1. Status of this Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

To learn the current status of any Internet-Draft, please check the "Iid-abstracts.txt" listing contained in the internet-drafts Shadow Directories on:

    ftp.is.co.za (Africa)
    nic.nordu.net (Europe)
    ds.internic.net (US East Coast)
    ftp.isi.edu (US West Coast)
    munnari.oz.au (Pacific Rim)
2. Introduction

This memo is informational. It specifies a clarification of the definition and use of the Opaque type defined in Simple Network Management Protocol (SNMP) Structure of Management Information (SMI). There are two versions of the SMI. The first, called SMIv1, is defined by RFCs 1155[1], 1212[2], and 1215[3]. The second, called SMIv2, is defined by RFCs 1902[4], 1903[5], and 1904[6]. This memo shows that the Opaque type is well defined, and after domestication it is an effective and low-cost solution to:

1) support 64-bit counters in SMIv1 and SNMPv1;
2) support future types added to the SMI without changing the SNMP protocol; and
3) support for a discriminated union type.

All of the solutions are accomplished without a change to the technical content of the specifications for the SNMPv1[7] and SNMPv2[8][9] protocols.

This memo specifies a clarification for both version 1 and version 2 of the SNMP SMI, which is a standard for the Internet community.

3. Background

The SMI defines the SNMP MIB module language, which is an augmented subset of the ASN.1 specification language[10]. The SNMP protocol is defined with a proper sub-set of ASN.1 notations and SNMP protocol messages are encoded using a proper sub-set of the Basic Encoding Rules (BER) for ASN.1[11].

The Opaque data type is defined in first version of the SMI for SNMP. The ASN.1 notation is found in section 6 of SMIv1[1] and is shown below:

```
Opaque ::= [APPLICATION 4] IMPLICIT OCTET
```

The Opaque data type is described in section 3.2.3.6 and is shown below:

```
This application-wide type supports the capability to pass arbitrary ASN.1 syntax. A value is encoded using the ASN.1 basic rules into a string of octets. This, in turn, is encoded as an OCTET STRING, in effect "double-wrapping" the original ASN.1 value.

Note that a conforming implementation need only be able to accept and recognize opaquely-encoded data. It need not be able to unwrap the data and then interpret its contents.

Further note that by use of the ASN.1 EXTERNAL type, encodings other than ASN.1 may be used in opaquely-encoded data.
```
Unfortunately, the last sentence in this description is not correct. This inaccuracy is fixed in the SMIv2[4], and the description for the Opaque data type from section 7.1.9 is shown below:

The Opaque type supports the capability to pass arbitrary ASN.1 syntax. A value is encoded using the ASN.1 Basic Encoding Rules into a string of octets. This, in turn, is encoded as an OCTET STRING, in effect "double-wrapping" the original ASN.1 value.

Note that a conforming implementation need only be able to accept and recognize opaquely-encoded data. It need not be able to unwrap the data and then interpret its contents.

The description was made correct by eliminating the last sentence, which was incorrect and provided no additional useful information.

The SNMPv2 WG added a policy statement in RFC 1902 restricting usage of the Opaque type. This restriction was due to several factors including:
1) misunderstandings caused by the description of the Opaque type in SMIv1;
2) incorrect "interpretations" spread by a few "SNMP experts"; and
3) no perceived mechanisms to describe the contents (or value) of an Opaque type when used.

This policy is specified in the following text from section 7.1.9 of the SMIv2:

The Opaque type is provided solely for backward-compatibility, and shall not be used for newly-defined object types.

A requirement of "standard" MIB modules is that no object may have a SYNTAX clause value of Opaque.

The intent of this memo is to clarify the meaning of the Opaque data type; show how it uniquely (and at a low cost) solves several important problems; and replace the policy from RFC 1902 restricting usage with the policy specified in this memo.

The harnessing of the Opaque type for practical use is called the "domestication of the Opaque type."
4. A Description of the Opaque Type

The Opaque type is defined in ASN.1 notation as the following:

```
Opaque ::= [APPLICATION 4] IMPLICIT OCTET
```

This ASN.1 definition means that when values of the Opaque data type are used in SNMP messages, which are encoded using the basic encoding rules of ASN.1 (BER)[11], the result is the following:

```
Serialization of a value, which is an SNMP Opaque data type
-----------------------------------------------
| tag | length | value is serialized                               |
-----------------------------------------------
|     |        | ---------------------------------------------------|
| V-tag | V-length | V-value |
-----------------------------------------------
```

where:
- tag is one octet with value of ‘44’h.
- length is one or more octets, but is typically one octet for a length value less than 128, two octets for a length value less than 256, and three octets for a length value less than 65536.
- value is a string of octets that is the BER serialization of a valid value of an ASN.1 type
- V-tag, V-length, and V-value are the serialization of an ASN.1 value using BER serialization

A common misinterpretation of the definition of the Opaque type is that values for it can be any string of octet values. This is incorrect, since the SMI clearly specifies that the value must be the BER serialization of a value for an ASN.1 type.

The following table contains examples of BER serialization. The first column contains ASN.1 data type specifications. The second column contains a valid value of the ASN.1 data type specified in the same row. The third column contains the BER serialization of the value specified in the same row. This column also specifies valid values of the Opaque data type. The fourth column contains the BER serialization of the value of data type Opaque in the same row.
These examples clearly demonstrate that the valid values of data type Opaque and their BER serialization are well defined. Examining the result of the BER serialization reveals that the original value is not changed. Serialization just adds an additional tag and length "around" the previously serialized value. BER serialization is called "wrapping a value." The values specified in column two (in the above table) are wrapped once in column three, and wrapped again (or "double wrapped") in column four.

5. The Domestication of the Opaque Data Type

There are two problems with the current definition of the Opaque type. First, there are no restrictions on the ASN.1 data types or values that can be "wrapped." Thus, values and ASN.1 data types, even those not allowed in SNMP, may be serialized. Secondly, usage of the Opaque data type does not require the ASN.1 data type of the double wrapped values to be specified. Thus, it is difficult, if not impossible, to "unwrap" a serialized value.

On the other hand, the Opaque data type does provide the lowest cost solution to two critical problems in SNMP. The first problem is how to support the addition of new basic data types to the SMI and protocol. (One example is 64-bit counters in SMI1 and SNMPv1.) The second problem is how to support a "union" data type that is needed in sophisticated environments such as mid-level managers.

The taming of the current (and "wild") definition of the Opaque data type for practical uses is called the "domestication of Opaque."

The domestication is easily accomplished with a simple harnessing of the definition of the Opaque data type. The replacement definition of the Opaque data type for SMIv1 and SMIv2 is:
The Opaque data type supports the capability to pass arbitrary ASN.1 syntax. A value is encoded using the ASN.1 Basic Encoding Rules into a string of octets. This, in turn, is encoded as an OCTET STRING, in effect "double-wrapping" the original ASN.1 value.

Note that a conforming implementation of the SNMP protocol shall be able to encode and decode SNMP PDUs with the value portion of a variable-bind pair using this type.

A conforming SNMP MIB module shall specify the ASN.1 type for the original values in the DESCRIPTION clause of the OBJECT-TYPE or TEXTUAL-CONVENTION constructs where the Opaque data type is used.

Furthermore, standards track MIB modules are restricted in their use of the ASN.1 data types wrapped by the Opaque data type. Only the ASN.1 types defined in the SMI for use as the value for the SYNTAX clause for columnar and scalar objects, the SEQUENCE data type, the CHOICE data type, and specially created APPLICATION data types may be used. Note, that these may be also be qualified with an "IMPLICIT" context-specific tag. However, context-specific tags greater than or equal to 32 are reserved for special situations, and cannot be used.

5.1. Support for New Types

SMIv2 added a new type, Counter64, not found in SMIv1. This type was added to address the need for event and flow counts in situations where a 32-bit counter rolls over too rapidly (such as in a networking device using high-speed transmission technology including FDDI and ATM). Unfortunately, object types that are defined with syntax of Counter64 cannot be converted to a MIB module in the SMIv1 format and cannot be accessed using the SNMPv1 protocol, since the type Counter64 is not defined in the SMIv1 or in the SNMPv1 protocol. However, instead of using the Counter64 type directly, the Opaque type can be used to hold the serialization of the Counter64 type, or any other new type that needs to be added to future SMI versions.

MIB modules written to use this approach must use a textual convention instead of the new type for the data type of object types, which is specified in the SYNTAX clause. Such a textual convention is an example domestication of the untamed Opaque data type. The value of the Opaque data type for the textual convention is restricted to a serialized value of a tagged version of the new data type. This approach allows only those SNMP managers or agents who need a new data type to be required to be upgraded. This approach requires no change, and does not impact existing SNMP compliant agents or managers.

The domestication of the Opaque data type reserves ASN.1 context-specific tags greater than or equal to 32 for special use. One use of
the reserved values is to use ASN.1 context-specific tags with values 48 and above for support of new SMI types in old versions of the SNMP protocol. This is done by adding the value of the tag for a "new" type to the base value 48 and using the resulting sum as the context-specific tag for the ASN.1 type.

For example, the tag for data type Counter64 is application-specific 6, which is ’46’h in BER. The sum of 48 (’30’h) and ’46’h is 118 (’76’h). Thus, the ASN.1 definition for this context-type is "[118] IMPLICIT Counter64," with the tag encoded in BER as ’9F76’h.

Note that BER encodes tags as three fields. These are class(cls), primitive/constructed flag(f), and number. If the number is less than 31, then all three fields are encoded into one octet. If the number is 31 or greater, then multiple octets are used to encode the tag. The format is shown below for one and two octet tags:

<table>
<thead>
<tr>
<th>One octet tag</th>
<th>Two octet tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>[cls</td>
<td>f] 0 - 30</td>
</tr>
</tbody>
</table>

where:

cls is 00 - universal
01 - application
10 - context specific
11 - private use

f is 0 - primitive
1 - constructed

5.1.1. 64-Bit Counters

To support 64-bit counters in SMIv1 and SMIv2 MIB modules, and use the SNMPv1 and SNMPv2 protocols, the following textual convention must be used in SYNTAX clause instead of the data type Counter64:

```
C64 TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
"A 64-bit counter which monotonically increases until it reaches a maximum value of (2^64)-1 (18446744073709551615 decimal), when it wraps around and starts increasing again from zero.

Counters have no defined 'initial' value, and thus, a single value of a counter has (in general) no information content. Discontinuities in the monotonically increasing value normally occur"
```
at re-initialization of the management system, and at other times as specified in the description of an object-type using this textual convention. If such other times can occur, for example, the creation of an object instance at times other than re-initialization, then a corresponding object should be defined with a SYNTAX clause value of TimeStamp (a well-known textual convention) indicating the time of the last discontinuity.

The value of the MAX-ACCESS clause for objects with a SYNTAX clause of this textual convention must be either ‘read-only’ or ‘accessible-for-notify’.

A DEFVAL clause is not allowed for objects using this textual convention.

The value is restricted to the BER serialization of the following ASN.1 type:

```plaintext
COUNTER64 ::= [118] IMPLICIT Counter64
```

(note: the value 118 is the sum of ’30’h and ’46’h)

The BER serialization of the length for values of this type must use the definite length, short encoding form.

For example, the BER serialization of value 56782 of type COUNTER64 is ‘9f760300ddce’h. (The tag is ‘9f76’h; the length is ’03’h; and the value is ’00ddce’h.) The BER serialization of value ’9f760300ddce’h of data type Opaque is ‘44069f760300ddce’h. (The tag is ‘44’h; the length is ’06’h; and the value is ’9f760300ddce’h.)"
5.1.2. Future New Data Types

In the future, there may be a need for a limited number of additional data types to support the usage of SNMP in management of networks other than those for computer data, such as heating and cooling systems, and automotive traffic control. Also, distributed management of computer data networks with so-called mid-level managers may require addition of new data types. The domestication of the opaque data type allows new data types to be added without disrupting existing systems and tools used to create them. The following text describes the process and requirements to add a new data type.

The addition of new data types is serious business and may not proceed without careful review of the network management area. A new data type may not be defined to associate semantics with an existing data type. (Note that this requirement would not have allowed the Counter or Gauge data types to be created as basic data types. Instead, they would have been textual conventions of an unsigned integer data type.) A new data type may be an ASN.1 universal type or an application-specific type. For each new data type defined, a textual convention must also be defined to wrap the new data type in an Opaque data type. The following example shows the definition of two new data types. The first is an application-specific data type and the second is a universal data type.

-- define a new data type using the next available application
-- specific tag (note: using BER, the tag for the type is ‘48’h)
New1Type ::= [APPLICATION 8] IMPLICIT OCTET STRING (SIZE(4))

-- define a textual convention to wrap the new data type
New1 TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
"A new data type with some characteristics specified. The value is restricted to the BER serialization of the following ASN.1 data type:
   NEW1TYPE ::= [120] IMPLICIT New1Type
   (note: the value 120 is the sum of ‘30’h and ‘48’h)"
The BER serialization of the length for values of this data type must use the definite length, short encoding form.

For example, the BER serialization of value '12345678'h of data type NEW1TYPE is '9f780412345678'h. (The tag is '9f78'h; the length is '04'h; and the value is '12345678'h.) The BER serialization of value '9f780412345678'h of data type Opaque is '44079f780412345678'h. (The tag is '44'h; the length is '07'h; and the value is '9f780412345678'h.)"  
SYNTAX Opaque (SIZE(7))

or

-- define a new data type based on an existing ASN.1 universal data type (note: using BER, the tag for the data type is '03'h)
New2Type ::= BIT STRING

-- define a textual convention to wrap the new data type
New2 TEXTUAL-CONVENTION
STATUS current
DESCRIPTION "A new data type with some characteristics specified.

The value is restricted to the BER serialization of the following ASN.1 data type:

NEW2TYPE ::= [51] IMPLICIT New2Type
(note: the value 51 is the sum of '30'h and '03'h)
The BER serialization of the length for values of this data type must use the definite length, short encoding form.

For example, the BER serialization of value '12345678'h of type NEW2TYPE is '9f33050012345678'h. (The tag is '9f33'h; the length is '05'h; and the value is '0012345678'h.) The BER serialization of value '9f33050012345678'h of type data type Opaque is '44089f33050012345678'h. (The tag is '44'h; the length is '08'h; and the value is '9f33050012345678'h.)"
SYNTAX Opaque (SIZE(4..65535))

5.2. Support for Unions

There is a need for a union data type that allows a value to be identified and that allows different value encodings based on the identification. This need was present when the first version of the SMI and the core IETF SNMP MIB were created. A union was needed to hold different types of network addresses. The solution that was created, the data type NetworkAddress, proved problematic and was not included in
the second version of the SMI. However, the need for a union of network addresses still exists. Other needs also exist for unions. For example, researchers have created mid-level managers that allow running of scripts to compute values, and have the resulting value retrievable via SNMP. The data type of a computed value may be any of the data types allowed by the SMI such as integers, strings, and object identifiers. Without a union data type, however, a mid-level manager must define several objects, each with a different data type of a potential result, and also define an object that specifies which of the objects actually contains the result. The development, maintenance, and operational costs of this approach are quite high. Fortunately, these needs and others are easily satisfied with a low-cost domestication of the Opaque data type.

5.2.1. Definition of SnmpUnion

The domestication of the Opaque data type reserves ASN.1 context-specific tags greater than or equal to 32 for special use. The ASN.1 context-specific tag with value of 47 is used to define the SNMP union. The domestication of opaque requires that the ASN.1 type definition be specified for the wrapped value. The definitions of the ASN.1 type for the SNMP union and the textual convention to wrap values as an Opaque data type follow:

```plaintext
-- A discriminated union
SnmpUnionType ::= [47] IMPLICIT SEQUENCE {
    memberId INTEGER (-2147483648.. 2147483647),
    memberType CHOICE {
        -- the first six data types are currently defined
        -- in the SNMP SMI
        int32Val INTEGER (-2147483648.. 2147483647),
        stringVal OCTET STRING (SIZE(0..65535)),
        oidVal OBJECT IDENTIFIER,
        noneVal NULL,
        uint32Val [APPLICATION 2] IMPLICIT INTEGER (0..4294967295),
        opaqueVal [APPLICATION 4] IMPLICIT OCTET STRING (SIZE(2..65535)),
        -- the last four data types are additional special
        -- APPLICATION types
        floatVal -- the "single format" as defined in
            -- Binary Floating Point[12][13]
            [APPLICATION 8] IMPLICIT OCTET STRING (SIZE(4)),
        doubleVal -- the "double format" as defined in
            -- Binary Floating Point[12][13]
            [APPLICATION 9] IMPLICIT OCTET STRING (SIZE(8)),
        int64Val [APPLICATION 10] IMPLICIT INTEGER (-9223372036854775808..
            9223372036854775807),
    }
}
```
uint64Val [APPLICATION 11] IMPLICIT
   INTEGER (0..18446744073709551615) }

-- The textual convention to wrap the SNMP union as an
-- Opaque data type
SnmpUnion TEXTUAL-CONVENTION
   STATUS current
   DESCRIPTION "A discriminated union, which is used to identify
one member from a choice of members. Each member
represents one kind of value. The objects or
textual conventions that specify this textual
convention in their SYNTAX clause shall specify
in their DESCRIPTION clause a list containing
the following information:
   1) discriminator value - identifies a member
   2) data type for member - one of int32, int64, string,
      oid, none, uint32, uint64, opaque, float, or double
   3) description for member - any semantics associated
      with the kind of value
Updates to objects (and textual conventions) using
this textual convention may add new members,
but may never remove or change the semantics of
previously defined members.

The value is restricted to the BER serialization of
the ASN.1 type SnmpUnionType.
The BER serialization of values of data type SnmpUnion
must 1) use primitive encoding, 2) use definite
encoding of the lengths, and 3) use the shortest
possible encoding of the lengths.

For example, the BER serialization of value
{ 1, int32Val 34 } of type SnmpUnionType is
'Bf2f06020101020122'h. (The tag is 'Bf2f'h; the length
is '06'h; and the value is '020101020122'h. The value
consists of items '020101'h and '020122'h. The first item
has tag '02'h; length '01'h; and value '01'h. The second
item has tag '02'h; length '01'h; and value '22'h.)
The BER serialization of value 'Bf2f06020101020122'h
of type Opaque is '4409Bf2f06020101020122'h."
   SYNTAX     Opaque (SIZE(7..65535))

5.2.2. Example Uses of SnmpUnion

With the SnmpUnion textual convention, objects can be defined in both
SMIv1 and SMIv2 formats and can be accessed via both SNMPv1 and SNMPv2
protocols. Shown below are definitions for the same object in both SMI
formats:
-- in SMIV1
exmplUnion OBJECT-TYPE
SYNTAX     SnmpUnion
ACCESS      read-write
STATUS      mandatory
DESCRIPTION
"Example object with syntax of union, available in both SNMPv1 and SNMPv2."
 ::= { exmpls 2 }

-- in SMIV2
exmplUnion OBJECT-TYPE
SYNTAX     SnmpUnion
MAX-ACCESS read-write
STATUS      current
DESCRIPTION
"Example object with syntax of union, available in both SNMPv1 and SNMPv2."
 ::= { exmpls 2 }

The following example shows usage of a union to define a textual convention for all the transport address types found in "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", [RFC 1906][9]:

Taddr TEXTUAL-CONVENTION
STATUS      current
DESCRIPTION
"A transport address. The address can be from any of the following protocol families: Internet UDP, Internet TCP, OSI CLNS, OSI CONS, AppleTalk DDP, and Novel IPX."

<table>
<thead>
<tr>
<th>ID</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
<td>no transport address</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>UDP - in network byte order, octets 1..4: IP address; octets 5..6: UDP port</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>TCP - in network byte order, octets 1..4: IP address; octets 5..6: TCP port</td>
</tr>
<tr>
<td>4</td>
<td>string</td>
<td>CLNS - octet 1: length of NSAP (an unsigned integer ‘n’ with value of either 0 or from 3 to 20); octets 2..(n+1): NSAP (in concrete binary representation); octets (n+2)..m: TSEL (a value of (up to 64) octets)</td>
</tr>
</tbody>
</table>
5  string  CONS - same format as CLNS addresses
6  string  DDP - a NBP name
   octet 1: value, ‘n’, is length of object;
   octets 2..(n+1): object (a value of
      (up to 32) octets);
   octet n+2: value, ‘p’, is length of type;
   octets (n+3)..(n+2+p): type (a value of
      (up to 32) octets);
   octet n+3+p: value, ‘q’, is length of zone;
   octets (n+4+p)..(n+3+p+q): zone (a value
      of (up to 32) octets).
   For comparison purposes, fields object,
   value, and zone are case-insensitive. All
   of these fields may contain any octet value
   other than 255 (hex ff).
7  string  IPX - in network byte order
   octets 1..4: network-number;
   octets 5..10: physical-address;
   octets 11..12: socket-number."

SYNTAX      SnmpUnion

The following example shows usage of a union to define a textual
convention for the value that results from running a script at a mid-
level manager.

ScriptResult TEXTUAL-CONVENTION
STATUS      current
DESCRIPTION
 "The result from running a script

   ID  Syntax  Description
   1  none    The result is not available yet
   2  uint32  Error running the script, the values are:
      1: syntax problem in script
      2: no response from script target
      3: invalid response from script target
      3: out of resources
   3  int32   Signed integer result
   4  int64   Big signed integer result
   5  string  String result
   6  oid     Object identifier result
   7  uint32  Unsigned integer result
   8  uint64  Big unsigned integer result
   9  float   Float result
   10 double  Double result"

SYNTAX      SnmpUnion
5.2.3. Example BER for SnmpUnion Values

Below is shown an object definition and a table containing the BER encoding of values for the object. This table illustrates the encoding of each kind of syntax allowed for a union member.

```plaintext
exmplUnionObj TEXTUAL-CONVENTION
  SYNTAX SnmpUnion
  MAX-ACCESS read-write
  STATUS current
  DESCRIPTION "An example object that shows each type of syntax allowed for members of a union:

<table>
<thead>
<tr>
<th>ID</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int32</td>
<td>Signed integer</td>
</tr>
<tr>
<td>2</td>
<td>int64</td>
<td>Big signed integer</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>String result</td>
</tr>
<tr>
<td>4</td>
<td>oid</td>
<td>Object identifier result</td>
</tr>
<tr>
<td>5</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>uint32</td>
<td>Unsigned integer result</td>
</tr>
<tr>
<td>7</td>
<td>uint64</td>
<td>Big unsigned integer result</td>
</tr>
<tr>
<td>8</td>
<td>opaque</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>float</td>
<td>Float result</td>
</tr>
<tr>
<td>10</td>
<td>double</td>
<td>Double result</td>
</tr>
</tbody>
</table>

::= { exmpl 3 }
```

<table>
<thead>
<tr>
<th>Member</th>
<th>Example</th>
<th>BER Serialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Syntax</td>
<td>Value</td>
</tr>
<tr>
<td>1</td>
<td>int32</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>int64</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>string</td>
<td>&quot;01&quot;</td>
</tr>
<tr>
<td>4</td>
<td>oid</td>
<td>1.3.6</td>
</tr>
<tr>
<td>5</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>uint32</td>
<td>56782</td>
</tr>
</tbody>
</table>

Expires 12/07/97
6. Suggestions for Further Work

The SnmpUnion textual convention is a powerful addition to the usage of SNMP. However, there is currently no direct support for it in the SMI. Thus, the identification, syntax, and description of the members of a union can only be specified in the DESCRIPTION clause for the object or textual convention where it is used. Inside a DESCRIPTION clause, there is no enforcement of proper specification. A MIB compiler cannot reliably parse the content of the DESCRIPTION clause and make that information available to its users, such as application programs. For these reasons, it is suggested that a construct be added to a future version of the SMI to specify the characteristics of members of a union. The resulting addition should be well defined so that it is parsable with a MIB compiler.

7. Acknowledgments

Thanks go to Sandy M. Perkins for editorial assistance and review.
8. References


