EOS Working Group
INTERNET-DRAFT
draft-ietf-eos-oops-00.txt

W. Hardaker
Network Associates Laboratories

February 2003

Object Oriented Protocol Operations for the Simple Network Management Protocol draft-ietf-eos-oops-00.txt

Status of this Memo

Copyright Notice

Abstract

This document defines new Protocol Data Units (PDUs) for use within the Simple Network Management Protocol (SNMP). The goals of the new PDUs are to reduce packet sizes and to reduce processing overhead required at the Command Responder side of the protocol.

Table of Contents

1. Introduction

This document specifies some new PDU types to optimize specific operations in SNMP. This draft is still work in progress, though the technical concepts are perceived by the author to be stable and mostly implementable at this point. More feedback from implementors sought at this point in time. Comments and discussion on this document are encouraged and should take place on the EOS working group mailing list (eos@ops.ietf.org).

1.1. Background and Motivations

Many recognized problems [5, 6] exist within the current PDUs defined for use by the SNMP protocol [1]. Many of these known problems are addressed by the new protocol operations defined within this draft. The general problems solved by this draft are described briefly in this section.

1.1.1. GET-NEXT complexity

Many problems result from a device's inability to access its data in a way that can be efficiently traversed when searching for the next object in a series of GET-NEXT or GET-BULK SNMP requests. Internally, many implementations differ in how data is stored in a device and thus it would be impossible to define a MIB which would be efficient when implemented on every architecture.

The operations defined in this document do not suffer from these problems, as the data can be returned in any order selected by the Command Responder. It MUST, however, return data in a consistent manner such that if the same data is requested twice at any point in time, the Command Responder will return the data in the same order. It is the exact order in which objects are returned is implementation specific.

It is likely that every Command Responder will benefit from this design decision and that either most Command Generators don't care about ordering in the first place or have the resources to perform ordering themselves via utilization of database technologies that are not realistically available on many Command Responder platforms. Command Generator's that do need sorted data may even need the data sorted in a different manner than the MIBs specify, in which case the sorting burden placed on the Command Responder is wasted.

1.1.2. OID compression

SNMPv2 PDUs are constructed using a sequence of varbinds. In many cases, when multiple objects are requested from the Command Responder, the OIDs contained within have common prefixes. It is widely recognized as a waste of bandwidth to duplicate this prefix repeatedly.

The PDUs defined in this document only rely on a base OID (e.g., an object identifier pointing to the root of a SNMP Object or Table), and the sub-objects (e.g., a row in a SNMP Table) underneath it are defined as references from the base OID. This allows large groups of data to be transmitted while only requiring a single OID to identify the top most grouping object (e.g., a table).

1.1.3. Row operations

The ability to manipulate an entire row within a SNMP Table at once has long been a frustration of programmers using the SET operations contained the SNMPv2 PDUs.

The PDUs defined in this document allow explicit data dependencies to be marked within the write operations. This allows Command Generators to create more powerful requests, while at the same easing implementation burden within the Command Responder.

1.1.4. Complex object relationships

Many people do not properly understand the interrelationships between SNMP Tables and have desired the ability to express their data in a more hierarchal fashion, such that SNMP tables can contain other SNMP tables. Although it is not possible to enable nested datatypes within SMIv2 today, the SMIng working group is trying to solve these problems and create the ability to define more complex relationships in SMIv3.

The PDUs defined in this document allow hierarchal object data to be transmitted efficiently and as a single group, thus allowing the complex structures defined within SMIv3 to be efficiently transmitted.

1.1.5. Retrieval of more data than is desired

Extracting only the data needed from an SNMP Table using the GETNEXT or GETBULK operations available in SNMP PDUs today is difficult at best. The request PDUs defined in this document allow more precise selection data and allow simple search criteria to be submitted to the remote Command Responders to help reduce the amount of data returned to the Command Generator. A delicate balance is required to ensure that the devices being polled for data are not taxed with expensive search requests, so the criteria imposed within these documents is restricted to a limited set of operators that may or may not be supported by the command responder implementation. The intention is not to shift all the complex decision processing burden from the Command Generator station to the command responder, but to increase efficiency on the wire where possible.

1.1.6. Row hole traversal and object grouping

When requests to GET, GET-NEXT, and GET-BULK data come back in a SNMP RESPONSE PDU, the data can be hard to organize back into a logical structure again. Additionally, GET-BULK responses interleave its data with END-OF-MIB exceptions which adds further

complexity to the data processing.

Since the PDUs defined in this document deal directly with MIB objects as a group, the data relationships within objects are preserved. This makes transferring large amounts of data efficient on both the sending and receiving side. The data groupings are now appropriately marked within the packet itself.

1.1.7. Index parsing and OID length restrictions

Although the encoding and decoding of Table indexes into and out of OIDs is algorithmic, many SNMP software packages and toolkits make mistakes in implementing the algorithm. Additionally, since the maximum length of an OID is limited to 128 sub-identifiers, it limits the size of the index data that can be utilized within an SNMP table. This limitation results in MIBs which are poorly designed and/or broken and MIBs with less-than-ideal table indexes. Finally, it is impossible to decode the indexes from an OID without knowledge of the MIB module definition.

The indexes within the PDUs defined in this document are encoded directly into the packet as opposed to being encoded into the OID. This simplifies both Command Generator and Command Responder code and reduces the chances of one side incorrectly encoding or decoding the indexes. Additionally, because encoding of indexes is done directly within the protocol no MIB module definition is needed to extract them from the protocol operation. Finally, it provides the ability to use a set of indexes which is larger than the OID-encoded imposed length of 128 sub-identifies.

1.1.8. Transactions

The limited set of SNMP write operation transactions have been difficult to cope with when large sets of data must be pushed around, since all the objects within a SNMP SET PDU must be set as if simultaneously. In actuality, configuration data often contains independent sets of data.

The Write-Object-PDU operation defined in this document defines a less restrictive and more flexible transaction model that lets large quantities of data be pushed more efficiently through a network.

1.1.9. Easy conversion to and from popular data formats

A desire has been shown by network operators for SNMP objects to be easily accessible and convertible to and from more human friendly expression languages and storage systems, like XML or SQL Databases. The objects within the PDUs contained in this document are designed

for such a purpose. It should be, for example, trivial to convert them to and from a hierarchical XML schema set describing the objects in question. For the example, the BER hierarchal packet format can be quickly converted to XML by replacing the sequence tags with XML ASCII delimiter tags.

1.1.10. Better error condition handling

Integer error codes are extremely useful for machine parsibility and interoperability, but it's frequently nice to have an extra error string that may be passed to an operator to assist in extra debugging of problems. The response PDUs defined in this document contain an "error string" for exactly this purpose.

Additionally, multiple errors can be returned at once allowing more comprehensive error reporting. In older PDUs only the first error encountered in processing a request could be returned in the response PDU.

1.2. Related work within the SMIng working group

The PDUs in this document are designed for use with the first three versions of the SMI language. There are specific portions of the protocol operations that are not needed for SMIv1 or SMIv2 documents. Command Responders and Command Generators which do not make use of any documents written in the SMIv3 format may not need the SMIv3 portions of the protocol, but MUST expect the possibility that these PDUs may arrive with these extensions requested.

1.3. Terminology

The SMIv1, SMIv2, and SMIv3 documents are not entirely consistent in their terminology. In this document, we will use the following terminology throughout this document. Most of the terminology is equivalent to what is expected to be in the SMIv3 documents.

Note: this section is not entirely finished yet, as we're awaiting the latest SMIng draft to match our terminology to theirs.

Aggregate
: An aggregate is a collection of information. In SMIv2 terms, this is roughly equivalent to a table. In SMIv3 this is an array or other structure with multiple objects contained within it.

Object
: An object is a particular instance of "something". In SMIv2 terms, this would be a row. In SMIv3 this would be an

instantiated scalar or a particular row within an array.

Element
An element is one particular part of a an object. In SMIv2 terms, this would be a column. In SMIv3 this would be a XXX, which may be defined as another object in itself.

Command
This document sticks with the SNMPv3 application documentation [4] terminology for describing the different application types that request and receive the PDUs defined in this document.

2. Transport Protocol Considerations

The PDUs defined in this document allow the transmission of large data sets in a more compact format than previous SNMP PDUs allowed. However, it is still recommended that large requests and responses be transmitted over a SNMP transmission domain which provides for guaranteed network packet delivery (e.g., TCP). Large responses, containing many objects, carried over transmission domains which can not guarantee delivery (e.g., UDP) are still likely to be problematic. It is well beyond the scope of this document to redesign a reliable transmission mechanism.

3. PDU definitions

This section defines the new PDUs in ASN.1 format. Supporting textual descriptions of PDU components are given in later sections.

3.1. ASN.1 definitions for the PDUs

```
SNMP-OOPS DEFINITIONS ::= BEGIN

--
-- Notes: ignore the OptionField for now.  They'll be fixed later.
-- (for instance, they should all be unique definitions)
--

--
-- DataTypes
--

   ElementSyntax ::= CHOICE {
         simple           SimpleSyntax,
         application-wide ApplicationSyntax
   }
```

```
SimpleSyntax ::= CHOICE {
      integer-value   INTEGER (-2147483648..2147483647),
      string-value    OCTET STRING (SIZE (0..65535)),
      objectID-value  OBJECT IDENTIFIER
}

ApplicationSyntax ::= CHOICE {
      ipAddress-value          IpAddress,
      counter-value            Counter32,
      timeticks-value          TimeTicks,
      arbitrary-value          Opaque,
      big-counter-value        Counter64,
      unsigned-integer-value   Unsigned32,
      integer64-value          Integer64,
      unsigned64-value         Unsigned64,
      float32-value            Float32,
      float64-value            Float64,
      float128-value           Float128
}

IpAddress ::= [APPLICATION 0]
   IMPLICIT OCTET STRING (SIZE (4))

Counter32 ::= [APPLICATION 1]
   IMPLICIT INTEGER (0..4294967295)

Unsigned32 ::= [APPLICATION 2]
   IMPLICIT INTEGER (0..4294967295)

Gauge32 ::= Unsigned32

TimeTicks ::= [APPLICATION 3]
   IMPLICIT INTEGER (0..4294967295)

Opaque ::= [APPLICATION 4]
   IMPLICIT OCTET STRING

-- APPLICATION 5 was used in 1442

Counter64 ::= [APPLICATION 6]
   IMPLICIT INTEGER (0..18446744073709551615)

-- APPLICATION 7 was used in 1442

Integer64 ::= [APPLICATION 8]
   IMPLICIT INTEGER (-9223372036854775808..9223372036854775807)

Unsigned64 ::= [APPLICATION 9]
```

```
          IMPLICIT INTEGER (0..18446744073709551615)

       Float32 ::=
           [APPLICATION 9]
               IMPLICIT OCTET STRING(SIZE(4)) -- IEEE format

       Float64 ::=
           [APPLICATION 10]
               IMPLICIT OCTET STRING(SIZE(8)) -- IEEE format

       Float128 ::=
           [APPLICATION 11]
               IMPLICIT OCTET STRING(SIZE(8)) -- IEEE format

       max-bindings INTEGER ::= 2147483647

    --
    -- Get-Object-PDU definition
    --

    Get-Object-PDU ::=
      [XXX]                                          -- 9?
      SEQUENCE {
         request-id                Unsigned32,
         gop-flags                 GOPFlags,
         option-field              OptionField,
         request-objects           RequestObjects
      }

    GOPFlags ::=
        IMPLICIT OCTET STRING (SIZE(1))
    --        errorOnUnknowFlag(0),
    --        requestAcknowledgment(1),

    OptionField ::= SEQUENCE { } -- XXX needs formal definition

    RequestObjects ::=
        SEQUENCE (SIZE (0..max-bindings)) OF
            RequestObject

    RequestObject ::=
        SEQUENCE {
            max-return-objects     Unsigned32,
            skip-objects           Unsigned32,
            cursor                 OCTET STRING (SIZE(1..256)),
            request-flags          RequestCharacteristics,

            option-field           OptionField,
```

```
            request-base-OID        OBJECT IDENTIFIER,
            request-element-list    ElementsRequestList,
            search-criteria         SearchCriteria
        }

    RequestCharacteristics ::=
        IMPLICIT OCTET STRING (SIZE(1))
    --        errorOnUnknowFlag(0),
    --        restartOnInvalidCursor(1),
    --        returnAllDataOnSearchFailure(2)

    ElementsRequestList ::=
        SEQUENCE (SIZE(0..max-bindings)) OF
            ElementSpecifier

    ElementSpecifier ::=
        CHOICE {
            -- request an index
            index-number[0]             IMPLICIT INTEGER (0..4294967295)

            -- request an attribute
            element-number[1]           IMPLICIT INTEGER (0..4294967295)

            -- fully qualified for AUGMENTation tables
            -- 0.0 prefix = request-base-OID for SMIv3
            subelement-specifier[2]     IMPLICIT OBJECT IDENTIFIER,

            -- request multiple [sub,...]elements
            -- *only* usable in an ElementRequestList
            -- (not in ElementData)
            multiple-subelement[3]
                IMPLICIT SEQUENCE {
                    -- fully qualified for AUGMENTation tables
                    -- 0.0 prefix = request-base-OID for SMIv3
                    element-specifier       OBJECT IDENTIFIER,
                    element-list            ElementRequestList
                }

            -- used for referencing external indexes of augmentation
            -- tables or for SMIv3 sub attributes with external indexes
            -- This is for use *only* in a SearchCriteria
            subelement-index[4]
                IMPLICIT SEQUENCE {
                    -- fully qualified for AUGMENTation tables
                    -- 0.0 prefix = request-base-OID for SMIv3
                    element-specifier       OBJECT IDENTIFIER,
                    element-index-number    INTEGER (0..4294967295)
                }
```

```
        }

    SearchCriteria ::=
        SEQUENCE {
            match-type                  MatchType
            data
                CHOICE {
                    -- sub-sequences for logical operations
                    sub-criteria[0]     IMPLICIT SEQUENCE
                                            (SIZE(0..max-bindings))
                                                OF SearchCriteria,

                    -- match operation data
                    match-data[1]       IMPLICIT SEQUENCE {
                        which                       ElementSpecifier,
                        what                        ElementSyntax
                    }
                }
        }

    MatchType ::= INTEGER {
        -- any datatypes:
        equals(0),
        not-equals(1),              -- note: not-equals NULL ::= exists

        -- numerical only:
        lessThan(10),
        lessThanOrEqual(11),
        greaterThan(12),
        greaterThanOrEqual(13),

        -- binary comparisons:
        regexp(20),
        not-regexp(21),

        -- logical operations:
        logicalAND(100),
        logicalOR(101),
        logicalNOT(102) -- sub-criteria must be exactly 1 in length

        -- ... IANA assigned up to 255
        -- enterprise specific:
        --    256*EnterpriseID to
        --    256*EnterpriseID + 255
    }

    --
    -- Get-Object-Repsonse-PDU definition
```

```
    --

    Get-Object-Repsonse-PDU ::=
      [XXX]                                          -- 10?
      SEQUENCE {
         request-id                Unsigned32,
         gop-flags                 GOPFlags,
         option-field              OptionField,
         return-objects            ReturnObjects
      }


    ReturnObjects ::=
        SEQUENCE (SIZE(0..max-bindings))
            OF ReturnObject

    ReturnObject ::=
        SEQUENCE {
            error-information-list  ErrorInformationList,
            cursor                  OCTET STRING (SIZE(0..256)),
            response-flags          RequestCharacteristics,
            option-field            OptionField,

            request-base-OID        OBJECT IDENTIFIER,

            returned-data-list      DataList
        }

    -- should be an empty sequence if no errors occurred
    ErrorInformationList ::=
        SEQUENCE (SIZE(0..max-bindings)) OF
            ErrorInformation

    ErrorInformation ::=
        SEQUENCE {
            error-status            ErrorCode,
            error-index             OBJECT IDENTIFIER,
            error-string            OCTET STRING (SIZE(0..1024))
        }

    ErrorCode ::=
        INTEGER {
            -- From RFC3416:
            noError(0),
            tooBig(1),
            noSuchName(2),
            badValue(3),
            readOnly(4),
```

```
            genErr(5),
            noAccess(6),
            wrongType(7),
            wrongLength(8),
            wrongEncoding(9),
            wrongValue(10),
            noCreation(11),
            inconsistentValue(12),
            resourceUnavailable(13),
            commitFailed(14),
            undoFailed(15),
            authorizationError(16),
            notWritable(17),
            inconsistentName(18),

            -- new in this document:
            unknownFlag(100),
            invalidCursor(101),
            unSupportedSearchOperation(102),
            tooComplex(103),
            createFailed(104),
            deleteFailed(105),
            getConfigObjectNotSupported(106)

            -- reserved for enterprise usage:
            -- 256*EnterpriseID to 256*EnterpriseID + 255
            -- Enterprise errors MUST be treatable as a genErr for
            -- applications that don't understand the error enumeration.
        }


    -- old speak: multiple rows
    DataList ::=
        SEQUENCE (SIZE(0..max-bindings)) OF
            ObjectData

    -- old speak: multiple columns with a row
    ObjectData ::=
        SEQUENCE (SIZE(0..max-bindings)) OF
            ElementData

    ElementData ::=
        SEQUENCE {
            which-element           ElementSpecifier,
            element-value           ElementValue
        }

    ElementValue ::=
```

```
        CHOICE {
            base-type-object        ElementSyntax,

            -- for SMIv3 sub-object aggregates and augmentation tables:
            sub-object              DataList
        }

    --
    -- Write-Object-PDU definition
    --

    Write-Object-PDU ::=
      [XXX]                                          -- 11?
      SEQUENCE {
         request-id                 Unsigned32,
         write-flags                WriteSemantics,
         option-field               OptionField,
         write-transactions         WriteTransaction
      }

    WriteSemantics ::=
        OCTET STRING (SIZE(1))
    --          errorOnUnknowFlag(0),
    --          requestAcknowledgment(1),
    --          returnDataOnlyOnError(2)

    WriteTransaction ::=
       SEQUENCE {
          transaction-flags         TransactionSemantics,
          option-field              OptionField,

          transaction-data-list     TransactionDataList
       }

    TransactionSemantics ::=
       OCTET STRING (SIZE(1))
    --       errorOnUnknowFlag(0),
    --       needSuccess(1),
    --       needAll(2),
    --       notOrderDependent(3)

    TransactionDataList ::=
       SEQUENCE (SIZE(0..max-bindings)) OF
          TransactionData

    TransactionData ::=
       CHOICE {
           create-transaction[0]    CreateTransaction,
```

```
          modify-transaction[1]    ModifyTransaction,
          delete-transaction[2]    DeleteTransaction,
          method-transaction[3]    MethodTransaction
          sub-transaction          WriteTransaction
       }

    CreateTransaction ::= SEQUENCE {
       request-base-OID        OBJECT IDENTIFIER,
       create-data-list        DataList
    }

    ModifyTransaction ::= SEQUENCE {
       request-base-OID        OBJECT IDENTIFIER,
       modify-search-criteria  SearchCriteria,
       modify-data-list        DataList
    }

    DeleteTransaction ::= SEQUENCE {
       request-base-OID        OBJECT IDENTIFIER,
       delete-search-criteria  SearchCriteria,
    }

    MethodTransaction ::= SEQUENCE {
       method-OID              OBJECT IDENTIFIER,
       method-arguments        DataList
    }

    --
    -- Write-Object-Response-PDU definition
    --

    Write-Object-Response-PDU ::=
      [XXX]                                        -- 12?
      SEQUENCE {
         request-id                        Unsigned32,
         write-flags                       WriteSemantics,
         option-field                      OptionField,
         error-information-list            ErrorInformationList,

         write-transaction-results         WriteTransactionResults
      }

    WriteTransactionResults ::=
       SEQUENCE (SIZE(0..max-bindings)) OF
          WriteTransactionResponse

    WriteTransactionResult ::=
       SEQUENCE {
```

```
        transaction-flags         TransactionSemantics,
        option-field              OptionField,

        transaction-response-list TransactionResponseList
      }

   TransactionResponseList ::=
      SEQUENCE (SIZE(0..max-bindings)) OF
         TransactionResponse

   TransactionResponse ::=
      CHOICE {
          create-response[0]    CreateResponse,
          modify-response[1]    ModifyResponse,
          delete-response[2]    DeleteResponse
      }

   CreateResponse ::= {
      request-base-OID        OBJECT IDENTIFIER,
      create-data-list        DataList             -- copy of sent
   }

   ModifyResponse ::= {
      request-base-OID        OBJECT IDENTIFIER,
      modify-data-list        DataList             -- ONLY indexes of
                                                   -- modified elements
                                                   -- and data
   }

   DeleteResponse ::= {
      request-base-OID        OBJECT IDENTIFIER,
      delete-data-list        DataList             -- ONLY indexes
   }

   MethodResponse ::= SEQUENCE {
      method-OID              OBJECT IDENTIFIER,
      method-return-data      DataList
   }

   --
   -- Get-Configuration-Objects-PDU definition
   --

   Get-Configuration-Objects-PDU ::=
     [XXX]                                          -- 13?
     SEQUENCE {
        request-id                Unsigned32,
        gcop-flags                GCOPFlags,
```

```
        option-field              OptionField,
        request-objects           ConfigurationObjectsList
      }

    GCOPFlags ::=
        IMPLICIT OCTET STRING (SIZE(1))
    --        errorOnUnknowFlag(0),
    --        requestAcknowledgment(1),

    ConfigurationObjectsList ::=
        SEQUENCE OF (SIZE (0..max-bindings)) OF
             ConfigurationObjects

    ConfigurationObjects ::=
        SEQUENCE {
            request-flags         ConfigurationFlags,
            option-field          OptionField,
            requested-config      RequestedConfig
        }

    ConfigurationFlags ::=
        IMPLICIT OCTET STRING (SIZE(1))
    --        errorOnUnknowFlag(0),

    RequestedConfig ::= OBJECT IDENTIFIER

    --
    -- Configuration-Objects-Response-PDU definition
    --
    Configuration-Objects-Response-PDU ::=
      [XXX]                                          -- 14?
      SEQUENCE {
         request-id               Unsigned32,
         gcop-flags               GCOPFlags,
         option-field             OptionField,
         config-results           ConfigurationResults
      }

    ConfigurationResults ::=
      SEQUENCE (SIZE(0..max-bindings)) OF
        ConfigurationResult

    ConfigurationResult ::=
        SEQUENCE {
            request-flags         ConfigurationResultFlags,
            option-field          OptionField,
            error-information     ErrorInformationList,
            config-transactions   WriteTransaction
```

```
        }

    --
    -- Notification-Objects-PDU definition
    --

    Notification-Object-PDU ::=
      [XXX]                                        -- 15?
      SEQUENCE {
         request-id                        Unsigned32,
         notification-flags                NotificationSemantics,
         option-field                      OptionField,

         notifications                     NotificationList
      }

    NotificatonList ::=
      SEQUENCE (SIZE(0..max-bindings)) OF
        Notification

    Notification ::=
       SEQUENCE {
         notification-id                   OBJECT IDENTIFIER,
         system-uptime                     TimeTicks,
         notification-string               OCTET STRING,

         notification-objects              NotificationObjects
       }

    NotificationObjects ::=
      SEQUENCE (SIZE(0..max-bindings)) OF
        NotificationObject

    NotificationObject ::=
      SEQUENCE {
        option-field                       OptionField,
        request-base-OID                   OBJECT IDENTIFIER,
        notification-data                  DataList
      }

    NotificationSemantics ::=
       OCTET STRING (SIZE(1))
    --      errorOnUnknowFlag(0),
    --      requestAcknowledgment(1)

    --
    -- Acknowledgment-PDU definition
    --
```

```
    -- Note that this PDU used to Acknowledge any PDU with a
    -- requestAcknowledgement flag set.

    Acknowledgment-PDU ::=
      [XXX]                                        -- 16?
      SEQUENCE {
         request-id                        Unsigned32,
         ack-flags                         AcknowledgmentFlags,
         option-field                      OptionField
      }

    AcknowledgmentFlags ::=
       OCTET STRING (SIZE(1))
    --      errorOnUnknowFlag(0),
    --      requestAcknowledgment(1)

    END
```

3.2. New PDU component definitions

This section clarifies the data components contained within the PDUs defined above. See the "PDU Processing" section below for information on how processing of all of these elements together should be performed.

3.2.1. Common PDU elements of the OOPS PDUs

The following is a list of PDU fields which are common to multiple PDU sets (for example, they may exist in both the Get-Object-PDU and Write-Object-PDU operation).

request-id
: A numeric field indicating the current request number of a PDU. The response PDUs (Get-Object-Response-PDU, Write-Object-Response-PDU, Configuration-Objects-Response-PDU and the Acknowledgment-PDUs) MUST set this field to the value of the request-id from the request PDUs (Get-Object-PDU, Write-Object-PDU, Get-Configuration-Object-PDU or Notification-Object-PDU) that generated the response.

common flags
: A few common flags exist in multiple spots within all of the PDUs. These flags are:

errorOnUnknowFlag

This flag is used to indicates that for any unknown flags in the flag list, the Command Responder processing the request MUST immediately return an error without processing the request further. Otherwise, any unknown flags MUST be ignored and the request MUST be processed as if the flags were not set. However, even though processing must continue, an error MUST be inserted into the appropriate error-list, however, to indicate that the flag was ignored..

requestAcknowledgment

This flag indicates that the Command Responder should immediately return an Acknowledgment-PDU after the request has been received. The Acknowledgment-PDU serves as a quick reply to indicate that the request was received and is about to be processed. Requests which are expected to need time while processing are encouraged to set this flag so retransmissions or re-connections will be less necessary.

Note: This flag is not intended to replace reliable transport mechanisms but merely to serve as an application level acknowledgment. Since the acknowledgment could be lost in a non-reliable transport protocol, utilization of reliable transport (e.g., TCP) is highly encouraged.

*-option-field

A sequence reserved for future use. Command responders which encounter unknown option types specified in this sequence MUST ignore the data and proceed as if the field itself wasn't included in the request. Future protocol extensions MUST define extensions to be implemented within this sequence in such a way that the option can be safely ignored by implementations which fail to understand it.

This field exists in multiple places in the PDU definitions contained within this document. In all cases future extension documents may define extension fields for use within these option-field sequences. The sequences defined in the future MUST contain only an ASN.1 OPTION-ized list of parameters. The option index numbers for the sequence will be assigned and administered by IANA.

XXX Note: Proper ASN.1 definitions of the option fields in

this document are missing.

ErrorInformation

The ErrorInformation sequences contain a list of errors associated with processing a request. Multiple errors may be present in this list, and not all of the errors reported may be fatal errors. Some errors reported in the ErrorInformation list may be warnings indicating troubles found while processing a request. It consists of the following elements:

error-status

This is a numeric value indicating the type of error being conveyed. It's value dictates the type, severity, and the component for which the error-index will refer to.

error-index

An index into "something", where "something" is defined by what the error-status field contains (as an example, the unSupportedSearchRange error indicates it's an index into the search objects in the original request). It is an OBJECT IDENTIFIER which should begin with a 0.0 (zeroDotZero) prefix, and the components of the identifier after that will indicate which sub-element of either the index-request-list, column-request-list or search-criteria field caused the problem.

error-string

This SHOULD be set to a human-readable, administrative string describing the particular error (in greater detail than the error-status field alone can indicate). Software implementations MUST NOT expect the contents of field to be machine parsable. Standards documents MUST NOT dictate the format of the data to be conveyed by this field.

*-base-OID

A base-oid is the fundamental structure that a PDU will operate on. For SMIv2 structures, for instance, this will be either a scalar or a table. For SMIv3 objects, it should point to the highest instantiation of an aggregate object.

ElementSpecifier

An ElementSpecifier is used in many places within the MIB to indicate which element is being referred to. It is used when both sending and receiving data. It is always used in conjunction with a base-oid. All of the elements referenced by the ElementSpecifier is related to the base-oid in some fashion. The sub-elements of the ElementSpecifier are:

index-number

This is used to reference an index of the SMI structure referenced by the base-oid. Indexes are enumerated from the INDEX clause of a SMI structure, starting with the number 1. If an index is accessible directly within the base-oid structure, the element-number specifier SHOULD be used instead.

element-number

This is used to reference an accessible element of the SMI structure referenced by the base-oid. The value directly corresponds with the assigned OID element number underneath the base-oid. For SMIv2 tables, this field MUST contain the column number being referenced. For SMIv2 scalars, this field MUST contain zero.

subelement-specifier

This is used for referencing a single sub-element within an augmentation table or other SMIv3 structure. SMIv3 sub-element structure references MUST begin with the nullOID prefix (0.0).

For example, if the base-oid was referring to the IF-MIB::ifTable, then a subelement-specifier of IF-MIB::ifName would reference the ifName field of the ifXTable augmentation table.

multiple-subelements

When multiple sub-elements need to be referenced within a PDU construct, the multiple-subelements field can be used to list numerous related fields.

For example, if the base-oid was referring to the IF-MIB::ifTable, then a multiple-subelement specifier might have a element-specifier of IF-MIB::ifXTable and a element-list including both the ifXTable's ifName and

ifHCInOctets columns.

subelement-index
This is a method of specifying indexes of sub-elements and externally defined elements (E.G., augmentation tables). This field is only used in SearchCriteria.

DataList
XXX

3.2.2. Get-Object-PDU specific PDU elements

The Get-Object-PDU and Get-Object-Response-PDU specific protocol elements are briefly summarized here:

request-objects
Each request within the Get-Object-PDU is broken into individual request-objects and each of these are processed independently by the receiving Command Responder. The results of each processed request-object are then combined into corresponding return-objects within the Get-Object-Response-PDU.

max-return-objects
The maximum number of objects to return in the Get-Object-Response-PDU. A value of 0 indicates that all available objects should be returned.

skip-objects
If the cursor field contains a zero-length string, an Command Responder MUST NOT return the first skip-objects number of objects that match the search criteria. Instead, objects numbering from (skip-objects + 1) to (max-return-objects + skip-objects + 1) must be returned. A value of 0 indicates that the Command Responder should start returning object data beginning with the first object that matches the search-criteria. The skip-objects field MUST be entirely ignored if the cursor (see below) field contents is valid.

cursor
An OCTET STRING defining the starting point for collecting

data. This field MUST be a zero length OCTET STRING when accessing the first object in a structure or table. When request processing stops because the max-returned-objects count had been reached for a given request-object, Command Responders MUST return a value in this field that can be used in future requests to resume a search from the stopping point . The value of the cursor is implementation dependent. Command Generators MUST treat it as generic data and MUST NOT expect it to be in format which is parsable by the Command Generator. Command Generators SHOULD use it when continuing a search operation in a follow-up request. Command responders MUST refer to the value of the skip-objects field if the cursor field is a zero length string. Command responders MUST ignore the value of the skip-objects field if the cursor field is a non-zero length string and is deemed to be a valid cursor.

Cursors SHOULD be defined by the command responder in such a way as to avoid referencing an existing object. Cursors SHOULD be valid at any time in the future regardless of whether or not the underlying data within the Command Responder has changed. I.E., cursors should be data independent whenever possible and should be a reference into the appropriate place in the storage mechanism and not a reference to a data row within the storage mechanism. Command Responders MAY define cursors in such a way that they are not valid after an Command Responder reboots, but this is discouraged.

If a cursor is deemed invalid by the command responder, the command responder must do one of two things:

a) If the restartOnInvalidCursor request-flag is set, the search operation MUST NOT be performed and an appropriate invalidCursor error should be returned in the response.

b) If the restartOnInvalidCursor flag is not set, the search operation should start at the point indicated by the skip-objects field.

As an example cursor, an Command Responder which was performing internal Get-Object-PDU translations to internal or subagent based GETNEXT instrumentation might return a cursor containing the BER encoding of the last OID returned within the response. The next request could merely continue processing using the encoded OID just as if a real GETNEXT had

come in.

request-flags
: A flag list indicating particular features to be used when processing the Get-Object-PDU. The two Get-Object-PDU specific flags are:

restartOnInvalidCursor
: This flag indicates that when an invalid cursor is sent in the Get-Object-PDU request, the Command Responder should behave as described in the cursor field description above.

returnAllDataOnSearchFailure
: When a search operation fails due to an unsupported match-type, this flag indicates how to handle the error condition. If the returnAllDataOnSearchFailure bit is set, the search condition must be functionally ignored as if it did not restrict the data being returned. If not set, an error unSupportedSearchOperation condition must be inserted into the error-information-list and the request in particular is not processed.

request-base-OID
: The base OID for a given element to request information from. For SMIv2 objects, this is either the OID pointing to a table or a particular scalar. For SMIv3 objects, it should point to the highest instantiation of an aggregate object.

request-element-list
: This sequence contains a list of attributes to be returned in the response. IE, what particular data attributes or columns of a object or tabel's row should be returned. If this sequence is empty, then all data for the requested request-base-OID object MUST be returned. Command Generators SHOULD specify the data they wish to have returned rather than leave this field empty. Command Generators SHOULD request accessible index elements be returned as element values rather than indexes values.

search-criteria
: This sequence contains criteria which indicates which elements

should be returned. If the search-criteria is understood and if the object data to be returned does not match the criteria imposed by the restrictions in a request, then it SHOULD NOT be returned.

Command Generator implementations should specify attribute matching in preference to index matching, and thus index matching SHOULD ONLY be used for externally defined index values which don't have a attribute number assignment within the object.

match-type

Specifies the criteria to be imposed for a given search value. match-type's are simple logical operators used in search-criteria expressions.

If the returnAllDataOnSearchFailure bit is set, then if a search operation fails due to a condition like an unsupported or unknown match-type, all data MUST BE returned as if the search-type operator had always returned "true" (i.e., the Get-Object-Response should contain all possible data that the match-type would have been used to discriminate against). Note that other, properly-understood, components of the search-criteria can still limit the data returned to the requester.

If the returnAllDataOnSearchFailure bit is not set then the related return-objects field for the given request MUST NOT contain any data when sent back to the requester.

XXX: mention WOPs

In either case, an unSupportedSearchOperation error condition will be added to the error-information-list field of the Get-Object-Response-PDU and the error-index field will be set to the search component that caused the failure. As mentioned above, if the returnAllDataOnSearchFailure bit is set data is still returned to the requester and the error can be viewed as a warning that possibly more data was returned than was requested.

Implementations SHOULD return a unSupportedSearchOperation error in the error-information-list for each unsupported unique match-type within the requested search-criteria.

response-flags
The flags field MUST be set according to how the response was handled within the responder. Ideally this should be an exact copy of the flags field from the request, assuming the flags were understood. Flags from the request which were not understood by the command responder MUST be set to 0 in the returned Get-Object-Response-PDU.

3.2.3. Write-Object-PDU specific PDU elements

These protocol elements are specific to Write-Object-PDU requests.

write-flags

Global flags to apply to the entire Write-Object-PDU. The are broken down as follows:

returnDataOnlyOnError
If the returnDataOnlyOnError bit is set, then the Write-Object-Response-PDU will only contain a duplicate set-objects portion of the message if there was an error somewhere in the processing of the message, otherwise the objects list will be truncated. Command Responder implementations SHOULD support this flag, but if for any reason they decide to return the set-objects data portion of the message they MUST NOT set the returnDataOnlyOnError bit. If the flag is set, and the command responder supports it, then the write-transaction-responses field must be an zero length sequence.

write-transactions
Flags which apply to each sub-element of the write transaction to be performed. Currently there are 3 bits to define transactional processing semantics to be used when processing this message. These values are described in greater detail in the next section.

TransactionData
Indicates what type of operation is to be performed by this transaction set and contains the data associated with that operation. Specifically:

create-transaction, create-data-list
: Creates a new row within a row. It is an error condition if either the row can not be created (noCreation) or if the row already existed (createFailed). The create-data-list parameter specifies the data to be used to create the row with.

modify-transaction, modify-search-criteria, modify-data-list
: Modifies an existing row or set of rows. The rows that are to be affected by the modify-transaction are identified by the modify-search-criteria parameters. The data in the modify-data-list parameter specifies what values will be updated by the request. Note that a modify-transaction can not be used to create new rows.

delete-transaction, delete-search-criteria
: Deletes a given row or set of rows. If a row can not be deleted for some reason, it is considered to be an error condition (deleteFailed). The delete-search-criteria identifies which rows should be deleted from the table.

method-transaction, method-OID, method-arguments
: The method-transaction is similar to a function call, defined by the documentation associated with the method-OID, in that it expects arguments (method-arguments) and generates return values (method-return-data). It is up to future specifications to define method OIDs that can make use of this functionality.

sub-transaction
: A sub-transaction most likely contains a collection of transactions which are bound together by a different transaction-flags settings.

modify-search-criteria

delete-search-criteria
: The search criteria imposed by the modify-search-criteria and delete-search-criteria are identical in operation to the Get-Object-PDU equivalents, except that instead of requested information back the describe which objects should be modified. Unlike the Get-Object-PDU match-type handling, an unsupported match-type will always trigger an

unSupportedSearchOperation error condition and no modification or deletion will occur for that transaction (or any other transactions dependent on the success of transaction one containing the error).

create-response

A create-response object returns a copy of all the data sent in a create request. The response MUST contain an exact copy of the related request if the returnDataOnerror bit in the write-transactions flags is not set. If the returnDataOnerror bit in the write-transactions flag is set, this MUST be a encoded as an empty sequence.

modify-response

A modify response returns the data affected by the outgoing transaction. The modify-search-criteria field MUST be an exact copy from the associated Write-Object-PDU modify-transaction component. The modify-data-list should contain all the elements modified by the request. It MUST include all index objects for the object elements that were modified, plus the data values from the request.

delete-response

A delete response returns the data affected by the outgoing transaction. The modify-search-criteria field MUST be an exact copy from the associated Write-Object-PDU delete-transaction component. It MUST ONLY include the index objects for the object elements that were deleted.

3.2.4. Get-Configuration-Object-PDU specific PDU elements

A Get-Configuration-Object-PDU is used to request current configuration state for some portion of an Command Responder. The Configuration-Objects-Response-PDU will contain transactions suitable for use in a Write-Object-PDU request which can be issued to the Command Responder to restore itself to its current state. These elements of the Get-Configuration-Object-PDU are described here.

requested-config

A list of object identifiers for which configuration is desired. These object identifiers can be one of:

a) Defined OIDs from a particular MIB module which describes what configuration components should be returned in the Configuration-Objects-Response-PDU.

b) An OID of an object which contains configuration information. IE, this could point to a particular table from a MIB module.

c) The ZeroDotZero (0.0) OID which can be used to request all configuration information from an Command Responder.

Unsupported OBJECT IDENTIFIERS requested will be reported in the ErrorInformationList in the Configuration-Objects-Response-PDU as a getConfigObjectNotSupported error.

3.2.5. Notification-Object-PDU specific elements

The Notification-Object-PDU can be used to send SNMP Notifications to a notification receiver. The elements contained within a Notification-Object-PDU are described in this section.

notification-id
: This is the notification ID as defined by NOTIFICATION-TYPE and other SMI macros.

system-uptime
: The current value of sysUpTime.0 when the notification was sent from the notification generator.

notification-string
: A human readable string describing the notification. Notification Receivers MUST NOT expect this field to be machine parsable.

notification-objects
: Data objects included with the notification. Minimally, this must contain the OBJECTS field as defined by NOTIFICATION-TYPE SMI or other similar SMI macros.

4. PDU processing

4.1. Processing a Get-Object-PDU

The following describes the procedure for processing a Get-Object-PDU to produce a Get-Object-Response-PDU. The Get-Object-Response-PDU MUST contain the same request-id as the Get-Object-PDU. The Get-Object-Response-PDU MUST contain the same number of return-objects as there were request-objects in the Get-Object-PDU being processed. return-objects MUST appear in a order which matches the request-objects they are associated with from the Get-Object-PDU.

Apply the following procedure to each of the request-objects within the Get-Object-PDU to generate a corresponding return-objects sequence to be placed in the Get-Object-Response-PDU.

Each request-object MUST generate a corresponding return-object with less than or equal to max-return-objects pieces as specified in the request-object unless the max-return-objects field value is zero. If max-return-objects field is zero then all objects meeting the search-criteria evaluation MUST be returned. [XXX: transport and sizing issues]

If the cursor field is specified in the request-object, the cursor field MUST be used, if it is valid, to determine a starting point for the data to be returned. If the contents of the cursor field are invalid or it is impossible to determine a valid starting place and the restartOnInvalidCursor bit is set in the search-criteria, then the data returned should start with the skip-objects+1 object. Note that when counting objects to skip, only objects which meet the requirements imposed by the search-criteria WILL be counted. If the restartOnInvalidCursor bit is not set, then no objects should be returned in the Get-Object-Response-PDU return-objects list. Regardless of whether the restartOnInvalidCursor bit is set or not, an invalidCursor error MUST be inserted into the error-information-list in the return-objects field in the Get-Object-Response-PDU.

The cursor field in the Get-Object-Response-PDU MUST be a zero-length OCTET STRING if the Get-Object-PDU's max-return-objects is not 0 and there are less than max-return-objects being returned in the Get-Object-Response-PDU. If the number of objects being returned is equal to the max-return-objects parameter, then the cursor field MAY be a zero length string if the last object being returned is also the last object in the possible set of data for the given search-criteria.

Implementations MUST return data in a dependable order. Successive requests with identical search-criteria and request-base-OIDs MUST return objects in the same order in which they were returned previously. I.E., if object A (as defined by it's indexes) is returned before object B at one point in time, it MUST always be returned before object B for any future requests.

When the maximum number of objects to be returned has been reached, a cursor MUST be constructed which meets the criteria described above. It MUST be usable in the future to determine the starting place for future follow-on Get-Object-PDUs. The contents of the cursor field SHOULD [XXX: maybe this should be a MUST?] be data independent such that it is impossible for a cursor constructed by an Command Responder to be considered invalid at any point in the future. Since cursors are designed to alleviate processing overhead associated with restarting searches in the future, it is highly recommended that Command Responder implementations make efficient and appropriate use of the cursor field.

When searching for objects which are to be returned to the requester, the search-criteria MUST be consulted. Any object not matching the supported elements of the criteria imposed by the search-criteria field MUST NOT be included in the response. When processing the search-criteria field, any match-type which is not supported by the processing engine MUST result in an unSupportedSearchOperation error being inserted into the response's error-information list. Additionally, one of the following must be followed based on the value of the returnAllDataOnSearchFailure flag:

a) If the returnAllDataOnSearchFailure bit is set, then the evaluation MUST return all the data as if the unsupported match-type imposed no filtering on the data. Generally, this would mean that any match-type should be considered as evaluating to true, except in the case where the evaluation is enclosed directly within a logicalNOT operation, in which case the result of the logicalNOT must evaluate to true.

b) If the returnAllDataOnSearchFailure bit is unset, then the response MUST contain an empty returned-data-list.

Logical operations (logicalAND, logicalOR, logicalNOT) WILL use the sub-criteria field of the SearchCriteria to specify a list of sub-criteria the logical operation is to operate on. If the match type is logicalNOT there MUST be exactly one element in the sub-criteria list.

When constructing a return-objects list, the request-element-list must be consulted to determine which elements of a given object are to be returned in the response. When sending a Get-Object-PDU, the request-element-list SHOULD use the element-number specifier when possible in preference to the index-number specifier when the index is an accessible element in an object.

4.2. Processing a Write-Object-PDU.

When a Write-Object-PDU is received by an Command Responder it must process the transaction(s) found in the write-transaction field. The transaction-flags field of the write-transaction indicates how the sub-transactions grouped in a single write-transaction relate to each other.

The main Write-Object-PDU contains a single WriteTransaction which itself may be composed of multiple sub-transactions (create, modify, delete, method, or sub-transactions). The hierarchal nature of the transaction-data-list allows for more complex transaction relationships to be defined. Command Responders MAY implement nested sub-transactions but are not required to in order to conform with this specification.

The flags that make up the transaction-flags can be summarized as follows:

needSuccess
: If the needSuccess bit is set the containing transaction is considered sucessful if and only if no error conditions must occur during the processing of the sub-transactions in the transactions-data-list. If processing the sub-transactions results in a failure then the containing transaction is also considered to be a failure. If this bit is not set then the containing transaction itself will always be considered successful.

needAll
: If the needAll bit is set each sub-transaction in the transaction-data-list must be attempted for this transaction to be considered successful. If this is not set, processing of the transactions-data-list MUST stop after the first transaction in the transaction-data-list which is successful.

notOrderDependent
: If the notOrderDependent bit is set the transactions contained in the transaction-data-list MAY be executed in an order. If the needAll bit is set they may even be processed in parallel. Note that all transactions must still revert to their previous state if an error condition requires a state rollback for all the transactions.

Thus, the following combinations of the first two flags can be better described in combination:

needSuccess = true, needAll = true: doAll [doUntilFailure]
All contained transactions must succeed. If any failure occurs, the entire set of transactions in the transactions-data-list must be rolled back. This mode is most similar to the previous SNMP SET-PDU.

needSuccess = false, needAll = true: tryAll
With this combination set, the command responder must try to achieve all the transactions, however if any transaction fails it is not required to roll back the rest of the transactions. The failed transactions themselves, however, must individually properly revert to the previous state. This mode allows a bunch of independent transactions to be specified in one management operation. It is roughly equivalent to a bunch of individual and independent SNMP SET-PDU operations.

needSuccess = true, needAll = false: doAtLeastOne [doUntilSuccess]
In this case, at least one must succeed for the transaction as a whole to succeed. More importantly, if any one transaction component does succeed, processing MUST be stopped and the transaction as a whole is considered successful. Another way to put it: at most one successful component is executed and never more than one. This mode is useful for specifying, for example, a transaction with one or more "fall-back" transactions in case one fails.

needSuccess = false, needAll = false: tryAtLeastOne
In this case, at least one must succeed for the transaction as a whole to succeed. More importantly, if any one transaction component does succeed, processing MUST be stopped. The different between this combination of bits and the previous is that this transaction itself always succeeds, even if all the sub-transactions fail. It is impossible for a transaction of this type to be considered a failure in itself.

Also, error reporting will always occur on failed objects even if they don't affect the containing transactions and surrounding transactions.

Note that more data can be returned in many cases when search operations required operations on multiple rows based on the search criteria.

XXX: response generation.

4.3. Processing a Get-Configuration-Object-PDU request

A Get-Configuration-Object-PDU will contain a sequence of OBJECT IDENTIFIERs indicating which particular configuration objects are being requested. The types of OBJECT IDENTIFIERs which may be specified in the list were previously defined.

When an Command Responder receives a Get-Configuration-Object-PDU it should return in the Configuration-Objects-Response-PDU a valid set of WriteTransactions which will return a device to the current configuration state at the time the request was received. Exactly one WriteTransaction MUST be returned for each OBJECT IDENTIFIER requested in the request-objects field.

4.4. Generating a Notification-Object-PDU

A Notification-Object-PDU is generated by first creating an appropriate request-id, filling in the appropriate bits of the notification-flags and specifying any options in the option-field. The notifications field is then filled in with the notifications to be sent in this PDU. Note that the Notification-Object-PDU allows for multiple notifications to be sent within a single PDU. This paves the way for future delivery of multiple notifications at once. Predicted applications capability for this included combined delivery of multiple notifications as well as automated Command Responder data sampling at regular intervals.

For each notification to be delivered, the following procedure is then used to create each notification with the notifications field: The notification-id field is filled in with the appropriate OBJECT IDENTIFIER as required by the specification defining the notification to be sent (E.G., a NOTIFICATION-TYPE macro for SMIv2 MIB modules). The system-uptime MUST contain the current value of the sysUpTime object for the Notification Generator sending the notification. The notification-string field should be filled in with a human readable string if possible, otherwise a zero-length OCTET STRING should be used instead.

The notification-objects field MUST then be populated with the objects required by the specification defining the notification (E.G., the OBJECTS field of the NOTIFICATION-TYPE macro for SMIv2 MIB modules). Notification-objects will need to make use of the subelement-specifier field of an ElementSpecifier when sending objects from a related table. See the "Examples" section below for an example of this. The notification-objects field MUST be populated in the same order as required by the defining specification. Any optional objects the Notification Generator wishes to append to the notification-objects field may then be appended to the generated notification-objects field.

5. Examples

Here are some example requests and responses of data retrieval. In each case, the curly braces indicate a sequence of some kind within BER.

5.1. Retrieve a specific row from the ifTable

This example retrieves the ifDescr and ifType columns for the 5th interface from the ifTable

The Request:

```
   Get-Object-PDU {
     request-id              1
     option-field            { }
     request-objects      { {
       max-return-objects    0          -- get all
       skip-objects          0          -- starting with the first
       cursor                ""         -- 0 length for 1st request
       flags                 0x00       -- none specified.
       option-field          {}

       request-base-object   IF-MIB::ifTable

       request-element-list  { element-number = 2,
                               element-number = 3 }

       search-criteria       { match-type = 0,  -- equals
                               match-data = {
                                   which = 1,   -- ifIndex
                                   what  = 5    -- 5th
                               } }
    } }
  }
```

The Response:

```
   Get-Object-Response-PDU {
     request-id              1
     option-field            {}
     response-objects ::= { {
       error-information-list     {}   -- no Errors
       cursor                     ""   -- no other data
       response-flags             0x00 -- none specified.
       option-field               {}   -- no options

       request-base-object        IF-MIB::ifTable
```

```
          returned-data-list { {
            { which-element = 2,
              element-value = "interface 5" },
            { which-element = 3,
              element-value = 6 }
          } }
        } }
      }
```

5.2. A multiple-packet example with a double filter

This example shows the retrieval of the elements of the ifTable which are of type "ethernetCsmacd" and have a ifSpeed >= 10,000,000 bps. Retrieve only 1 row at a time, and include columns from the ifXTable augmentation table. Note that the response has the objects returned in the order the remote Command Responder specified [interface 12, then interface 5], not in something that would directly corresponds to the older-style lexicographical ordering. The order returned must be consistent from one request to the next (i.e., interface 12 will always be returned before interface 5 in any future requests at any point in time where they are both to be returned), but the manner in which the ordering is accomplished is implementation specific. The cursor is used to indicate to the base Command Responder where the request should restart from.
The first request:

```
   Get-Object-PDU {
     request-id              2
     option-field            { }
     request-objects      { {
       max-return-objects    1        -- only return 1
       skip-objects          0        -- starting with the first
       cursor                ""       -- 0 length for first request
       flags                 0x00     -- none
       option-field          {}

       request-base-object   IF-MIB::ifTable

       -- get ifIndex, ifDescr, and ifType and
       -- ifHCInOctets, ifHCOutOctets from the ifXTable
       request-element-list  { element-number = 1,
                               element-number = 2,
                               element-number = 3,
                               multiple-subelement = {
                                 element-specifier = ifXTable,
                                 element-list = {
                                    6, 10
```

```
                                    }
                                  }
                                }

          search-criteria       { match-type = 100,   -- AND
                                  sub-criteria = {
                                   {
                                     match-type = 0,  -- equals
                                     match-data = {
                                        which = 3,    -- ifType
                                        what  = 6     -- ethernetCsmacd
                                     }
                                   },
                                   {
                                     -- greaterThanOrEqual
                                     match-type = 12,
                                     match-data = {
                                        which = 5,    -- ifSpeed
                                        what  = 10000000
                                     }
                                   }
                                  } }
       } }
     }
```

The first response:

```
   Get-Object-Response-PDU {
     request-id             2
     option-field           {}
     response-objects ::= { {
       error-information-list     {}             -- no Errors
       cursor                     "interface 12"
       response-flags             0x00           -- none
       option-field               {}

       request-base-object        IF-MIB::ifTable

       returned-data-list { {
         { which-element = 1,
           element-value = 12 },
         { which-element = 2,
           element-value = "interface 12" },
         { which-element = 3,
           element-value = 6 }
         { which-element = ifXTable
           element-value = {
              { which-element = 6,
```

```
                  element-value = 12345 },
                { which-element = 10,
                  element-value = 12345 },
              }
           }
         } }

       } }
     }
```

The second request to obtain the more data:

```
   Get-Object-PDU {
     request-id             3
     option-field           { }
     request-objects      { {
       max-return-objects   1             -- only return 1
       skip-objects         0             -- functionally not used
       cursor               "interface 12" -- continue
       flags                0x00          -- none
       option-field         {}

       request-base-object  IF-MIB::ifTable

       -- get ifIndex, ifDescr, and ifType and
       -- ifHCInOctets, ifHCOutOctets from the ifXTable
       request-element-list  { element-number = 1,
                               element-number = 2,
                               element-number = 3,
                               multiple-subelement = {
                                 element-specifier = ifXTable,
                                 element-list = {
                                    6, 10
                                 }
                               }
                             }

       search-criteria       { match-type = 100,   -- AND
                               sub-criteria = {
                                {
                                  match-type = 0,  -- equals
                                  match-data = {
                                     which = 3,    -- ifType
                                     what  = 6     -- ethernetCsmacd
                                  }
                                },
                                {
                                  -- greaterThanOrEqual
```

```
                                    match-type = 12,
                                    match-data = {
                                       which = 5,     -- ifSpeed
                                       what  = 10000000
                                    }
                                  }
                                 } }
      } }
    }
```

The second response. The zero length cursor means there is no further information and no further requests are needed.

```
   Get-Object-Response-PDU {
     request-id              3
     option-field            {}
     response-objects ::= { {
       error-information-list     {}            -- no Errors
       cursor                     ""            -- has last object
       response-flags             0x00          -- none
       option-field               {}

       request-base-object        IF-MIB::ifTable

       returned-data-list { {
         { which-element = 1,
           element-value = 5 },
         { which-element = 2,
           element-value = "interface 5" },
         { which-element = 3,
           element-value = 6 }
         { which-element = ifXTable
           element-value = {
              { which-element = 6,
                element-value = 23456 },
              { which-element = 10,
                element-value = 23456 },
            }
          }
       } }

     } }
   }
```

5.3. A Write-Object-PDU example

The following is an example of a Write-Object-PDU request which will set the ifAdminStatus flag to "down" for all interfaces in the ifTable of type 'basicISDN'. Additionally, an acknowledgment is

requested to verify that an Command Responder has started received the request.

```
   Write-Object-PDU {
     request-id              10
     write-flags             0x40          -- requestAcknowledgment
     option-field            { }

     write-transactions      {
       transaction-flags     0x00          -- none specified
       option-field          { }

       transaction-data-list {
         modify-transaction  {
           request-base-OID          IF-MIB::ifTable
           modify-search-criteria    { match-type = 0,  -- equals
                                       match-data = {
                                         which = 3,     -- ifType
                                         what  = 20     -- basicISDN
                                       }
                                     }
           modify-data-list          { {
                                       -- ifAdminStatus
                                       which-element 7
                                       element-value 2  -- down
                                     } }
         }
       }
     } }
   }
```

First, the Command Responder will send an acknowledgment indicating the reception of the Write-Object-PDU. This is sent because the requestAcknowledgment flag was set in the write-flags field.

```
   Acknowledgment-PDU {
     request-id              10
     ack-flags               0x00
     option-field            {}
   }
```

The Write-Object-Response-PDU will indicate which interfaces were actually affected by the Write-Object-PDU (in this example the 4th and 2nd interfaces):

```
   Write-Object-Response-PDU {
     request-id              10
     write-flags             0x00               -- none
     option-field            {}                 -- no options
     error-information-list  {}                 -- no Errors
     write-transaction-results ::= { {
```

```
        transaction-flags          0x00          -- none
        option-field               {}            -- no options
        transaction-response-list ::= {
          modify-transaction ::= {
            request-base-OID          IF-MIB::ifTable
            modify-data-list ::= {
              { which-element/index-number    1      -- ifIndex
                element-value                 4   }  -- 4th interface
              { which-element/index-number    1      -- ifIndex
                element-value                 2   }  -- 2nd interface
            }
          }
        }
      } }
    }
```

5.4. A Get-Configuration-Object-PDU example

The following shows how a Command Generator might request the current configuration of the SNMP-TARGET-MIB::snmpTargetAddrTable, and an example response from a Command Responder.

```
  Get-Configuration-Objects-PDU {
     request-id                 30
     gcop-flags                 0x00
     option-field               { }
     request-objects            { {
       request-flags            0x00
       option-field             { }
       request-objects          SNMP-TARGET-MIB::snmpTargetAddrTable
     } }
  }
```

The following shows a possible response for the above request.

```
  Configuration-Objects-Response-PDU {
     request-id                   30
     gcop-flags                   0x00
     option-field                 { }
     config-results { {
         request-flags            0x00
         option-field             { }
         error-information        { }
         config-transactions {
             transaction-flags    0x60 -- needSuccess, needAll
             option-field         { }
             transaction-data-list {
               -- deletes all existing rows
               delete-transaction {
                 request-base-OID   snmpTargetAddrTable
```

```
                  delete-search-criteria { }  -- all data!
                 }
                 -- creates a new row
                 create-transaction {
                   request-base-OID   snmpTargetAddrTable
                   create-data-list {
                       -- snmpTargetAddrName
                       { which-element/index-number   1
                         element-value                "localhost" }
                       -- snmpTargetAddrTDomain
                       { which-element/element-number 2
                         element-value                snmpUDPDomain }
                       -- snmpTargetAddrTAddress
                       { which-element/element-number 3
                         element-value                0x7f00000100a2 }
                       -- snmpTargetAddrTimeout
                       { which-element/element-number 4
                         element-value                5 }
                       -- snmpTargetAddrRetryCount
                       { which-element/element-number 5
                         element-value                5 }
                       -- snmpTargetAddrTagList
                       { which-element/element-number 6
                         element-value                "mytag" }
                       -- snmpTargetAddrParams
                       { which-element/element-number 7
                         element-value                "myparam" }
                       -- snmpTargetAddrStorageType
                       { which-element/element-number 8
                         element-value                3 }
                       -- snmpTargetAddrRowStatus
                       { which-element/element-number 9
                         element-value                4 }
                    }
                 }
               }
             }
          } }
      }
```

5.5. A Notification-Object-PDU example: IF-MIB::linkUp

The following shows a standard linkUp notification sent using the new Notification-Object-PDU.

```
   Notification-Object-PDU {
     request-id              40
     notification-flags      0x40  -- requestAcknowledgment
     option-field            {}
```

```
        notifications ::= { {
          notification-id       IF-MIB::linkUP
          system-uptime         100
          notification-string   "Interface 12 enabled at the console"
          notification-objects  ::= {
             { which-element = IF-MIB::ifTable
               element-value = {
                 { which-element = 1,     -- ifIndex
                   element-value = 12 },  -- #12
                 { which-element = 7,     -- ifAdminStatus
                   element-value = 1 },   -- up
                 { which-element = 8,     -- ifOperStatus
                   element-value = 1 }    -- up
               }
             }
          }
        } }
      }
```

The notification receiver would then respond with the following acknowledgment:

```
   Acknowledgment-PDU {
     request-id              40
     ack-flags               0x00
     option-field            {}
   }
```

6. References

6.1. Normative References

[1] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", RFC3416, December 2002.

[2] Blumenthal, U. and B. Wijnen, "The User-Based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[3] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

[4] Levi, D., Meyer, P. and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

6.2. Informative References

[5] Woodcock, D., "Operator Requirements of Infrastructure Management Methods", Internet Draft draft-ops-operator-req-mgmt-02.txt, expired August 2002.

[6] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", Internet Draft draft-iab-nm-workshop-00.txt, expires April 2003.

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

7. Intellectual Property

8. Security Considerations

XXX: discuss VACM

The protocol defined in this document by itself does not provide a secure environment. Even if the network itself is secure (for example by using IPSec), there is no control as to who on the secure network is allowed access to management information.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model STD 62, RFC 3414 [2] and the View-based Access Control Model STD 62, RFC 3415 [3] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity is properly configured so that:

- only those principals (users) having legitimate rights can access or modify the values of any MIB objects supported by that entity;

- the occurrence of particular events on the entity will be communicated appropriately;

- the entity responds appropriately and with due credence to events and information that have been communicated to it.

9. IANA Considerations

IANA will need to administer assignment rights for the following field identifiers within the PDUs defined in this document:

option-field OPTIONAL tags
The BER encoded OPTIONAL-field tags for the various *-option-field PDU components will need to be assigned and administered by IANA.

XXX Some more XXX.

10. Acknowledgements

Many people participated in various discussions about the PDUs defined in this document. Special thanks go to:
Michael Baer (Network Associates Laboratories)
Robert Moore (IBM)
David Perkins (Riverstone Networks and SNMPinfo)
Randy Presuhn (BMC Software)
Dave Shield (University of Liverpool)
Steve Waldbusser (Next Beacon)
Glenn Waters (Nortel Networks)

11. Editor's Addresses

Wes Hardaker
P.O. Box 382
Davis, California 95617
USA
Email: hardaker@tislabs.com

12. Full Copyright Statement