0 -- 0 Singletons
       max-repetitions = 2 -- head/tail enabled

    -- RowOp 1

       -- RowIdentifier
       vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
       vb1.value = 1.0.0    -- head search entire table

       vb2.name  = 0.2 -- snmpNotifyTag
       vb2.value = NULL

       vb3.name  = 0.5 -- snmpNotifyRowStatus
       vb3.value = NULL
```

```
       -- RowOp 2

          -- RowIdentifier
          vb4.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb4.value = 1.0      -- tail search entire table

          vb5.name  = 0.2 -- snmpNotifyTag
          vb5.value = NULL

          vb6.name  = 0.5 -- snmpNotifyRowStatus
          vb6.value = NULL
      }

      ResponsePdu
      (
          request-id   = 11
          error-status = noError

       -- RowOp 1

          -- RowIdentifier
          vb1.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb1.value = 1.0.4.114.111.119.49 -- "row1"

          vb2.name  = 0.2 -- snmpNotifyTag
          vb2.value = "tag1"

          vb3.name  = 0.5 -- snmpNotifyRowStatus
          vb3.value = Active

       -- RowOp 2

          -- RowIdentifier
          vb4.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb4.value = 1.0.4.114.111.119.50 -- "row2"

          vb5.name  = 0.2 -- snmpNotifyTag
          vb5.value = "tag2"

          vb6.name  = 0.5 -- snmpNotifyRowStatus
          vb6.value = Active

       -- RowOp 3

          -- RowIdentifier
          vb7.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb7.value = 1.0.4.114.111.119.52 -- "row4"
```

```
          vb8.name  = 0.2 -- snmpNotifyTag
          vb8.value = "tag4"

          vb9.name  = 0.5 -- snmpNotifyRowStatus
          vb9.value = Active

       -- RowOp 4

          -- RowIdentifier
          vb10.name  = 0.0.6.3.13.1.1 -- snmpNotifyTable
          vb10.value = 1.0.4.114.111.119.53 -- "row5"

          vb11.name  = 0.2 -- snmpNotifyTag
          vb11.value = "tag5"

          vb12.name  = 0.5 -- snmpNotifyRowStatus
          vb12.value = Active
      }
```

This GetBulkRow requests data from two columns of the same table. The response shows RowOps and Singletons interspersed with one another (i.e. rows with only one object to return cause Singletons to be returned instead of RowOps, which be definitions MUST have two or more Operands in GetBulkRow responses). Although this example uses a shorthand notation (i.e. fooTable), the actual OIDs would utilize OID inheritance wherever possible. Also note that row3 did not produce a result.

```
   GetBulkRow
   (
       request-id      = 12
       non-repeaters   = 0 -- 0 Singletons
       max-repetitions = 0 -- head/tail disabled

    -- RowOp 1

       -- RowIdentifier
       vb1.name  = fooTable
       vb1.value = 1.0 -- search entire table

       vb2.name  = 0.2 -- fooColumnA
       vb2.value = NULL

       vb3.name  = 0.3 -- fooColumnB
       vb3.value = NULL
   }

   ResponsePdu
```

```
      (
          request-id   = 12
          error-status = noError

       -- RowOp 1

          -- RowIdentifier
          vb1.name  = fooTable
          vb1.value = 1.0.1 -- row 1

          vb2.name  = 0.2 -- fooColumnA
          vb2.value = 1

          vb3.name  = 0.3 -- fooColumnB
          vb3.value = 1

       -- Singleton 1

          vb4.name  = fooColumnA.2 -- row 2
          vb4.value = 2

       -- Singleton 2

          vb5.name  = fooColumnB.3 -- row 4
          vb5.value = 4

       -- RowOp 2

          -- RowIdentifier
          vb6.name  = fooTable
          vb6.value = 1.0.5 -- row 5

          vb7.name  = 0.2 -- fooColumnA
          vb7.value = 5

          vb8.name  = 0.3 -- fooColumnB
          vb8.value = 5

       -- Singleton 3

          vb9.name  = fooColumnA.6 -- row 6
          vb9.value = 6
      }
```

B.6. EditRow

This EditRow hypothetically produces the shown response which

indicates several errors: first, varbind 2 caused a wrongValue error, secondly, varbind 2 also caused an application specific error 11 (perhaps tooManySlicesInserted), and finally, varbinds 3 and 6 both caused yet another application specific error 14 (perhaps tempTooHigh). Additionally, the returned Operand values are set to NULL to indicate the third-phase (get) was not executed (due to test-phase failure).

```
EditRow
(
    request-id      = 13
    non-repeaters   = 0 -- no Singletons
    max-repetitions = 0

 -- RowOp 1

    -- RowIdentifier
    vb1.name  = toasterTable
    vb1.value = 1.0.1   -- toaster/row "1"

    vb2.name  = 0.2 -- toasterSlicesInserted
    vb2.value = 100

    vb3.name  = 0.3 -- toasterTemp
    vb3.value = 1000

 -- RowOp 2

    -- RowIdentifier
    vb4.name  = toasterTable
    vb4.value = 1.0.2   -- toaster/row "2"

    vb5.name  = 0.2 -- toasterSlicesInserted
    vb5.value = 2

    vb6.name  = 0.3 -- toasterTemp
    vb6.value = 1000
}

ResponsePdu
(
    request-id   = 13
    error-status = 10 (wrongValue)
    error-index  = 2

 -- RowOp 1

    -- RowIdentifier
```

```
          vb1.name  = toasterTable
          vb1.value = 1.0.1   -- toaster/row "1"

          vb2.name  = 0.2 -- toasterSlicesInserted
          vb2.value = NULL

          vb3.name  = 0.3 -- toasterTemp
          vb3.value = NULL

       -- RowOp 2

          -- RowIdentifier
          vb4.name  = toasterTable
          vb4.value = 1.0.2   -- toaster/row "2"

          vb5.name  = 0.2 -- toasterSlicesInserted
          vb5.value = NULL

          vb6.name  = 0.3 -- toasterTemp
          vb6.value = NULL

       -- Extended errors:

          vb7.name  = 0.0.0.11.2.0.14.3.6
          vb7.value = NULL
      }
```

C. Known issues

This section will be deleted before becoming an RFC.

These are known issues that need to be resolved before going standards track:

- Should the coexistence and transition section be moved to a different document?

- Need to specify a way for extended error codes to be mapped to meaningful error messages. Perhaps IANA would maintain an IANAExtendedErrorCodes textual-convention (along the lines of IANAIfType) and all extended error codes would thus be controlled via IANA. Also, a small range of the codes may need to be reserved up-front for use by the IETF so that the standard SNMP error-status codes and varbind-level exceptions can also be mapped to extended code values. For example, we may want to return "tooBig" in the case where a successful GetBulkRow response is returned with only a portion of the

requested data. The tooBig extended error is a "more" signal to perform at least one more GetBulkRow (using the Instance in the last RowOp as a beginnning search range).

- What other extended error codes should be reserved and defined up front? For example, perhaps an "unsupportedOperation" needs to be defined so that the EOS requests could return that error on a RowOp by RowOp basis (where a given subagent is non-EOS capable and others are). In such cases, incrementing the droppedPdu counter may not be possible.

- How can SetRequests and/or EditRows be used to safely create rows in tables using RowState in multi-mgr environments? It would be ideal to have such a mechanism (without re-creating RowStatus all over again) in order to ease the transition to EOS-capable networks.

- Should this document contain procedures to translate EOS operations to/from conventional operations?

- Should the MIB defs in this document (and all other EOS documents) instead be located in one EOS document? This could reduce the number of modules needed to be IMPORTED in future MIBs/appls.

D. Full Copyright Statement

Acknowledgement

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