Network Working Group
Request for Comments: 1903
Obsoletes: 1443
Category: Standards Track

SNMPv2 Working Group
J. Case
SNMP Research, Inc.
K. McCloghrie
Cisco Systems, Inc.
M. Rose
Dover Beach Consulting, Inc.
S. Waldbusser
International Network Services

January 1996

Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)

Status of this Memo

Table of Contents

1. Introduction

A management system contains: several (potentially many) nodes, each with a processing entity, termed an agent, which has access to management instrumentation; at least one management station; and, a management protocol, used to convey management information between the agents and management stations. Operations of the protocol are carried out under an administrative framework which defines

authentication, authorization, access control, and privacy policies.

Management stations execute management applications which monitor and control managed elements. Managed elements are devices such as hosts, routers, terminal servers, etc., which are monitored and controlled via access to their management information.

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules. These modules are written using a subset of OSI's Abstract Syntax Notation One (ASN.1) [1], termed the Structure of Management Information (SMI) [2].

When designing a MIB module, it is often useful to define new types similar to those defined in the SMI. In comparison to a type defined in the SMI, each of these new types has a different name, a similar syntax, but a more precise semantics. These newly defined types are termed textual conventions, and are used for the convenience of humans reading the MIB module. It is the purpose of this document to define the initial set of textual conventions available to all MIB modules.

Objects defined using a textual convention are always encoded by means of the rules that define their primitive type. However, textual conventions often have special semantics associated with them. As such, an ASN.1 macro, TEXTUAL-CONVENTION, is used to concisely convey the syntax and semantics of a textual convention.

For all textual conventions defined in an information module, the name shall be unique and mnemonic, and shall not exceed 64 characters in length. (However, names longer than 32 characters are not recommended.) All names used for the textual conventions defined in all "standard" information modules shall be unique.

1.1. A Note on Terminology

For the purpose of exposition, the original Internet-standard Network Management Framework, as described in RFCs 1155 (STD 16), 1157 (STD 15), and 1212 (STD 16), is termed the SNMP version 1 framework (SNMPv1). The current framework is termed the SNMP version 2 framework (SNMPv2).

2. Definitions

```
SNMPv2-TC DEFINITIONS ::= BEGIN

IMPORTS
    ObjectSyntax, TimeTicks
        FROM SNMPv2-SMI;


-- definition of textual conventions

TEXTUAL-CONVENTION MACRO ::=
BEGIN
    TYPE NOTATION ::=
                  DisplayPart
                  "STATUS" Status
                  "DESCRIPTION" Text
                  ReferPart
                  "SYNTAX" Syntax

    VALUE NOTATION ::=
                  value(VALUE Syntax)

    DisplayPart ::=
                  "DISPLAY-HINT" Text
                | empty

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

    ReferPart ::=
                  "REFERENCE" Text
                | empty

    -- uses the NVT ASCII character set
    Text ::= """" string """"

    Syntax ::=
                  type(ObjectSyntax)
                | "BITS" "{" Kibbles "}"
    Kibbles ::=
                  Kibble
                | Kibbles "," Kibble
    Kibble ::=
                  identifier "(" nonNegativeNumber ")"
```

```
END


DisplayString ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "255a"
    STATUS       current
    DESCRIPTION
            "Represents textual information taken from the NVT ASCII
            character set, as defined in pages 4, 10-11 of RFC 854.

            To summarize RFC 854, the NVT ASCII repertoire specifies:

              - the use of character codes 0-127 (decimal)

              - the graphics characters (32-126) are interpreted as
                US ASCII

              - NUL, LF, CR, BEL, BS, HT, VT and FF have the special
                meanings specified in RFC 854

              - the other 25 codes have no standard interpretation

              - the sequence 'CR LF' means newline

              - the sequence 'CR NUL' means carriage-return

              - an 'LF' not preceded by a 'CR' means moving to the
                same column on the next line.

              - the sequence 'CR x' for any x other than LF or NUL is
                illegal.  (Note that this also means that a string may
                end with either 'CR LF' or 'CR NUL', but not with CR.)

            Any object defined using this syntax may not exceed 255
            characters in length."
    SYNTAX       OCTET STRING (SIZE (0..255))


PhysAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1x:"
    STATUS       current
    DESCRIPTION
            "Represents media- or physical-level addresses."
    SYNTAX       OCTET STRING


MacAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1x:"
```

```
    STATUS       current
    DESCRIPTION
            "Represents an 802 MAC address represented in the
            'canonical' order defined by IEEE 802.1a, i.e., as if it
            were transmitted least significant bit first, even though
            802.5 (in contrast to other 802.x protocols) requires MAC
            addresses to be transmitted most significant bit first."
    SYNTAX       OCTET STRING (SIZE (6))


TruthValue ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "Represents a boolean value."
    SYNTAX       INTEGER { true(1), false(2) }

TestAndIncr ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "Represents integer-valued information used for atomic
            operations.  When the management protocol is used to specify
            that an object instance having this syntax is to be
            modified, the new value supplied via the management protocol
            must precisely match the value presently held by the
            instance.  If not, the management protocol set operation
            fails with an error of 'inconsistentValue'.  Otherwise, if
            the current value is the maximum value of 2^31-1 (2147483647
            decimal), then the value held by the instance is wrapped to
            zero; otherwise, the value held by the instance is
            incremented by one.  (Note that regardless of whether the
            management protocol set operation succeeds, the variable-
            binding in the request and response PDUs are identical.)

            The value of the ACCESS clause for objects having this
            syntax is either 'read-write' or 'read-create'.  When an
            instance of a columnar object having this syntax is created,
            any value may be supplied via the management protocol.

            When the network management portion of the system is re-
            initialized, the value of every object instance having this
            syntax must either be incremented from its value prior to
            the re-initialization, or (if the value prior to the re-
            initialization is unknown) be set to a pseudo-randomly
            generated value."
    SYNTAX       INTEGER (0..2147483647)


AutonomousType ::= TEXTUAL-CONVENTION
```

```
    STATUS       current
    DESCRIPTION
            "Represents an independently extensible type identification
            value.  It may, for example, indicate a particular sub-tree
            with further MIB definitions, or define a particular type of
            protocol or hardware."
    SYNTAX       OBJECT IDENTIFIER


InstancePointer ::= TEXTUAL-CONVENTION
    STATUS       obsolete
    DESCRIPTION
            "A pointer to either a specific instance of a MIB object or
            a conceptual row of a MIB table in the managed device.  In
            the latter case, by convention, it is the name of the
            particular instance of the first accessible columnar object
            in the conceptual row.

            The two uses of this textual convention are replaced by
            VariablePointer and RowPointer, respectively."
    SYNTAX       OBJECT IDENTIFIER


VariablePointer ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "A pointer to a specific object instance.  For example,
            sysContact.0 or ifInOctets.3."
    SYNTAX       OBJECT IDENTIFIER


RowPointer ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "Represents a pointer to a conceptual row.  The value is the
            name of the instance of the first accessible columnar object
            in the conceptual row.

            For example, ifIndex.3 would point to the 3rd row in the
            ifTable (note that if ifIndex were not-accessible, then
            ifDescr.3 would be used instead)."
    SYNTAX       OBJECT IDENTIFIER


RowStatus ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "The RowStatus textual convention is used to manage the
```

creation and deletion of conceptual rows, and is used as the value of the SYNTAX clause for the status column of a conceptual row (as described in Section 7.7.1 of [2].)

The status column has six defined values:

- 'active', which indicates that the conceptual row is available for use by the managed device;

- 'notInService', which indicates that the conceptual row exists in the agent, but is unavailable for use by the managed device (see NOTE below);

- 'notReady', which indicates that the conceptual row exists in the agent, but is missing information necessary in order to be available for use by the managed device;

- 'createAndGo', which is supplied by a management station wishing to create a new instance of a conceptual row and to have its status automatically set to active, making it available for use by the managed device;

- 'createAndWait', which is supplied by a management station wishing to create a new instance of a conceptual row (but not make it available for use by the managed device); and,

- 'destroy', which is supplied by a management station wishing to delete all of the instances associated with an existing conceptual row.

Whereas five of the six values (all except 'notReady') may be specified in a management protocol set operation, only three values will be returned in response to a management protocol retrieval operation: 'notReady', 'notInService' or 'active'. That is, when queried, an existing conceptual row has only three states: it is either available for use by the managed device (the status column has value 'active'); it is not available for use by the managed device, though the agent has sufficient information to make it so (the status column has value 'notInService'); or, it is not available for use by the managed device, and an attempt to make it so would fail because the agent has insufficient information (the state column has value 'notReady').

NOTE WELL

> This textual convention may be used for a MIB table, irrespective of whether the values of that table's conceptual rows are able to be modified while it is active, or whether its conceptual rows must be taken out of service in order to be modified. That is, it is the responsibility of the DESCRIPTION clause of the status column to specify whether the status column must not be 'active' in order for the value of some other column of the same conceptual row to be modified. If such a specification is made, affected columns may be changed by an SNMP set PDU if the RowStatus would not be equal to 'active' either immediately before or after processing the PDU. In other words, if the PDU also contained a varbind that would change the RowStatus value, the column in question may be changed if the RowStatus was not equal to 'active' as the PDU was received, or if the varbind sets the status to a value other than 'active'.

Also note that whenever any elements of a row exist, the RowStatus column must also exist.

To summarize the effect of having a conceptual row with a status column having a SYNTAX clause value of RowStatus, consider the following state diagram:

	STATE			
ACTION	A status column does not exist	B status col. is notReady	C status column is notInService	D status column is active
set status column to createAndGo	noError or inconsistent-Value ->D	inconsist-entValue	inconsistent-Value	inconsistent-Value
set status column to createAndWait	noError or wrongValue see 1	inconsist-entValue	inconsistent-Value	inconsistent-Value
set status column to active	inconsistent-Value	inconsist-entValue	noError	noError
		or		
		see 2 ->D	->D	->D
set status column to notInService	inconsistent-Value	inconsist-entValue	noError	noError ->C
		or		or
		see 3 ->C	->C	wrongValue
set status column to destroy	noError ->A	noError ->A	noError ->A	noError ->A
set any other column to some value	see 4	noError see 1	noError ->C	see 5 ->D

(1) goto B or C, depending on information available to the agent.

(2) if other variable bindings included in the same PDU, provide values for all columns which are missing but required, then return noError and goto D.

(3) if other variable bindings included in the same PDU, provide values for all columns which are missing but required, then return noError and goto C.

(4) at the discretion of the agent, the return value may be either:

inconsistentName: because the agent does not choose to create such an instance when the corresponding RowStatus instance does not exist, or

inconsistentValue: if the supplied value is inconsistent with the state of some other MIB object's value, or

noError: because the agent chooses to create the instance.

If noError is returned, then the instance of the status column must also be created, and the new state is B or C, depending on the information available to the agent. If inconsistentName or inconsistentValue is returned, the row remains in state A.

(5) depending on the MIB definition for the column/table, either noError or inconsistentValue may be returned.

NOTE: Other processing of the set request may result in a response other than noError being returned, e.g., wrongValue, noCreation, etc.

Conceptual Row Creation

There are four potential interactions when creating a conceptual row: selecting an instance-identifier which is not in use; creating the conceptual row; initializing any objects for which the agent does not supply a default; and, making the conceptual row available for use by the managed device.

Interaction 1: Selecting an Instance-Identifier

The algorithm used to select an instance-identifier varies for each conceptual row. In some cases, the instance-identifier is semantically significant, e.g., the destination address of a route, and a management station selects the instance-identifier according to the semantics.

In other cases, the instance-identifier is used solely to distinguish conceptual rows, and a management station without specific knowledge of the conceptual row might examine the instances present in order to determine an unused instance-identifier. (This approach may be used, but it is often highly sub-optimal; however, it is also a questionable practice for a naive management station to attempt conceptual row creation.)

Alternately, the MIB module which defines the conceptual row might provide one or more objects which provide assistance in determining an unused instance-identifier. For example, if the conceptual row is indexed by an integer-value, then an object having an integer-valued SYNTAX clause might be defined for such a purpose, allowing a management station to issue a management protocol retrieval operation. In order to avoid unnecessary collisions between competing management stations, 'adjacent' retrievals of this object should be different.

Finally, the management station could select a pseudo-random number to use as the index. In the event that this index was already in use and an inconsistentValue was returned in response to the management protocol set operation, the management station should simply select a new pseudo-random number and retry the operation.

A MIB designer should choose between the two latter algorithms based on the size of the table (and therefore the efficiency of each algorithm). For tables in which a large number of entries are expected, it is recommended that a MIB object be defined that returns an acceptable index for creation. For tables with small numbers of entries, it is recommended that the latter pseudo-random index mechanism be used.

Interaction 2: Creating the Conceptual Row

Once an unused instance-identifier has been selected, the management station determines if it wishes to create and activate the conceptual row in one transaction or in a negotiated set of interactions.

Interaction 2a: Creating and Activating the Conceptual Row

The management station must first determine the column requirements, i.e., it must determine those columns for which it must or must not provide values. Depending on the complexity of the table and the management station's knowledge of the agent's capabilities, this determination can be made locally by the management station. Alternately, the management station issues a management protocol get operation to examine all columns in the conceptual row that it wishes to create. In response, for each column, there are three possible outcomes:

- a value is returned, indicating that some other management station has already created this conceptual row. We return to interaction 1.

- the exception 'noSuchInstance' is returned, indicating that the agent implements the object-type associated with this column, and that this column in at least one conceptual row would be accessible in the MIB view used by the retrieval were it to exist. For those columns to which the agent provides read-create access, the 'noSuchInstance' exception tells the management station that it should supply a value for this column when the conceptual row is to be created.

- the exception 'noSuchObject' is returned, indicating that the agent does not implement the object-type associated with this column or that there is no conceptual row for which this column would be accessible in the MIB view used by the retrieval. As such, the management station can not issue any management protocol set operations to create an instance of this column.

Once the column requirements have been determined, a management protocol set operation is accordingly issued. This operation also sets the new instance of the status column to 'createAndGo'.

When the agent processes the set operation, it verifies that it has sufficient information to make the conceptual row available for use by the managed device. The information available to the agent is provided by two sources: the management protocol set operation which creates the conceptual row, and, implementation-specific defaults supplied by the agent (note that an agent must provide implementation-specific defaults for at least those objects which it implements as read-only). If there is sufficient information available, then the conceptual row is created, a 'noError' response is returned, the status column is set to 'active', and no further interactions are necessary (i.e., interactions 3 and 4 are skipped). If there is insufficient information, then the conceptual row is not created, and the set operation fails with an error of 'inconsistentValue'. On this error, the management station can issue a management protocol retrieval operation to determine if this was because it failed to specify a value for a required column, or, because the selected instance of the status column already existed. In the latter case, we return to interaction 1. In the former case, the management station can re-issue the set operation with the additional information, or begin interaction 2 again using 'createAndWait' in order to negotiate creation of the conceptual row.

NOTE WELL

> Regardless of the method used to determine the column requirements, it is possible that the management station might deem a column necessary when, in fact, the agent will not allow that particular columnar instance to be created or written. In this case, the management protocol set operation will fail with an error such as 'noCreation' or 'notWritable'. In this case, the management station decides whether it needs to be able to set a value for that particular columnar instance. If not, the management station re-issues the management protocol set operation, but without setting a value for that particular columnar instance; otherwise, the management station aborts the row creation algorithm.

Interaction 2b: Negotiating the Creation of the Conceptual Row

The management station issues a management protocol set operation which sets the desired instance of the status

column to 'createAndWait'. If the agent is unwilling to process a request of this sort, the set operation fails with an error of 'wrongValue'. (As a consequence, such an agent must be prepared to accept a single management protocol set operation, i.e., interaction 2a above, containing all of the columns indicated by its column requirements.) Otherwise, the conceptual row is created, a 'noError' response is returned, and the status column is immediately set to either 'notInService' or 'notReady', depending on whether it has sufficient information to make the conceptual row available for use by the managed device. If there is sufficient information available, then the status column is set to 'notInService'; otherwise, if there is insufficient information, then the status column is set to 'notReady'. Regardless, we proceed to interaction 3.

Interaction 3: Initializing non-defaulted Objects

The management station must now determine the column requirements. It issues a management protocol get operation to examine all columns in the created conceptual row. In the response, for each column, there are three possible outcomes:

- a value is returned, indicating that the agent implements the object-type associated with this column and had sufficient information to provide a value. For those columns to which the agent provides read-create access (and for which the agent allows their values to be changed after their creation), a value return tells the management station that it may issue additional management protocol set operations, if it desires, in order to change the value associated with this column.

- the exception 'noSuchInstance' is returned, indicating that the agent implements the object-type associated with this column, and that this column in at least one conceptual row would be accessible in the MIB view used by the retrieval were it to exist. However, the agent does not have sufficient information to provide a value, and until a value is provided, the conceptual row may not be made available for use by the managed device. For those columns to which the agent provides read-create access, the 'noSuchInstance' exception tells the management station that it must issue additional management protocol set operations, in order to provide a value associated with this column.

- the exception 'noSuchObject' is returned, indicating that the agent does not implement the object-type associated with this column or that there is no conceptual row for which this column would be accessible in the MIB view used by the retrieval. As such, the management station can not issue any management protocol set operations to create an instance of this column.

If the value associated with the status column is 'notReady', then the management station must first deal with all 'noSuchInstance' columns, if any. Having done so, the value of the status column becomes 'notInService', and we proceed to interaction 4.

Interaction 4: Making the Conceptual Row Available

Once the management station is satisfied with the values associated with the columns of the conceptual row, it issues a management protocol set operation to set the status column to 'active'. If the agent has sufficient information to make the conceptual row available for use by the managed device, the management protocol set operation succeeds (a 'noError' response is returned). Otherwise, the management protocol set operation fails with an error of 'inconsistentValue'.

NOTE WELL

A conceptual row having a status column with value 'notInService' or 'notReady' is unavailable to the managed device. As such, it is possible for the managed device to create its own instances during the time between the management protocol set operation which sets the status column to 'createAndWait' and the management protocol set operation which sets the status column to 'active'. In this case, when the management protocol set operation is issued to set the status column to 'active', the values held in the agent supersede those used by the managed device.

If the management station is prevented from setting the status column to 'active' (e.g., due to management station or network failure) the conceptual row will be left in the 'notInService' or 'notReady' state, consuming resources indefinitely. The agent must detect conceptual rows that have been in either state for an abnormally long period of

time and remove them. It is the responsibility of the DESCRIPTION clause of the status column to indicate what an abnormally long period of time would be. This period of time should be long enough to allow for human response time (including 'think time') between the creation of the conceptual row and the setting of the status to 'active'. In the absense of such information in the DESCRIPTION clause, it is suggested that this period be approximately 5 minutes in length. This removal action applies not only to newly-created rows, but also to previously active rows which are set to, and left in, the notInService state for a prolonged period exceeding that which is considered normal for such a conceptual row.

Conceptual Row Suspension

When a conceptual row is 'active', the management station may issue a management protocol set operation which sets the instance of the status column to 'notInService'. If the agent is unwilling to do so, the set operation fails with an error of 'wrongValue'. Otherwise, the conceptual row is taken out of service, and a 'noError' response is returned. It is the responsibility of the DESCRIPTION clause of the status column to indicate under what circumstances the status column should be taken out of service (e.g., in order for the value of some other column of the same conceptual row to be modified).

Conceptual Row Deletion

For deletion of conceptual rows, a management protocol set operation is issued which sets the instance of the status column to 'destroy'. This request may be made regardless of the current value of the status column (e.g., it is possible to delete conceptual rows which are either 'notReady', 'notInService' or 'active'.) If the operation succeeds, then all instances associated with the conceptual row are immediately removed."

```
    SYNTAX       INTEGER {
                     -- the following two values are states:
                     -- these values may be read or written
                     active(1),
                     notInService(2),
```

```
                    -- the following value is a state:
                    -- this value may be read, but not written
                    notReady(3),

                    -- the following three values are
                    -- actions: these values may be written,
                    --   but are never read
                    createAndGo(4),
                    createAndWait(5),
                    destroy(6)
                }


TimeStamp ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "The value of the sysUpTime object at which a specific
            occurrence happened.  The specific occurrence must be
            defined in the description of any object defined using this
            type."
    SYNTAX       TimeTicks


TimeInterval ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "A period of time, measured in units of 0.01 seconds."
    SYNTAX       INTEGER (0..2147483647)


DateAndTime ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "2d-1d-1d,1d:1d:1d.1d,1a1d:1d"
    STATUS       current
    DESCRIPTION
            "A date-time specification.

            field  octets  contents                  range
            -----  ------  --------                  -----
              1      1-2   year                      0..65536
              2       3    month                     1..12
              3       4    day                       1..31
              4       5    hour                      0..23
              5       6    minutes                   0..59
              6       7    seconds                   0..60
                           (use 60 for leap-second)
              7       8    deci-seconds              0..9
              8       9    direction from UTC        '+' / '-'
              9      10    hours from UTC            0..11
```

```
             10      11    minutes from UTC          0..59

            For example, Tuesday May 26, 1992 at 1:30:15 PM EDT would be
            displayed as:

                                 1992-5-26,13:30:15.0,-4:0

            Note that if only local time is known, then timezone
            information (fields 8-10) is not present."
    SYNTAX       OCTET STRING (SIZE (8 | 11))


StorageType ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "Describes the memory realization of a conceptual row.  A
            row which is volatile(2) is lost upon reboot.  A row which
            is either nonVolatile(3), permanent(4) or readOnly(5), is
            backed up by stable storage.  A row which is permanent(4)
            can be changed but not deleted.  A row which is readOnly(5)
            cannot be changed nor deleted.

            If the value of an object with this syntax is either
            permanent(4) or readOnly(5), it cannot be modified.
            Conversely, if the value is either other(1), volatile(2) or
            nonVolatile(3), it cannot be modified to be permanent(4) or
            readOnly(5).

            Every usage of this textual convention is required to
            specify the columnar objects which a permanent(4) row must
            at a minimum allow to be writable."
    SYNTAX       INTEGER {
                     other(1),       -- eh?
                     volatile(2),    -- e.g., in RAM
                     nonVolatile(3), -- e.g., in NVRAM
                     permanent(4),   -- e.g., partially in ROM
                     readOnly(5)     -- e.g., completely in ROM
                 }


TDomain ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
          "Denotes a kind of transport service.

          Some possible values, such as snmpUDPDomain, are defined in
          'Transport Mappings for Version 2 of the Simple Network
          Management Protocol (SNMPv2)'."
```

```
    SYNTAX       OBJECT IDENTIFIER


TAddress ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
          "Denotes a transport service address.

          For snmpUDPDomain, a TAddress is 6 octets long, the initial 4
          octets containing the IP-address in network-byte order and the
          last 2 containing the UDP port in network-byte order.  Consult
          'Transport Mappings for Version 2 of the Simple Network
          Management Protocol (SNMPv2)' for further information on
          snmpUDPDomain."
    SYNTAX       OCTET STRING (SIZE (1..255))


END
```

3. Mapping of the TEXTUAL-CONVENTION macro

The TEXTUAL-CONVENTION macro is used to convey the syntax and semantics associated with a textual convention. It should be noted that the expansion of the TEXTUAL-CONVENTION macro is something which conceptually happens during implementation and not during run-time.

For all descriptors appearing in an information module, the descriptor shall be unique and mnemonic, and shall not exceed 64 characters in length. (However, descriptors longer than 32 characters are not recommended.) Further, the hyphen is not allowed as a character in the name of any textual convention.

3.1. Mapping of the DISPLAY-HINT clause

The DISPLAY-HINT clause, which need not be present, gives a hint as to how the value of an instance of an object with the syntax defined using this textual convention might be displayed. The DISPLAY-HINT clause may be present if and only if the syntax has an underlying primitive type of INTEGER or OCTET STRING. (Note, however, that the semantics defined for a particular syntax can cause the use of DISPLAY-HINT for that syntax to make no sense, e.g., for Counter32 [2].)

When the syntax has an underlying primitive type of INTEGER, the hint consists of an integer-format specification, containing two parts. The first part is a single character suggesting a display format, either: 'x' for hexadecimal, or 'd' for decimal, or 'o' for octal, or 'b' for binary. The second part is always omitted for 'x', 'o' and

'b', and need not be present for 'd'. If present, the second part starts with a hyphen and is followed by a decimal number, which defines the implied decimal point when rendering the value. For example:

```
Hundredths ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "d-2"
    ...
    SYNTAX     INTEGER (0..10000)
```

suggests that a Hundredths value of 1234 be rendered as "12.34"

When the syntax has an underlying primitive type of OCTET STRING, the hint consists of one or more octet-format specifications. Each specification consists of five parts, with each part using and removing zero or more of the next octets from the value and producing the next zero or more characters to be displayed. The octets within the value are processed in order of significance, most significant first.

The five parts of a octet-format specification are:

(1) the (optional) repeat indicator; if present, this part is a '*', and indicates that the current octet of the value is to be used as the repeat count. The repeat count is an unsigned integer (which may be zero) which specifies how many times the remainder of this octet-format specification should be successively applied. If the repeat indicator is not present, the repeat count is one.

(2) the octet length: one or more decimal digits specifying the number of octets of the value to be used and formatted by this octet-specification. Note that the octet length can be zero. If less than this number of octets remain in the value, then the lesser number of octets are used.

(3) the display format, either: 'x' for hexadecimal, 'd' for decimal, 'o' for octal, or 'a' for ascii. If the octet length part is greater than one, and the display format part refers to a numeric format, then network-byte ordering (big-endian encoding) is used interpreting the octets in the value.

(4) the (optional) display separator character; if present, this part is a single character which is produced for display after each application of this octet-specification; however, this character is not produced for display if it would be immediately followed by the display of the repeat terminator character for this octet-specification. This character can be any character other than a decimal digit and a '*'.

(5) the (optional) repeat terminator character, which can be present only if the display separator character is present and this octet-specification begins with a repeat indicator; if present, this part is a single character which is produced after all the zero or more repeated applications (as given by the repeat count) of this octet-specification. This character can be any character other than a decimal digit and a '*'.

Output of a display separator character or a repeat terminator character is suppressed if it would occur as the last character of the display.

If the octets of the value are exhausted before all the octet-format specification have been used, then the excess specifications are ignored. If additional octets remain in the value after interpreting all the octet-format specifications, then the last octet-format specification is re-interpreted to process the additional octets, until no octets remain in the value.

3.2. Mapping of the STATUS clause

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

The values "current", and "obsolete" are self-explanatory. The "deprecated" value indicates that the definition is obsolete, but that an implementor may wish to support the use of this textual convention to foster interoperability with older implementations.

3.3. Mapping of the DESCRIPTION clause

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

Note that, in order to conform to the ASN.1 syntax, the entire value of this clause must be enclosed in double quotation marks, and therefore cannot itself contain double quotation marks, although the value may be multi-line.

3.4. Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a textual cross-reference to a related item defined in some other published work.

3.5. Mapping of the SYNTAX clause

The SYNTAX clause, which must be present, defines abstract data structure corresponding to the textual convention. The data structure must be one of the alternatives defined in the ObjectSyntax CHOICE or the BITS construct (see section 7.1 in [2]).

4. Security Considerations

Security issues are not discussed in this memo.

5. Editor's Address

Keith McCloghrie
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
US

Phone: +1 408 526 5260
EMail: kzm@cisco.com

6. Acknowledgements

This document is the result of significant work by the four major contributors:

Jeffrey D. Case (SNMP Research, case@snmp.com)
Keith McCloghrie (Cisco Systems, kzm@cisco.com)
Marshall T. Rose (Dover Beach Consulting, mrose@dbc.mtview.ca.us)
Steven Waldbusser (International Network Services, stevew@uni.ins.com)

In addition, the contributions of the SNMPv2 Working Group are acknowledged. In particular, a special thanks is extended for the contributions of:

Alexander I. Alten (Novell)
Dave Arneson (Cabletron)
Uri Blumenthal (IBM)
Doug Book (Chipcom)
Kim Curran (Bell-Northern Research)
Jim Galvin (Trusted Information Systems)
Maria Greene (Ascom Timeplex)
Iain Hanson (Digital)
Dave Harrington (Cabletron)
Nguyen Hien (IBM)
Jeff Johnson (Cisco Systems)
Michael Kornegay (Object Quest)

Deirdre Kostick (AT&T Bell Labs)
David Levi (SNMP Research)
Daniel Mahoney (Cabletron)
Bob Natale (ACE*COMM)
Brian O'Keefe (Hewlett Packard)
Andrew Pearson (SNMP Research)
Dave Perkins (Peer Networks)
Randy Presuhn (Peer Networks)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Jon Saperia (BGS Systems)
Bob Stewart (Cisco Systems, bstewart@cisco.com), chair
Kaj Tesink (Bellcore)
Glenn Waters (Bell-Northern Research)
Bert Wijnen (IBM)

7. References

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

[2] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1902, January 1996.