Structure of Management Information (SMI)
(RFC 1155)

Object Syntax

- All the objects within a MIB are formally defined.

- ASN.1 (Abstract Syntax Notation) is used to define each object of the MIB.

- ASN.1 specifies:
  - data type of the object
  - allowable forms of the object
  - relationship to other object within the MIB

- Each type of the object has an identifier associated with

  OBJECT IDENTIFIER (OID)

  The object is named using its OID.

- Each OID has a unique value. The value is a sequence of integers

  The sequence has a tree structure.
  It begins with the root of the tree.

- One subtree under (dod) node is allocated for administration of the internet.

  Internet OBJECT IDENTIFIER:: = {iso(1) org(3) dod (6) 1}

- SMI defines 4 nodes under internet

  - directory: future use of OSI-Dir (x.500)
  - management: used for MIB-objects
  - experimental: used for Internet Experiments Objects
  - private: used for private defined object
    it has also a child enterprises
    this allows vendors to enhance the mgmt. of their devices

ASN.1 – Classes

1. UNIVERSAL:

Application – Independent data types
Data types allowed:

- Integer	(UNIVERSAL 2)
- Octetstring	(UNIVERSAL 4)
- Null	(UNIVERSAL 5)
- Object Identifier	(UNIVERSAL 6)
- Sequence, Sequence of	(UNIVERSAL 16)

1.1. Object-Identifier (OID)

- Unique identifier of an object
- OID = sequence of integer
- Sequence = identifies the location of object within the MIB structure

Example: tcpConnTable

ISO	org	dod	internet	mgmt	mib-2	tcp	tcpConnTable
1	3	6	1	2	1	6	13

i.e => ***OID = 1.3.6.1.2.1.6.13***

1.2. Sequence and Sequence of

Sequence = Define an ordered list of values of one or more other data types

Ex: tcpConnTable

Sequence of = Ordered, variable nr. of elements, all of one type

The data types ***sequence*** and ***sequence of*** are used to construct tables.

2. APPLICATION – TYPE

- Each application has to define its own APPLICATION Data Type.

- RFC 1155 defines the following APPLICATION-Data Types.

  - networkaddress: = IPAddress, at the moment the only defined address

  - ipaddress: 32 Bit address (IP-convention)

  - counter: Non-Negative Integer which may be only incremented
    max. value: $2^{32} - 1$
    if counter = Max Value
    then counter = 0 (wrap to 0)

  - gauge: Non-Negative Integer
    if gauge = Max. Value
    then gauge = Decrement Value
    (Gauge do not wrap to 0)

  - time – ticks: Non-Negative – Integer
    Counts the hundreds of seconds since one event
    (i.e. Cold Start)

  - opaque: supports arbitrary data
    data are encoded as OCTET STRING.
    the data maybe in any ASN.1 format.

Objects Definition

- A MIB consists out of set of objects
- Each Object has:
  - Type and
  - Value
- Definition of an object-type => Syntactic Description
- Particular Instance of an Obj.Type => Object Instance
- Macro definition => A set of related Obj Types used to define managed objects

- Macro Instance: an instance generated from a macro by supplying arguments for the parameters

- Macro Instance Value: specific entity with a specific value

- RFC1155 defines objects for MIB I

- RFC1212 defines objects for MIB II

- OBJECT-TYPE Macro (see Def. of OBJECT-TYPE Macro)

  - Defined by RFC1212

  - The key components:

  - SYNTAX: - Abstract syntax for an Obj-Type
    - Syntax must use the UNIVERSAL and APPLICATION-WIDE Types.

  - ACCESS: The way in which an obj-Instance may be accessed via SNMP or other protocol. Access restrictions options allowed: RO, RW, WO, NOT-ACCESSIBLE

  - STATUS: Implementation support required for the object.
    Options: a) mandatory = supported
    b) optional
    c) deprecated = not supported by the next version
    d) obsolete = no need to implement support for this Object

  - DescrPart: Text description of the semantic of the Obj.
    This component is optional

  - ReferPart: Text cross-reference to an obj. defined within other MIB-Module

  - Index Part: used for Table Definition

  - DefValPart: defines a Default-Value which maybe used when an Obj-Instance is created

  - VALUE NOTATION: specifies the name used to access the object by SNMP

Example of an Object – Type definition

```
tcpMaxConn OBJECT-TYPE
        SYNTAX       INTEGER
        ACCESS       read-only
        STATUS       mandatory
        DESCRIPTION
           “The limit on the total nr. of TCP Connections -------“
        : : = {  tcp 4 }
```

```
IMPORTS   ObjectName, Object Syntax FROM FRC-1155-SMI

OBJECT-TYPE MACRO : :=
BEGIN
     TYPE NOTATION ::= “SYNTAX”   type (TYPE ObjectSyntax)
                       “ACCESS”   Access
                       “STATUS”   Status
                       DescrPart
                       ReferPart
                       IndexPart
                       DefValPart
VALUE NOTATION ::= value (VALUE ObjectName)

Access ::= “read-only” | “read-write” | “write-only” | “not-accessible”

Status  ::= “mandatory” | “optional” | “obsolete” | “deprecated”

DescrPart ::=“DESCRIPTION” value(description DisplayString) |empty

ReferPart ::= “REFERENCE” value (reference DisplayString) | empty

IndexPart ::= “INDEX” “{IndexTypes “} “

IndexTypes ::= IndexType | IndexTypes “,” IndexType

IndexType ::= value (indexobject ObjectName) -- if indexobject. use the Syntax
```

```
                                          -- value of the correspondent
                                          -- OBJECT-TYPE invocation
              |type (indextype)          -- otherwise use named SMI type;
                                         -- must conform to IndexSyntax below

DefValPart ::= "DEFVAL" "{" value (defvalue ObjectSyntax) "}" empty

DisplayString ::= OCTET STRING SIZE (=..255)

END

IndexSyntax ::= CHOICE { number INTEGER (0..MAX),
                         string OCTET STRING,
                         object OBJECT IDENTIFIER
                         address NetworkAddress,
                         IpAddress IpAddress }
```

Table Definition

- SMI supports only:
  2-dimensional table with scalar values

- The definition uses:
  - "Sequence" and "Sequence of" ASN.1 Obj-Types
  - Index Part of the OBJECT-TYPE Macro

- Example:

tcpConnTable = 1.3.6.1.2.1.6.13

Consider the following instance of a TCP connection table:

tcpConnState	tcpConnLocAdr	...LocPort	..RemAdr	..RemPort
5	**10.0.0.99**	**12**	**9.1.2.3**	**15**
2	**0.0.0.0**	**99**	**0.0.0.0**	**0**
3	**10.0.0.99**	**14**	**89.1.1.42**	**84**
	INDEX	**INDEX**	**INDEX**	**INDEX**

- Contains information about TCP-Connections

- For each connection the following information is stored:
  - State: Connection Status: 11 States are considered
  - Local Address: Local IP-Adr
  - Local Port: Local TCP-Port
  - Remote Address: Remote IP-Adr to which the connection is maintained.
  - Remote Port: Remote TCP-Port

- The Table consists out of :

SEQUENCE OF TcpConnEntry

- Each Element of this sequence is a Row within the table

- Each Row => SEQUENCCE of 5 Scalar – Elements
  The scalar elements are mandatory
  Element-Type: INTEGER, ipAddress, INTEGER,
  ipAddress, INTEGER

INDEX→ determines which Obj.-Value(s) will be used to identify one row in the table
at any time a TCP-Socket (i.e. IP-Adr + TCP-Port) supports only one connection
i.e.:
the identification of a row in the table in that case can be done by using the following INDEX.

LocalAddress, LocalPort, RemoteAddress, RemotePort

- The table contains 3 Rows

- The entire table => Single Instance of the Obj-Type
  tcpConnTable

- Each row = Instance of the Obj-Type
  tcpConnEntry
  i.e.
  entire table = 3 x Instances of tcpConnEntry

- Each scalar elem = exists 3 times within the table (i.e. 3 instances of obj tcpConnState)

- Each scalar element can be uniquely identified by using

Column Obj-ID + INDEX

Example:
Consider the table analyzed above

*		**		

a) the scalar element " * " has to be identified.

Column = tcpConnState = 1.3.6.1.2.1.6.13.1.1
INDEX = 10.0.99.12.9.1.2.3.15

i.e. OID for scalar element "*" is:

OID = > 1.3.6.1.2.1.6.13.1.1.10.0.0.99.12.9.1.2.3.15

The returned value will be => 5

b) the scalar element " ** " has to be identified

Column = tcpConnLocal Port = 1.3.6.1.2.1.6.13.1.3
INDEX = 10.0.99.12.9.1.2.3.15

i.e. OID for scalar elem "**" is:

OID => 1.3.6.1.2.1.6.13.1.3.10.0.0.99.12.9.1.2.3.15 = 12

- Let us present all of the above more compact.

The Index for each row of the table is:

INDEX 1 = 10.0.0.99.12.9.1.2.3.15

INDEX 2 = 0.0.0.0.99.0.0.0.0.0

INDEX 3 = 10.0.0.99.14.89.1.1.42.84

tcpConnEntry = 1.3.6.1.2.1.6.13.1 = X

The table can be represented as follows:

X.1.INDEX1	**X.2.INDEX1**	**-------------**	**X.5.INDEX1**
X.1.INDEX2	**X.2.INDEX2**	**-------------**	**X.5.INDEX2**
X.1.INDEX3	**X.2.INDEX3**	**-------------**	**X.5.INDEX3**

tcpConnTable OBJECT-TYPE

```
            SYNTAX    SEQUENCE OF TcpConnEntry
            ACCESS   not-accessible
            STATUS    mandatory
            DESCRIPTION
                      "A table containing TCP connection-specific information."
            ::= {  tcp 13  }

tcpConnEntry OBJECT-TYPE
             SYNTAX   TcpConnEntry
            ACCESS   not-accessible
            STATUS    mandatory
            DESCRIPTION
                      "Information about a particular TCP connectin. An object of
                         this type is transient, in that it ceases to exist when (or
soon                          after) the connection makes the transition to the
CLOSED                             state."
             INDEX    { tcpConnLocalAddress,
                         tcpConnLocalPort,
                         tcpConnRemAddress,
                         tcpConnRemPort   }
              ::= { tcpConnTable 1 }

tcpConnEntry ::= SEQUENCE { tcpConnState INTEGER,
                         tcpConnLocalAddress IpAddress,
                         tcpConnLocalPort   INTEGER   (0..65535),
                         tcpConnRemAddress  IpAddress
                         tcpConnRemPort  INTEGER (0..65535) }

tcpConnState OBJECT-TYPE
           SYNTAX      INTEGER  { closed  (1),
                                  listen    (2),
                                  synSent (3),
                                  synReceived   (4),
                                  established      (5),
                                  finWait1     (6),
                                  finWait2     (7),
                                  closeWait   (8),
                                  lastAck       (9),
                                  closing         (10),
                                  timeWait       (11),
                                  delete TCB   (12)  }
             ACCESS     read-write
```

```
        STATUS     mandatory
        DESCRIPTION        "The state of this TCP connection.
        The only value which may be set by a management station is
          delete TCB(12)." Accordingly, it is appropriate for an agent to return
          a 'badValue' response if a management station attempts to set this
          object to any other value.

          If a management station sets this object to the value
          deleteTCB(12), then this has the effect of deleting the TCB (as
          defined in RFC 793) of the corresponding connection on the
          managed node, resulting in immediate termination of the connection.

          As an implementation-specific option, a RST segment my be sent
          from the managed node to the other TCP endpoint (note however
          that RST segments are not sent reliably)."

    ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS    mandatory
        DESCRIPTION
            "The local IP address for this TCP connection. In the case of a
              connection in the listen state which is willing to accept connections
              for any IP interface associated with the node, the value 0.0.0.0 is
              used."
    ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
        SYNTAX   INTEGER (0..65535)
        ACCESS   read-only
        STATUS    mandatory
        DESCRIPTION
            "The local port number for this TCP connection."
    ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
        SYNTAX   IpAddress
        ACCESS   read-only
        STATUS    mandatory
```

```
        DESCRIPTION
            “The remote IP address for this TCP connection.”
    ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
        SYNTAX   INTEGER (0..65535)
        ACCESS   read-only
        STATUS    mandatory
        DESCRIPTION
            “The remote port number for this TCP connection
     ::= { tcpConnEntry 5 }
```

Example 2: **ipRouteTable**

IP-Group within the MIB contains following Objects:

```
ip (mib-2 4)
    :
    :
    ipRouteTable (21)
          |
          ipRouteEntry(1)
               |-ipRoute Dest(1)
               |-ipRouteIfIndex(2)
               |-ipRouteMetric1(3)
               |-ipRouteMetric2(4)
               |-ipRouteMetric3(5)
               |-ipRouteMetric4(6)
               |-ipRouteNextHop(7)
               |-ipRouteType(8)
               |-ipRouteProto(9)
               |-ipRouteAge(10)
               |-ipRouteMask(11)
               |-ipRouteMetric5(12)
               |-ipRouteInfo(13)
```

ipRouteDest		**ipRouteMetric1**		**ipRouteNextHop**	
9.1.2.3	**---**	**3**	**---------**	**99.0.0.3**	**-----**
10.0.0.51	**---**	**5**	**---------**	**89.1.1.42**	**-----**
10.0.0.99	**---**	**5**	**---------**	**89.1.1.42**	**-----**

ipRouteTable = 1.3.6.1.2.1.4.21
ipRouteEntry = 1.3.6.1.2.1.4.21.1 = X
INDEX = ipRouteDest = 1.3.6.1.2.1.4.21.1.1

INDEX1 = 9.1.2.3

INDEX2 = 10.0.0.51

INDEX3 = 10.0.0.99

ipRouteDest		ipRouteMetric1		ipRouteNextHop	
X.1.INDEX1	**-----**	**X.3.INDEX1**	**-----**	**X.7.INDEX1**	**-------**
X.1.INDEX2	**-----**	**X.3.INDEX2**	**-----**	**X.7.INDEX2**	**-------**
X.1.INDEX3	**-----**	**X.3.INDEX3**	**-----**	**X.7.INDEX3**	**-------**

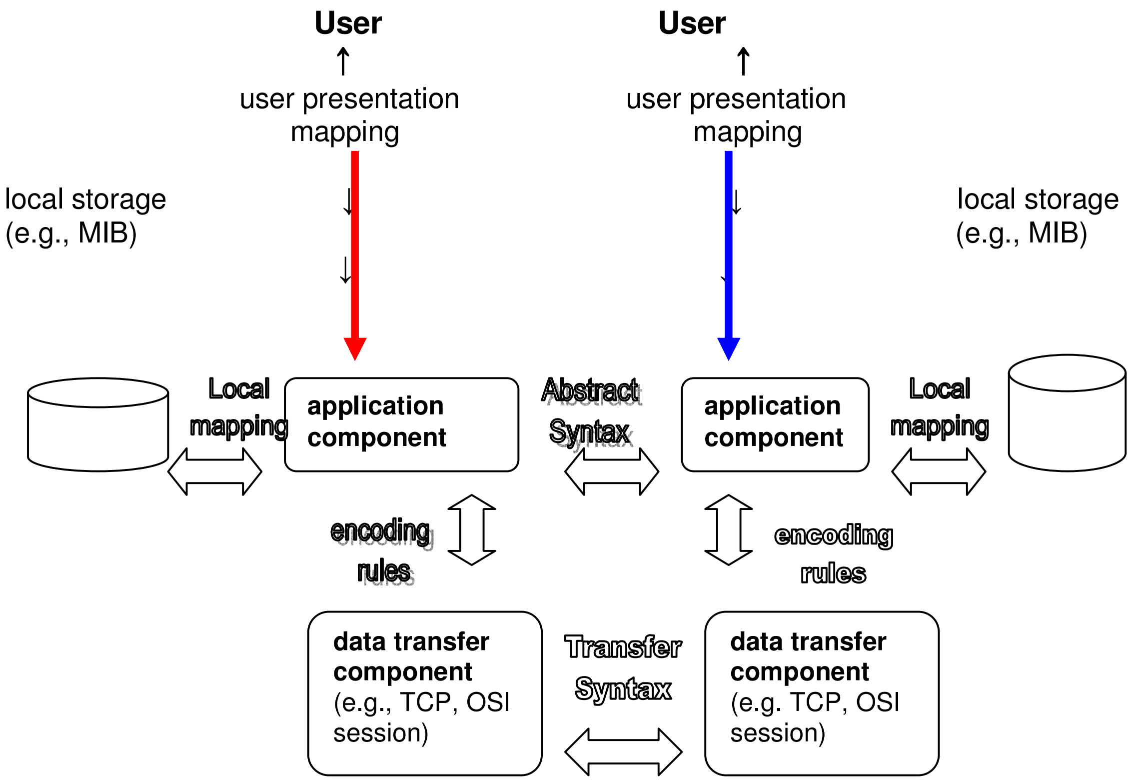
ASN.1: Abstract Syntax Language

- A formal language developed and standardized by
  - ITU (X.208)
  - ISO 8824

- ASN.1 is used to define
  - Abstract Syntax of application data
  - the structure of presentation protocol data units PDUs
  - management information base (MIB) for SNMP and for OSI system management

- The meaning of Abstract Syntax:
  Consider the following model:
  (s. fig. B1/pg. 563 → Stallings)

Reference Model Architecture :



The use of abstract and transfer syntaxes

1. Data Transfer Component:

- Mechanism for transferring data between the end systems
- In case of TCP/IP – stack this component consists of SNMP, FTP, etc.
- The data received from the application are binary values or a sequence of bytes
  The binary value is assembled into the Service Data Unit (SDU) and into PDU for passing between protocol entities.

2. *Application Component:*

- Concerned with the user's view of data which is semantics of data
- Provides a representation of data that can be converted into binary values → i.e. syntax of data
- Under ASN.1, application component represents Data-Type & Data Values using an abstract syntax
- Abstract Syntax specifies data independently from any specific representation (similar to the data-type representation C-language)
- Application protocols describe their PDUs using abstract syntax.

Advantage: Application components of different systems may exchange information

- The elements within MIB are also defined using abstract syntax
  The application (management programs) convert this abstract notation in order to store the information in the memory.

Conclusion:

A) Application Component translates between:
Abstract syntax of the application and
Transfer syntax that describes the data values in a binary from

B) Advantage for using abstract and transfer syntax

a) Common representation for exchange data between different systems

b) In each particular system an application uses special representation of data
i.e. the method of using abstract/transfer syntax resolves the differences in representation between cooperating applications

ASN.1 Concepts

- Module

  → Building Block of ASN.1

  → Helps to define data structure:
  A structure is in form of a named module
  Name of the module can be used to name the structure (see examples)

- Abstract Data Types

  Type => Collection of values

  Type classes:
  - simple → does not have components

  - structured → it has components

  - tagged: type derived from other types

  - other: includes CHOICE and ANY-type: types without tags

    CHOICE → list of alternative known types only one is used to create a value

    ANY → arbitrary value of an arbitrary type used when the possible type is not known in advance

Data Types:

- A data type is identified by its tag.
  2 or more ASN.1 types are the same if
  their tag-numbers are the same

- Classes of Data Types:

  - UNIVERSAL: - application independent types
    - defined in the RFC
      (see table pg 568 Stallings)

  - APPLICATION-WIDE: relevant for a particular application

  - CONTEXT SPECIFIC: but applicable in a limited context

  - PRIVATE: defined by users: not specified by any standard

  Example: Data Type = UNIVERSAL 3

Class:	UNIVERSAL
Tag:	3
Data Values:	Sequence of Bits

- Macro Definition: ASN.1 allows the user to extend the syntax of ASN.1 by defining types or family of type and their values

Basic Encoding Rules (BER)
(Standard CCITT = X.209 and ISO 8825)

- Encoding Structure: based on *type-length-value* (TLV)
  Each ASN.1 value may be encoded with TLV

  - type → indicates: - ASN.1-type
    - Class of type
    - Encoding primitive or constructed

  - length → indicates length of the value representation

  - value → value of ASN.1 as a string of octets

. - Encoding structure is recursive:
  Value portion of TLV => one or more TLV structures

- Encoding methods:

  1) Primitive → definite length encoding

  2) Constructed → definite length encoding

  3) Constructed → indefinite length encoding

(a) Encoding of each value

Identifier	Length	Contents

definite-length encoding

Identifier	Length	Contents	EOC

EOC = 0000_{16}

indefinite-length encoding

(b) Identifier field

←←←← one octet→→→→

Class	P/C	Tag number

←←←← leading octet→←2nd octet→ ← last octet→

Class	P/C	11111	1	XXXXXXX	………	0	XXXXXXX

Class:
00 = universal
01 = Application
10 = Context specific
11 = Private

P/C = primitive encoding
P/C = constructing encoding

Tag number:
1 = Boolean type
2 = integer type
3 = Bitstring type
4 = Octetstring type
5 = Null type
6 = Object identifier type
9 = Real Type
10 = Enumerated type
16 = Sequence and sequence-of types
17 = Set and set-of types
18-22, 25-27 = Character string types
23-24 = Time types
>30:XX…X = Tag number

(c) Length field

←←←← one octet→→→→

0	Length (L)

short definite form: $1 \leq L \leq 127$

←←one octet→←←K octets→→

1	K	Length (L)

long definite form: $128 \leq L \leq 2^{1008}$

←←←← one octet→→→→

1	00000000

indefinite form; value terminated by EOC

Tag	Type Name	Set of Values
		BASIC Types
UNIVERSAL 1	BOOLEAN	TRUE or FALSE
UNIVERSAL 2	INTEGER	The positive and negative whole numbers, including zero
UNIVERSAL 3	BIT STRING	A sequence of zero or more bits
UNIVERSAL 4	OCTET STRING	A sequence of zero or more octets
UNIVERSAL 9	REAL	Real numbers
UNIVERSAL 10	ENUMERATED	An explicit list of integer values that an instance of a data type may take
		OBJECT TYPES
UNIVERSAL 6	OBJECT IDENTIFIER	The set of values associated with information objects allocated by this standard
UNIVERSAL 7	Object descriptor	Human-readable text providing a brief description of an information object
	CHARACTER STRING TYPES	
UNIVERSAL 18	NumericString	Digits 0 through 9, and the space character
UNIVERSAL 19	PrintableString	Printable characters
UNIVERSAL 20	TeletexString	Character set defined by CCITT Recommendation T.61
UNIVERSAL 21	VideotexString	Set of alphabetic aqnd graphical characters defined by CCITT Recommendations T.100 and T.101
UNIVERSAL 22	IA5String	International alphabet 5 (equivalent to ASCII)
UNIVERSAL 25	GraphicString	Character set defined by ISO 8824
UNIVERSAL 26	VisibleString	Character set defined by ISO 646 (equivalent to ASCII)
UNIVERSAL 27	GeneralString	General character string
	MISCELLANEOUS TYPES	
UNIVERSAL 5	NULL	The single value NULL; commonly used where several alternatives are possible but none of the applies

UNIVERSAL 8	EXTERNAL	A type defined in some external document (It need not be one of the valid ASN.1 types.)
UNIVERSAL 23	UTCTime	Consists of the date – specified with a two-digit year, a two-digit month, and a two-digit day – followed by the time specified in hours, minutes, and optionally seconds – followed by an optional specification of the local time differential from universal time.
UNIVERSAL 24	GeneralizedTime	Consists of the date – specified with a four-digit year, a two-digit month, and a two-digit day – followed by the time – specified in hours, minutes, and optionally seconds – followed by an optional specification of the local time differential from universal time.
UNIVERSAL 9-15	Reserved	Reserved for addenda to the ASN.1 standard
UNIVERSAL 28-	Reserved	Reserved for addenda to the ASN.1 standard

STRUCTURED TYPES

UNIVERSAL 16	SEQUENCE and SEQUENCE-OF	*Sequence:* defined by referencing a fixed, ordered list of types; each value is an ordered list of values, one from each component type *Sequence-of:* defined by referencing a single existing type; each value is an ordered list of zero or more values of the existing type
UNIVERSAL 17	SET and SET-OF	*Set*: defined by referencing a fixed, unordered list of types, some of which may be declared optional; each value is an unordered list of values, one from each component type *Set-of:* defined b referencing a single existing type; each value is an unordered list of zero or more values of the existing type.

1) Primitive Method

Fields: (s. figure)

- Identifier

- Length: No. of octets of the contents field.
  in case length > 128
  then K = specifies no. of additional length octets

- Contents: ASN.1 value as a string of octets
  Encoding rules are précised in the standard.
  i.e.
  Integer: 2's complement notation max. $2^{32} - 1$

  Bit, Octet, Ch. Strings:
  may be encoded primitive or constructed (i.e. broken up into a no. of substrings)

  Obj. Identifier (OID)
  - Sequence of integer:
    First 2 integer = single subidentifier
    i.e. OID with N integer = N -1 SubID
    Formula:
    $Z = (X \times 40) + Y$

    where: $X = 1^{st}$ integer
    $Y = 2^{nd}$ integer

  Result:

SubID Value	1^{st} Integer	2^{nd} Integer
$0 \leq Z \leq 39$	0	Z
$40 \leq Z \leq 79$	1	Z-40
$8 \leq Z$	2	Z-80

2) **Constructed – Definite – Length Encoding**

- Used for: - simple string types: sequence, sequence of defined by implicit and explicit tagging

Identifier: s. figure

Length: similar with primitive encoding

Contents: concatenation of the complete BER encoding (identifier, length, contents) of the components of the value

3) **Constructed Indefinite – Length Encoding**

- Used for:
  - Simple string types
  - Structured types
  - Structured derived from simple and structured types
  - any type defined by explicit tagging.

- It does not require that "length" is known in advance.

- Fields:

Identifier: s. constructed definite length

Length: 1 x Octet = 80H

Contents: s. constructed definite length

End of Contents: 2 x Octets = 0000H

Example 1

Representation of a counter with a value → $2^{32}-1$ = FF FF FF FF$_{16}$

- Definition of Counter-Type

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

- Value Encoding

  Hexadecimal Encoding =

41	05	00 FF FF FF FF
Identifier	Length	Contents

(1) Identifier

0	1	0	0	0	0	0	1

Application — Primi-tive Encod-ing — Tag Number

(2) Length

0	0	0	0	0	1	0	1

ID for Short form: 1≤L≤128 — Length of Contents = 5 Bytes

(3) Contents 2's Compliment of the FF FF FF FF_{16}

1. Byte 2. Byte 3. Byte 4. Byte 5. Byte

0 0	F F	F F	F F	F F

Integer ⏮⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⏭
Non-neg. Number

Rule for integer non-negative: 1st Bit = 0 (Most significant Bit = MSB)
Remaining Bits= Binary Form of the Number

<u>Example 2:</u>

Representation of an object Identifier (OID)

Suppose an OID = 1.3.6.1.2.1.1.1.0

<u>Encoding value:</u>

<u>06 08 2B 06 01 02 01 01 01 00</u>

(1) <u>Identifier</u>

0	0	0	0	0	1	1	0

⏮ ⇔⇔ ⏭ ⏮⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⏭

UNIVERSAL Object Identifier

(2) <u>Length</u>

0	0	0	0	1	0	0	0

⏮⏭ ⏮⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⇔⏭

ID
SF

Length = 8 Bytes

(3) <u>Contents</u> = 8 Bytes

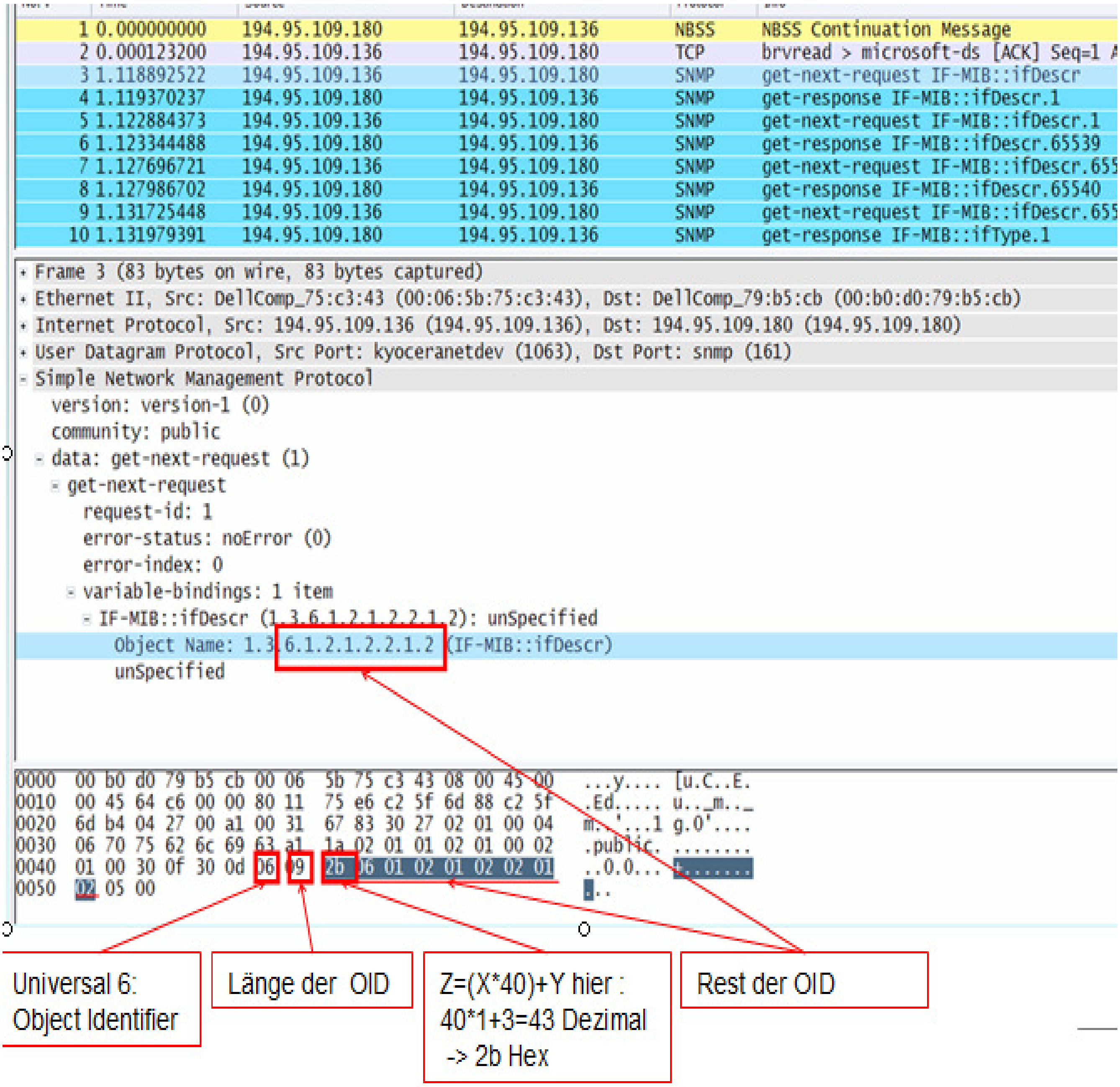
1st Byte → Encoding of the first OID – Sub-identifiers according to the following rule:

1^{st} ID x 40_{Dec} + 2^{nd} ID =

1 x 40_{DEC} + 3 = 43_{DEC} = $2B_H$

2^{nd} Byte to 8^{th} Byte = Hex Value of the MIB-Tree
i.e. 06 01 02 01 01 01 00

Beispiel : TLV - OID



Beispiel: TLV - String