ASN.1
Abstract Syntax Notation One

Facultad de Ingeniería – Tecnólogo en Telecomunicaciones, 2016

ASN.1

- ASN.1 : Abstract Syntax Notation One
  - Es un lenguaje formal diseñado para especificar la sintaxis de tipos de datos.
  - Tipos de datos a ser intercambiados entre diferentes sistemas.
    - Algunos ejemplos:
      - Telefonía (SS7, Otros)
      - Banca, tarjetas de crédito
      - Aviación
      - Redes de Datos (SNMP, LDAP)

ASN.1

- Es un standard de ISO, desarrollado originalmente a través del ITU-T.

  - X.208 de ITU-T
  - X.680 de ITU-T
  - Período de trabajo: 1988-1997

ASN.1

- Algunas definiciones:
  - Application Component (AC):
    - representa a las entidades de aplicación.
      - bases de datos
      - archivos de texto o de imágenes
      - Información de gestión
    - El AC ve la información de acuerdo a la aplicación concreta, por ello la información para el AC tiene más estructura

ASN.1

- Algunas definiciones:
  - Data Transfer Component (DTC):
    - representa a las entidades que se encargan de transportar la información a través de la red.
      - TCP/UDP/IP
      - Protocolos del stack OSI
    - El DTC ve la información como una secuencia de valores binarios de bytes que deben ser transportados a través de una red.

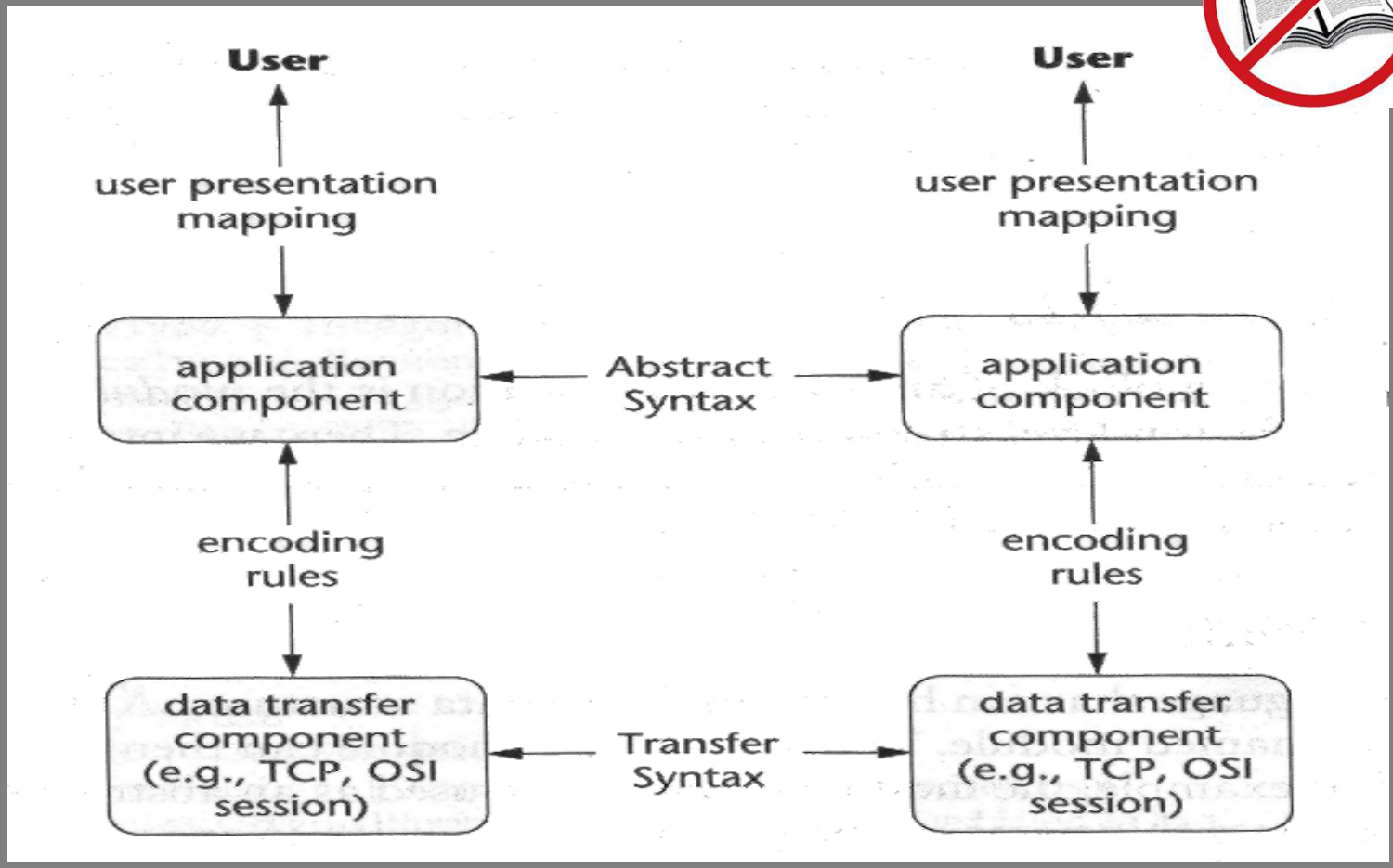
ASN.1

- Algunas definiciones:
  - sintaxis abstracta:
    - sintaxis que utilizamos para describir los tipos de datos de manera independiente de la codificación utilizada para transmitir, del lenguaje de programación y de la implementación en general.

ASN.1

- Algunas definiciones:
  - sintaxis de transferencia:
    - especificación del tipo de datos en patrones de bits y bytes que son transmitidos a través del cable.
  - reglas de codificación (*encoding rules*):
    - conjunto de reglas que nos permiten convertir de la sintaxis abstracta a la sintaxis de transferencia.

ASN.1


User
User
user presentation mapping
user presentation mapping
application component
Abstract Syntax
application component
encoding rules
encoding rules
data transfer component (e.g., TCP, OSI session)
Transfer Syntax
data transfer component (e.g., TCP, OSI session)

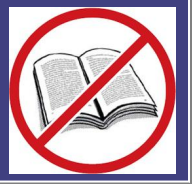
ASN.1

- Resumiendo :
  - ASN.1 nos proporcionará dos grandes grupos de funcionalidad :
    - Un conjunto de reglas y una sintaxis para especificar tipos de datos.
    - Un conjunto de reglas (BER) que especifican cómo codificar los tipos de datos anteriormente definidos para su transmisión a través de la red.
      - Para decodificar lo recibido ídem!

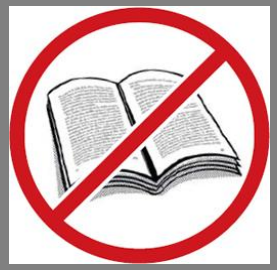
ASN.1

- Reglas para definir tipos de datos :
  - La unidad básica de ASN.1 es el *módulo*. Todas las definiciones de tipos de datos ocurrirán dentro de algún módulo.
  - La sintaxis para definir un módulo tiene la forma:

```
<modulereference> DEFINITIONS ::=
        BEGIN
                EXPORTS
                IMPORTS
                AssignmentList
        END
```



ASN.1

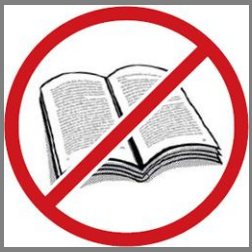


- El texto es de formato libre :
  - espacios blancos pueden repetirse, lineas blancas también.
- Comentarios :
  - Encerrados entre doble guión:
    - -- esto es comentario --
  - Encerrados entre doble guión y fin de línea:
    - -- esto es comentario

ASN.1

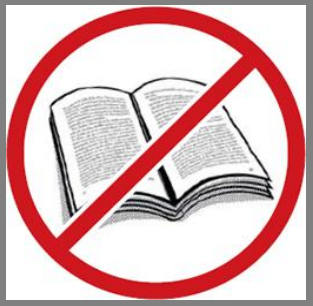
- Tipos de datos abstractos :
  - Para ASN.1 un tipo de datos es una colección de valores.
  - Pueden ser de cuatro tipos :
    - simples:
      - tipos atómicos, sin estructura, escalares.
    - estructurados:
      - tipos que contienen componentes.
    - *“tagged”:*
      - cualquier tipo definido en función de otro tipo
    - other:
      - Algunas construcciones especiales, `CHOICE` y `ANY`

ASN.1



- Tags o Marcas:
  - Todos los tipos de ASN.1 diferentes de `CHOICE` o `ANY` tienen un tag (marca).
  - El tag se compone de un *nombre de clase* y de un número no negativo.
  - Se definen cuatro diferentes tipos de tag :
    - Universal
    - Application-Wide
    - Context-Specific
    - Private

ASN.1

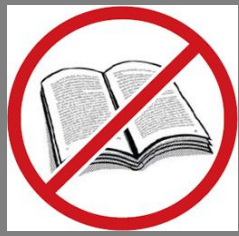


- Tipos Universales Simples:
  - `Boolean:  universal 1`
  - `Integer:  universal 2`
  - `Real:  universal 9`
  - `BitString`
  - `OctectString:`
  - `Enumerated:`
  - `ObjectIdentifier y ObjectDescriptor:`
  - `Null:`
- Muchos más!

ASN.1

- Tipos Universales Estructurados:
  - **SEQUENCE :**
    - Se define a través de una lista ordenada y fija de tipos. El valor de una SEQUENCE es una lista ordenada de valores de cada tipo referenciado.
  - **SEQUENCE-OF :**
    - Se define a través de una referencia a un único tipo. El valor de una SEQUENCE-OF es una lista ordenada de valores del tipo referenciado.
  - **SET :**
    - idem SEQUENCE pero no importa el orden.
  - **SET-OF :**
    - idem SEQUENCE-OF pero no importa el orden.

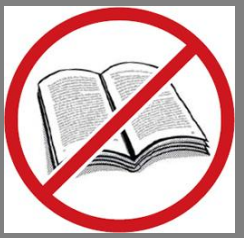
ASN.1



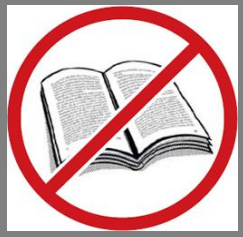
- Definicion de tipos estructurados (definición en BNF):

```
SequenceType ::= SEQUENCE {ElementTypeList} | SEQUENCE { }
ElementTypeList ::= ElementType | ElementTypeList, ElementType
ElementType ::=
   NamedType                    |
   NamedType OPTIONAL           |
   NamedType DEFAULT Value      |
   COMPONENTS OF Type
```

ASN.1



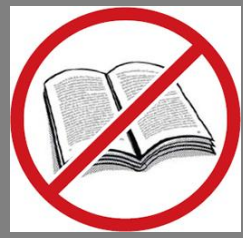
- Tipos *Tagged*:
  - Todos los tipos menos `CHOICE` y `ANY` son tagged en ASN.1!!
  - Además del tag que define a un tipo, puedo definir un nuevo tag.
  - El nuevo tag define un nuevo tipo ***isomorfo*** al anterior, pero distinto de él.
  - Útil cuando dos tipos, a pesar de tener la misma sintaxis, representan distintos objetos.
  - El tag se reflejará en la codificación, por lo cual la distinción de tipos se recuperará luego de la transmisión.



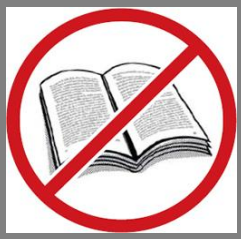
ASN.1

- `CHOICE` y `ANY`: Otros tipos estructurados
  - `CHOICE`:
    - Similar a un `union` de `C`.
    - Es un valor escalar del cual su tipo no se conoce con exactitud y solo será conocido en “runtime”, pero se sabe que pertenece a un cierto conjunto posible de tipos.
      - `choiceType ::= CHOICE (AltTypeList)`
      - `AltTypeList ::= NamedType | AltTypeList, NamedType`

ASN.1



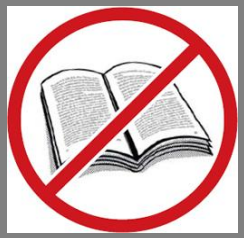
- CHOICE y ANY:
  - ANY:
    - Representa un valor escalar del cual su tipo no se conoce con antelación, pero tampoco se tienen “pistas” de qué puede ser.
    - El valor asignado puede ser de cualquier tipo.
      - `AnyType ::= ANY`



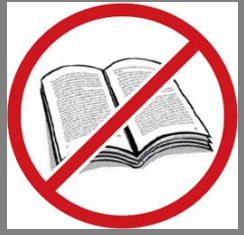
ASN.1

- Subtipos:
  - Un subtipo se deriva de un tipo padre mediante una restricción de los posibles valores que este puede tomar.
  - Esto puede ser recursivo, es decir puedo tener subtipos de subtipos.
    - enteros -> enteros positivos ->enteros positivos pares y enteros positivos impares.
- Subtipos Contenidos :
  - Son aquellos formados mediante la inclusión de varios subtipos de un mismo nivel.
    - enteros positivos = enteros positivos pares + enteros positivos impares.

ASN.1



- Algunas restricciones posibles para formar subtipos :
  - value-range:
    - solo para enteros o reales
  - permited-alphabet :
    - solo para strings
      - solo-vocales
      - solo-digitos



ASN.1

- Algunas restricciones posibles para formar subtipos :
  - size constraint :
    - solo para strings, sequences y sets
    - fija largo máximo y mínimo
  - inner subtyping :
    - solo para sequences, sets y choices
    - solo incluye aquellos combinaciones de valores de los tipos que componen al tipo padre que satisfacen diferentes vínculos.

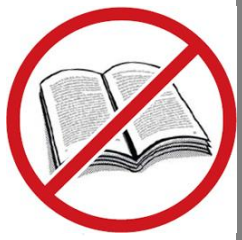
ASN.1

- Ejemplos de ASN.1 : BD de Personal

```
Name:                   John P Smith
Title:                  Director
Employee Number:        51
Date of Hire:           17 September 1971
Name of Spouse:         Mary T Smith
Number of Children:     2

Child Information
    Name:               Ralph T Smith
    Date of Birth:      11 November 1957

Child Information
    Name:               Susan B Jones
    Date of Birth:      17 July 1959
```

ASN.1

- Ejemplos de ASN.1: BD de Personal



```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
        Name,
        title [0] VisibleString,
        number EmployeeNumber,
        dateOfHire [1] Date,
        nameOfSpouse [2] Name,
        children [3] IMPLICIT SEQUENCE OF ChildInformation DEFAULT {} }

ChildInformation ::= SET {
        Name,
        dateOfBirth [0] Date}

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
        givenName VisibleString,
        initial VisibleString,
        familyName VisibleString }

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT VisibleString -- YYYYMMDD
```

ASN.1

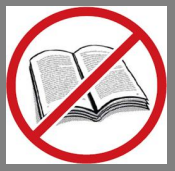
- Macros:
  - Macros son las facilidades que nos da ASN.1 para definir nuevos tipos iguales en todo aspecto a los tipos predefinidos.
  - Cuando hablamos de una “macro” hablamos de un nuevo tipo de datos definido en función de los existentes.
  - Las definiciones de la SMI de SNMP hacen uso intenso de macros.

ASN.1

- Macros:
  - Generar extensiones a la sintaxis de ASN.1
  - Definir nuevos tipos con nuevas reglas y sus valores.
  - Diferentes niveles:
    - notación de las macros
    - definición de una macro
    - instancias de una macro

ASN.1

- Macros
  - Una macro genera toda una familia de nuevos tipos a través de sus instancias.
  - Se comportan de manera similar a un `template` de C++
  - Las macros NO extienden las reglas de codificación, sino que luego de ser instanciadas, los tipos resultantes se codifican según las reglas de codificación



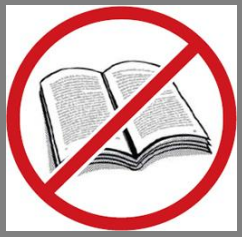
ASN.1

- Macros:
  - Forma General:

```
<macroname> MACRO ::=
BEGIN
    TYPE NOTATION ::= <new-type-syntax>
    TYPE VALUE ::= <value-syntax>
    <supporting productions>
END
```

  - Para las macros se debe escribir su nombre todo en mayúsculas.

ASN.1



- new-type-syntax y value-syntax se definen aplicando BNF (forma Backus-Naur)
- En `supporting-productions` se definen todos aquellos símbolos no terminales que hayan sido utilizados para definir alguna de las anteriores partes.
- Las strings que aparecen entre comillas son parte textual de la definición de la macro.

ASN.1

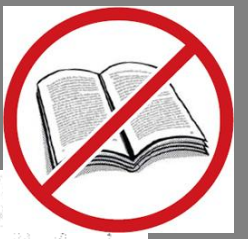
- Macros:
  - La palabra clave `type` se utiliza para determinar en que punto de la nueva notación se espera un nombre de tipo.
    - `type (nombre-de-tipo)`
  - La palabra clave `value` se utiliza para determinar en que punto de la nueva notación se espera un valor de un tipo determinado.
    - `value (nombre-de-valor nombre-de-tipo)`
  - La palabra clave VALUE define finalmente la implementación del nuevo tipo que hemos definido.

ASN.1

- Macros : Ejemplo
  - “Par ordenado” de dos tipos cualesquiera.
  - Sintaxis propuesta:
    - `PAIR TYPE-X=type1 TYPE-Y=type2`
  - Notación para los valores:
    - `(X=valor1, Y=valor2)`
  - Ejemplo de uso :
    - `T1 ::= PAIR TYPE-X=INTEGER TYPE-Y=BOOLEAN`
    - `(X=3, Y=TRUE)`

ASN.1

- Macros: Ejemplo de definición



```
PAIR MACRO ::=
BEGIN
  TYPE NOTATION ::=
    "TYPE-X" "=" type (Local-type-1)   --Expects any ASN.1 type and assigns it
                                       --to the variable Local-type-1
    "TYPE-Y" "=" type (Local-type-2)   --Expects a second ASN.1 type and assigns i-
                                       --to the variable Local-type-2

  VALUE NOTATION ::=
    "("
    "X" "=" value (Local-value-1 Local-type-1) --Expects a value for the type in
                                               --Local-type-1 and assigns it
                                               --to the variable Local-value-1
    "Y" "=" value (Local-value-2 Local-type-2) --Expects a value for the type in
                                               --Local-type-2 and assigns it
                                               --to the variable Local-value-2

    <VALUE SEQUENCE {Local-type-1, Local-type-2}
      ::= {Local-value-1, Local-value-2} >     --This "embedded" definition return-
                                               --the final value as the value of -
                                               --sequence of the two types

    ")"
END
```

ASN.1

- Basic Encoding Rules:
  - Las BER son el conjunto de reglas que a partir de la especificación en ASN.1 de un tipo de datos nos permiten generar su representación en secuencias de bytes adecuadas para su transmisión.
  - Estandarizadas en X.209 (ITU-T) e ISO 8824

ASN.1

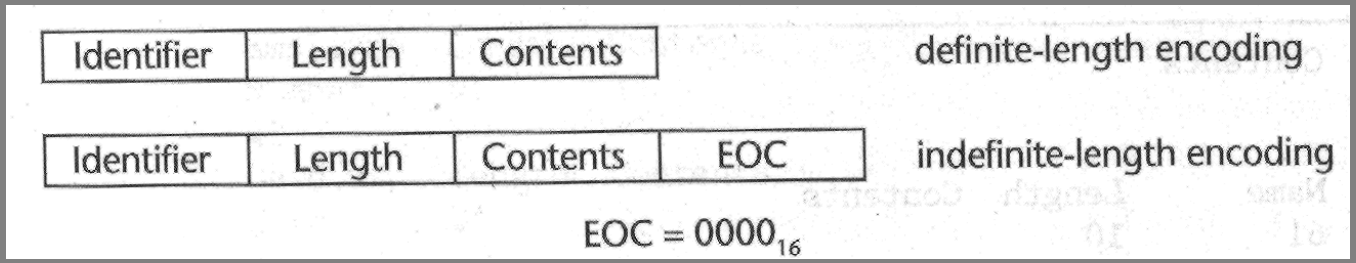
- BER:
  - Se definen formas de codificación para cada tipo primitivo, las cuales pueden no ser únicas.
  - La codificación se basa en ternas T-L-V (type-length-value).
  - Todo tipo simple se puede codificar como una TLV.
  - Para los tipos estructurados, esta codificación se puede aplicar de manera recursiva. En el campo Valor de una TLV puede ir una nueva estructura TLV.

ASN.1

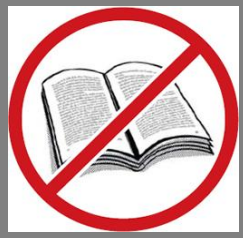
- BER:
  - `type`: indica el tipo de los datos, así como la clase y si la codificación es primitiva o construida.
  - `lenght`: indica el largo de la representación del valor.
  - `value`: representa el valor del objeto como una cadena de bytes.
  - Hay tres métodos para codificar los valores :
    - primitivo y de largo conocido
    - construido y de largo conocido
    - construido y de largo indefinido

ASN.1

- BER:
  - T-L-V:


Identifier
Length
Contents
definite-length encoding
Identifier
Length
Contents
EOC
indefinite-length encoding
EOC = 0000$_{16}$

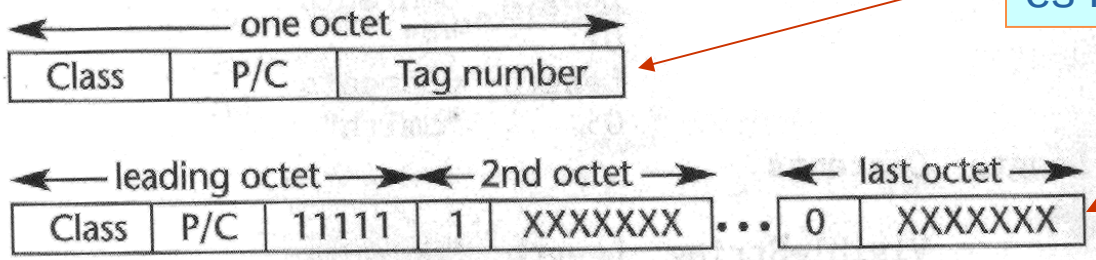
ASN.1



- BER: Campo *identifier*:

Si el tag number es menor de 31

Si el tag number es mayor de 31

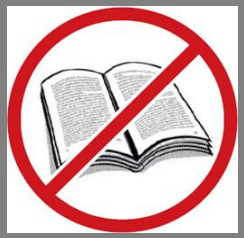


Class:
00 = Universal
01 = Application
10 = Context specific
11 = Private

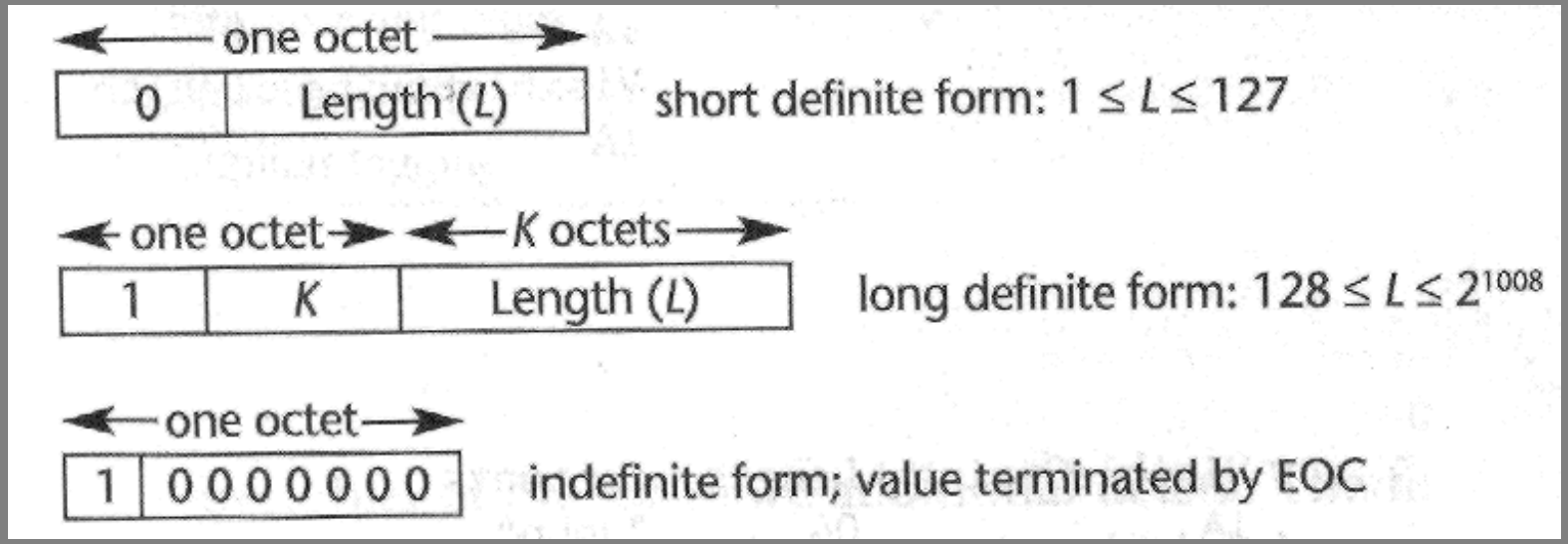
P/C = primitive encoding
P/C = constructed encoding

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

ASN.1



- BER: Campo *length*:

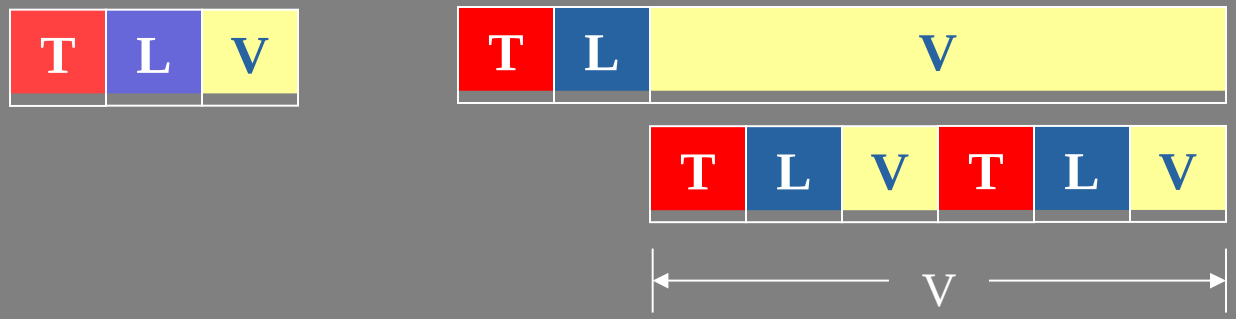


ASN.1

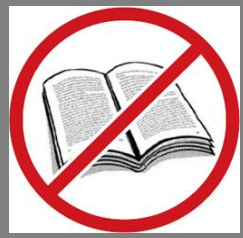
- BER: Campo *contents*:
  - Incluye la representación en cadena de bytes del valor del objeto codificado.
  - El estándar define cómo se hace esto para cada tipo primitivo.
    - Los strings irán como una tira de bytes
    - Los enteros irán como 32 o 64 bits

ASN.1

- Codificación construida y de largo definido:
  - Se utiliza para codificar strings, SEQUENCE's y SET's y tipos *tagged*.
  - En el caso de las SEQUENCEs y SETs en el campo Value se concatenan las representaciones BER de cada uno de los componentes del tipo.



ASN.1



- BER: Codificación Construida y de largo indefinido:
  - Usos similares al anterior.
  - No se necesita conocer el largo de la codificación previamente.

ASN.1

- Codificación del Contenido:
  - Las BER definen las reglas de codificación para cada tipo primitivo.
  - Enteros representados en complemento a 2.
  - Booleanos representados como un byte con cero (*false*) o algún valor no-cero (*true*)
  - Null representado con contenido vacío
  - Strings representadas a través de los valores de los caracteres que las componen.

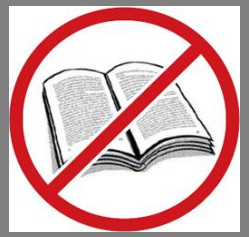
Definiciones SMI
Structure of Management Information

Facultad de Ingeniería – Tecnólogo en Telecomunicaciones, 2016

SMI

- **SMI**: Structure of Management Information.
- Es un conjunto de definiciones básicas comunes a todos los módulos MIB que van a contener definiciones con información de gestión.
- SMI, SMIv2, SMIng

SMI



- Definida en :
  - RFC1157 y RFC1212 para la versión 1 de SNMP.
- Se divide en tres partes:
  - Definiciones de módulos (*information modules*)
    - Define una macro de ASN.1, MODULE-IDENTITY
  - Definiciones de objetos
    - Define una macro de ASN.1, OBJECT-TYPE
  - Definiciones de notificaciones
    - Define una macro de ASN.1, NOTIFICATION-TYPE

SMI

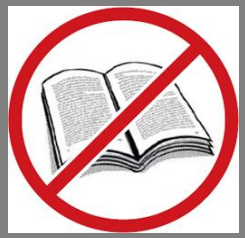
- La SMI:
  - Define el marco general bajo el cual las diferentes MIBs serán construidas.
  - Acotar la complejidad y las “ambigüedades” de ASN.1 para asegurar la compatibilidad.
    - En ASN.1 muchas cosas pueden hacerse de varias maneras!
  - Acotar las posibilidades del ASN.1 a las limitaciones del protocolo SNMP.

SMI

- La SMI:
  - Provee una forma estandarizada para agregar una nueva MIB.
  - Provee una sintaxis estandarizada para definir nuevos objetos, así como la sintaxis de sus valores.
  - Una técnica estandarizada para codificar estos valores.

SMI

- Macro **`OBJECT-SYNTAX`**:
  - La SMI define los tipos de datos permitidos:
    - Básicos (Universales de ASN.1)
      - `Integer, Octectstring, Null, Object identifier, Sequence.`
    - De Aplicación :
      - `ipaddress, counter, gauge, timeticks, opaque`



SMI

	SMIv1	SMIv2
SIMPLE TYPES:	INTEGER	INTEGER
	OCTET STRING	OCTET STRING
	OBJECT IDENTIFIER	OBJECT IDENTIFIER
	-	Integer32
APPLICATION-WIDE TYPES:	-	Unsigned32
	Gauge	Gauge32
	Counter	Counter32
	-	Counter64
	TimeTicks	TimeTicks
	IpAddress	IpAddress
	Opaque	Opaque
	NetworkAddress	-
PSEUDO TYPES:	-	BITS

SMI

- Definición de nuevos objetos:
  - ASN.1 da la sintaxis base para estas definiciones.
  - SMI define los tipos básicos que se pueden utilizar.
  - SMI define un conjunto de MACROS que definen el marco a través de las cuales se definen los nuevos objetos.
  - Todas las definiciones de objetos de MIB deben cumplir con las definiciones de la SMI.

SMI

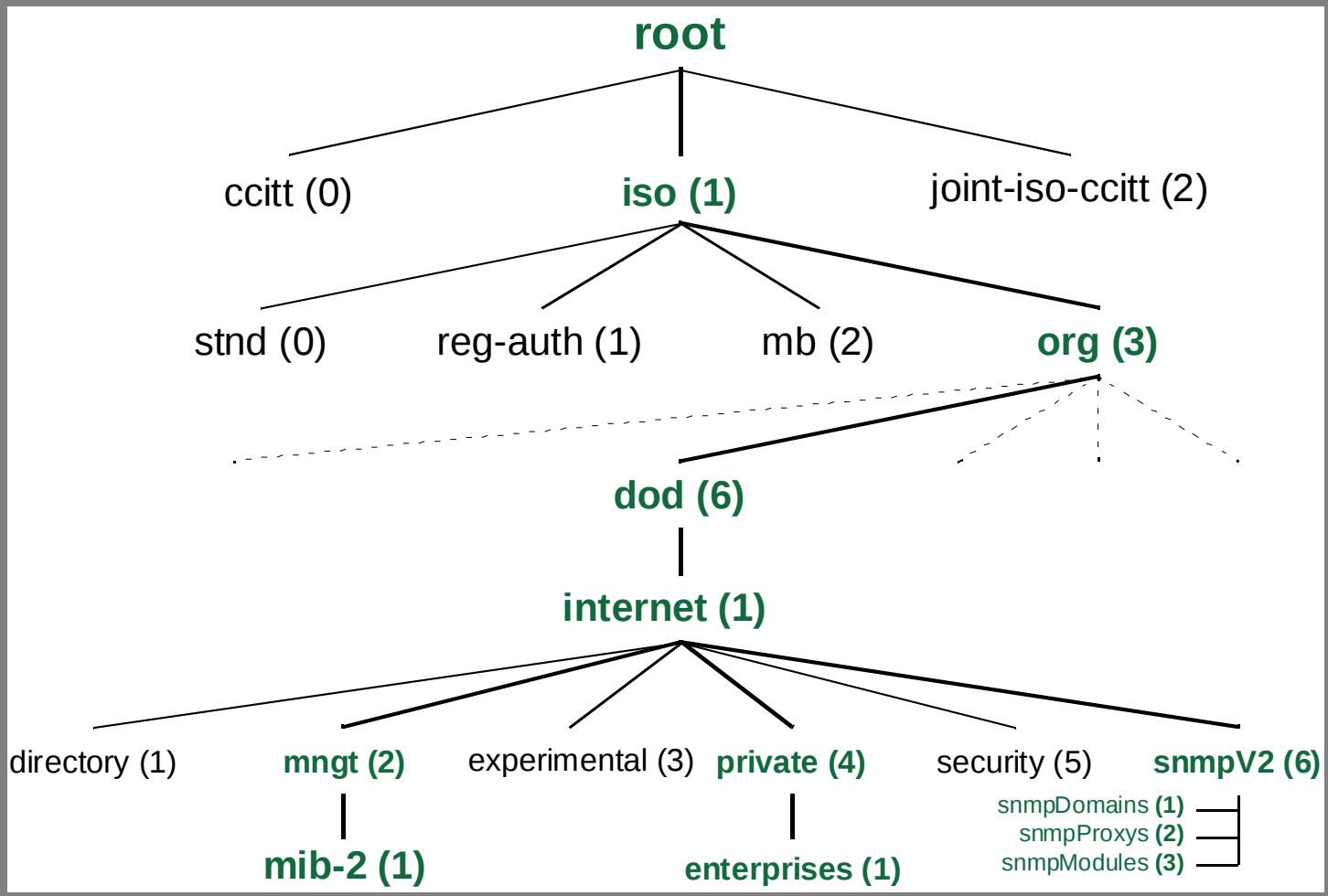
- A través de la SMI tal como esta definida, SNMP solo soporta dos estructuras posibles para un objeto:
  - Objetos escalares
    - Memoria libre, tiempo desde último reboot, etc.
  - Tablas bidimensionales de escalares
    - Ejemplo: tabla de enrutamiento

SMI

- Estructura de árbol de la MIB:
  - Las definiciones ASN.1 de los objetos de gestión de la SMI imponen la estructura de árbol sobre la MIB.
  - Todo objeto de gestión creado debe estar acompañado entre otras cosas por un OBJECT-IDENTIFIER.
    - Es una secuencia de enteros que define el “camino de árbol”.

SMI

- Estructura de árbol de la MIB:



SMI

- Macro **`OBJECT-TYPE`** :
  - Es la principal macro definida por SMI, la utilizamos para definir nuevos objetos.
  - Restringe el ASN.1 que se puede utilizar.
  - Apoya la estructura de árbol de la MIB.

SMI

- Macro **`OBJECT-TYPE`**:
  - `SYNTAX`:
    - El tipo abstracto del objeto. Debe ser una instancia del tipo ObjectSyntax (RFC 1155)
    - `ObjectSyntax` define un subconjunto de los tipos universales de ASN.1 más los tipos de aplicación que ya hemos visto.
  - ACCESS:
    - Define la forma en que se puede acceder a un objeto:
    - `read-write, read-only, write-only, read-write, no-accessible`

SMI

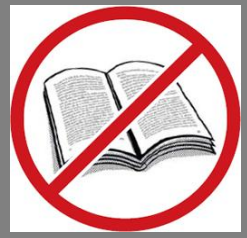
- Macro **`OBJECT-TYPE`**:
  - `STATUS`:
    - `current, mandatory, optional, deprecated`
  - `DescrPart, ReferPart` : referencias textuales a otros objetos, opcionales.
  - `IndexPart` : opcional, utilizado en la definición de tablas. Solamente esta presente si el objeto es parte de una “fila” de la tabla.

SMI

- Macro **OBJECT-TYPE**, ejemplo:

```
-- Definición de un objeto “address”
address          OBJECT-TYPE
SYNTAX           IpAddress
MAX-ACCESS       read-write
STATUS           current
DESCRIPTION      "The Internet address of this system"
::= {NEW-MIB  1}
```

SMI



- Definición de una MIB:
  - Como definir una MIB?
  - Tarea práctica que vamos a implementar!
  - Pasos a ver :
    - Estructura general
    - Definición de Objetos “no-hoja”
    - Definición de Objetos Escalares
    - Definición de Tablas

SMI

- Estructura General

```
NEW-MIB DEFINITIONS ::=
   BEGIN

      IMPORTS <defs de otras MIBs y SMI>


      <definicion de objetos hojas y no-hojas>



   END
```

SMI

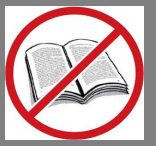
- Estructura general

```
FING-MIB DEFINITIONS ::=

BEGIN

IMPORTS

    Counter FROM RFC1155-SMI

    OBJECT-TYPE, enterprises FROM SNMPv2-SMI;

        < -- definiciones de objetos -- >

END
```

SMI

- Estructura general :
  - Para toda nueva MIB de SNMP versión 1 que construyamos importaremos al menos :
    - Algunos tipos de datos de RFC-1155
    - La macro `OBJECT TYPE` de RFC-1212
  - El nombre del archivo o el modulo que contiene a las definiciones anteriores puede llamarse diferente de un software de gestión a otro.

SMI



- Definición de objetos “no-hoja”

```
Name  OBJECT IDENTIFIER ::= {...}
```

- Ejemplo :

```
EXAMPLE:
info  OBJECT IDENTIFIER ::= {NEW-MIB  2}
```

SMI

- Escalares:

```
address OBJECT-TYPE
SYNTAX          IpAddress
MAX-ACCESS      read-write
STATUS          current
DESCRIPTION     "The Internet address of this system"
::= {NEW-MIB  1}
```

SMI

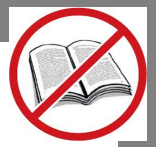
- Tablas :
  - La definición de objetos tabulares se hace de manera similar pero hay que respetar algunas convenciones :
    - SNMP solo puede devolver objetos escalares.
    - La tabla será una secuencia de filas.
    - Cada fila será una secuencia de campos.

SMI

- Tablas:
  - `myTable ::= SEQUENCE OF myEntry`
    - define la posicion de la tabla en el arbol
    - debe ser “not-accesible”
  - `myEntry ::= SEQUENCE OF MyEntry`
    - define cada una de las filas de la tabla
    - debe ser “not-accesible”
    - MyEntry luego es una secuencia de tipos elementales.

SMI

- Tablas:



```
routeTable      OBJECT-TYPE
SYNTAX          SEQUENCE OF RouteEntry
MAX-ACCESS      not-accessible
STATUS          current
DESCRIPTION     "This entity's routing table"
::= {NEW-MIB  3}
```

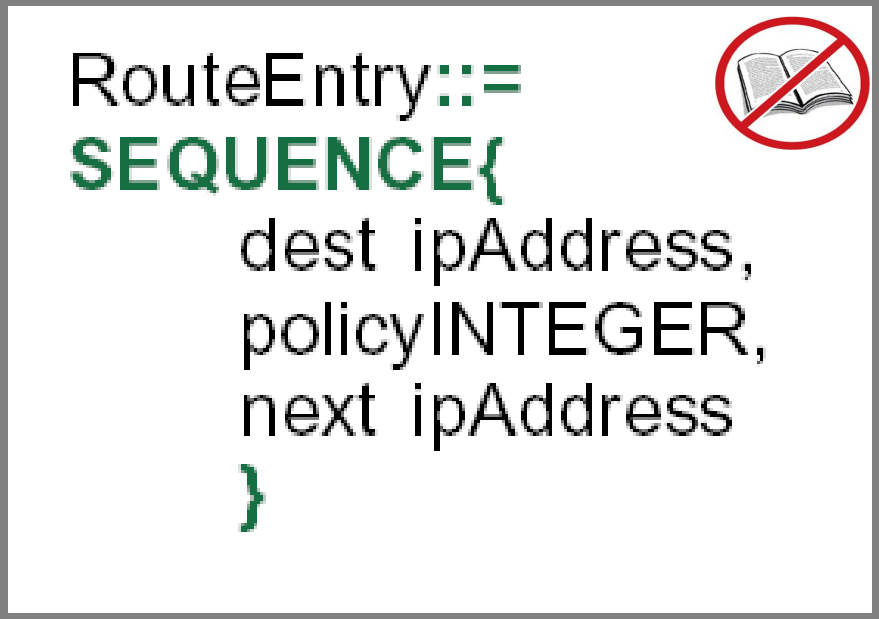
SMI

- Tablas :

```
routeEntry      OBJECT-TYPE
SYNTAX          RouteEntry
MAX-ACCESS      not-accessible
STATUS          current
DESCRIPTION     "A route to a particular destination"
INDEX           {dest, policy}
::= {routeTable  1}
```

SMI

- Tablas :


RouteEntry::=
SEQUENCE{
dest ipAddress,
policyINTEGER,
next ipAddress
}

SMI

- Tablas:

```
dest OBJECT-TYPE
SYNTAX      ipAddress
ACCESS      read-only
STATUS      current
DESCRIPTION"The address of a particular destination"
::= {route-entry  1}

policy OBJECT-TYPE
SYNTAX      INTEGER {
             costs(1)  -- lowest delay
             reliability(2) }  -- highest reliability
ACCESS      read-only
STATUS      current
DESCRIPTION"The routing policy to reach that destination"
::= {route-entry  2}
```

SMI

- Tablas:

```
next OBJECT-TYPE
SYNTAX      ipAddress
ACCESS      read-write
STATUS      current
DESCRIPTION"The internet address of the next hop"
::= {route-entry  3}
```

SMI

- Ejemplo de MIB (super simple!):

```
FING-MIB DEFINITIONS ::= BEGIN
IMPORTS
    Counter FROM RFC1155-SMI
    OBJECT-TYPE, enterprises FROM SNMPv2-SMI;

    fing        OBJECT IDENTIFIER ::= { enterprises 10526 }
    fingMib     OBJECT IDENTIFIER ::= { fing 1 }
```

SMI

- Ejemplo de MIB:

```
nProcs     OBJECT-TYPE
      SYNTAX INTEGER
      MAX-ACCESS read-only
      STATUS current
      DESCRIPTION “MIB de prueba"
      ::= { fingMib 1 }
END
```