# Lecture 6:
# Abstract Syntax Notation One
# ASN.1

Prof. Shervin Shirmohammadi
SITE, University of Ottawa



## Abstract Syntax Notation One

- Both the information and communications models need to be specified syntactically and semantically.
- This requires a language that specifies the management protocol in the application layer.
- This is where Abstract Syntax Notation One (ASN.1) plays a role.
- ASN.1 is actually more than a syntax; it's a language.
- Addresses both syntax and semantics
- Two type of syntax
  - Abstract syntax: set of rules that specify data type and structure for information storage
  - Transfer syntax: set of rules for communicating information between systems
- Makes application layer protocols independent of lower layer protocols.
- Can generate machine-readable code: Basic Encoding Rules (BER) is used in management modules.
- It is based on the Backus-Nauer Form (BNF)

# Backus-Nauer Form (BNF)

- BNF constructs are developed from primitives.
  - *<name> ::= <definition>*
  - <name> is "entity"
  - ::= "defined as"
  - <definition> is "primitive"
- Example: Simple Arithmetic Expression entity (<SAE>) is constructed from the primitives <digit> and <op>
  - <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  - <number> ::= <digit> | <digit><number>
  - <op> ::= + | - | x | /
  - <SAE> ::= <number>|<SAE>|<SAE><op><SAE>
- Example:
  - 9 is primitive 9
  - 19 is construct of 1 and 9
  - 619 is construct of 6 and 19

Simple Arithmetic Expression
<SAE> ::= <SAE><op><SAE>
Example: 13 x 2



# Type and Value

- The format of each line is defined as an assignment
  - <BooleanType> ::= BOOLEAN
  - <BooleanValue> ::= TRUE | FALSE
- ASN.1 module is a group of assignments; for example:

```
person-name Person-Name::=
        {
        first      "John",
        middle   "I",
        last     "Smith"
        }
```

# Data Type: Example 1

- Module name starts with capital letters
- Data types:
  - Primitives: NULL, GraphicString
  - Constructs
    - Alternatives : CHOICE
    - List maker: SET, SEQUENCE
    - Repetition: SET OF, SEQUENCE OF:

```
PersonnelRecord ::= SET
        {       Name,
         title      GraphicString,
         division   CHOICE {
            marketing      [0]  SEQUENCE
              {Sector,
               Country},
            research       [1]  CHOICE
              {product-based   [0]  NULL,
               basic           [1]  NULL},
            production     [2]  SEQUENCE
              {Product-line,
               Country }            }
etc.
```

**Figure 3.13 ASN.1 Data Type Definition Example 1**



# Data Type: Example 2

A list of invoices.

```
Trade-message ::= SEQUENCE
    {invoice-no        INTEGER
     name              GraphicString,
     details       SEQUENCE OF
                       SEQUENCE
                {part-no      INTEGER
                 quantity     INTEGER},
     charge        REAL,
     authenticator     Security-Type}

Security-Type ::= SET
    {   …
        …
        …  }
```

**Figure 3.14 ASN.1 Data Type Definition Example 2**

# ASN.1 Symbols

| Symbol | Meaning |
|---|---|
| ::= | Defined as |
| \| | or, alternative, options of a list |
| - | Signed number |
| -- | Following the symbol are comments |
| {} | Start and end of a list |
| [] | Start and end of a tag |
| () | Start and end of subtype |
| .. | Range |



# MIB Definition Example

The terms DEFINITIONS, BEGIN, and END, are keywords. This statement means that the RFC1213-MIB Module is being defined

```
RFC1213-MIB DEFINITIONS ::= BEGIN
…
…
…
END
```

# ASN.1 Data Type Conventions

| Data Types | Convention | Example |
|---|---|---|
| Object name | Initial lowercase letter | sysDescr, etherStatsPkts |
| Application data type | Initial uppercase letter | Counter, IpAddress |
| Module | Initial uppercase letter | PersonnelRecord |
| Macro, MIB module | All uppercase letters | RMON-MIB |
| Keywords | All uppercase letters | INTEGER, BEGIN |



# Data Type: Structure & Tag

- We will now use the ASN.1 notation to define the various data types and apply them to describe objects of SMI and MIB.
- Data types are defined based on Structure and Tags
  - Structure defines how data type is built
  - Tag uniquely identifies the data type

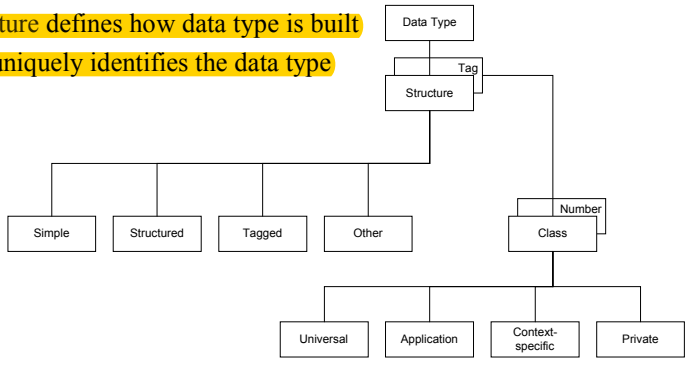



Figure 3.15 ASN.1 Data Type Structure and Tag

# Structure

- Simple (values are specified directly)
  - PageNumber ::= INTEGER
  - ChapterNumber ::= INTEGER
- Structure / Construct (Contains other types)
  - BookPageNumber ::= SEQUENCE
    {ChapterNumber, Separator, PageNumber}
    *Example: {1-1}*
  - BookPages ::= SEQUENCE OF {BookPageNumbers}
    *Example: {1-1, 2-3, 6-25}*
- Tagged (Used primarily for efficiency)
  - Derived from another type; given a new ID
- Other types: (Data type that is not pre-defined)
  - values chosen from CHOICE and ANY types



# Other Types

- CHOICE (No Data Types)

```
research Research ::= CHOICE
        {product-based   [0]      NULL,
         basic           [1]      NULL}
```

- CHOICE (Data Types)

```
research Research ::= CHOICE
        {product-based ProductType,
         basic     VisibleString}
        ProductType ::= VisibleString
```

- ANY (From the previous definitions)

```
Research ::= CHOICE
        {product-based ANY,
         basic     ANY}
```

# Example of Sequence Structure

- Two ways to define all the pages of a book as a sequence of page numbers making certain that they are in order.

```
BookPages ::= SEQUENCE OF { BookPageNumber}
                                or
BookPages ::=
        SEQUENCE OF
        {
        SEQUENCE
           {ChapterNumber, Separator, PageNumber}
        }
```



# Tag

- Tag uniquely identifies a data type. It is required for encoding the data types for communication.
- Every data type except CHOICE and ANY have data tags associated with them.
- Comprised of a class and tag number.
- Four Class types:
  - Universal – is the most common, like global variables in a software program.
  - Application - only in the application used, override universal.
  - Context-specific - specific context in application.
  - Private - used extensively by commercial vendors .

| Tag | Type Name | Set of Values |
|---|---|---|
| Universal 1 | BOOLEAN | TRUE or FALSE |
| Universal 2 | INTEGER | 0, Positive and negative numbers |
| Universal 3 | BIT STRING | A string of binary digits or null set |
| Universal 4 | OCTET STRING | A string of octets or null set |
| Universal 5 | NULL | Null, single valued |
| Universal 6 | OBJECT IDENTIFIER | Set of values associated with the object |
| Universal 7 | Object description | Human readable text describing the object |
| Universal 8 | EXTERNAL | The type is external to the standard |
| Universal 9 | REAL | Real numbers, expressed in scientific notation Mantissa × base$^{exponent}$ |
| Universal 10 | ENUMERATED | Specified list of integers |
| Universal 11 | ENCRYPTED | Encrypted information |
| Universal 12–15 | Reserved for future use | |
| Universal 16 | SEQUENCE and SEQUENCE OF | Ordered list of types |
| Universal 17 | SET and SET OF | Unordered list of types |
| Universal 18 | NumericString | Digits 0–9, space |
| Universal 19 | PrintableString | Printable characters |
| Universal 20 | TeletexString | Character set specified by CCITT Recommendation T.61 |



# Context Specific Tag Example

```
PersonnelRecord ::= SET
        {       Name,
         title          GraphicString,
         division   CHOICE
            marketing           [0]  SEQUENCE
              {Sector,
               Country},
           research         [1]  CHOICE
              {product-based  [0]  NULL,
               basic          [1]  NULL},
           production       [2]  SEQUENCE
              {Product-line,
               Country    }            }
```

**Figure 3.13 ASN.1 Data Type Definition Example 1**

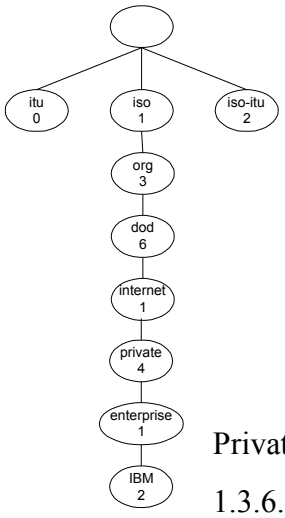
# ASN.1 Module SNMP MIB Example

Address translation table.

```
IpNetMediaEntry ::=SEQUENCE{
        ipNetToMediaIfIndex             INTEGER,
        ipNetToMediaPhysAddress         PhysAddress,
        ipNetToMediaNetAddress          IpAddress,
        ipNetToMediaType                INTEGER}
```



# Object Names and SMI Object Tree




- In a MIB there is an identifier for each occurrence of an object.

internet OBJECT IDENTIFIER ::=
{ISO(1) ORG(3) DOD(6) INTERNET(1)}

Private type identifier for IBM

1.3.6.1.4.1.2

# Walkthrough Example

Description of a Personal Record

| | |
|---|---|
| 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 |



# Walkthrough Example (…)

ASN.1 description of the record structure

```
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
```

# Walkthrough Example (…)

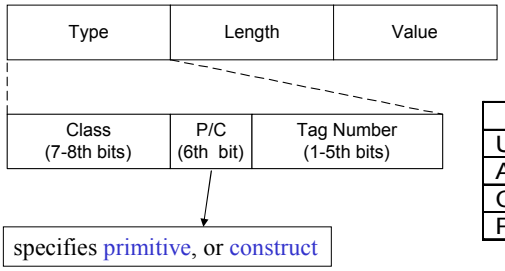
ASN.1 Description of a Record Value

{{ givenName “John”, initial “T”, familyName “Smith”},
  title “Director”
  number  51
  dateOfHire  “19710917”
  nameOfSpouse { givenName “Mary”, initial “T”, familyName “Smith”},
  children
  {{{ givenName “Ralph”, initial “T”, familyName “Smith”},
      dateOfBirth “19571111”},
  {{ givenName “Susan”, initial “B”, familyName “Jones”},
      dateOfBirth “19590717”}}}



# TLV Encoding

- In ASN.1 ASCII text data is encoded into a bit-oriented data representation called TLV (Type, Length, and Value).
- TLV type, length, and value are components of the structure.




| Class | 8th bit | 7th bit |
|---|---|---|
| Universal | 0 | 0 |
| Application | 0 | 1 |
| Context-specific | 1 | 0 |
| Private | 1 | 1 |

# ASN.1 Macro

- ASN.1 allows extensions to define new data types and values through a Macro definition.
- They also facilitate grouping of instances of an object.

```
<macroname> MACRO ::=
BEGIN
    TYPE NOTATION ::= <syntaxOfNewType>
    VALUE NOTATION ::= <syntaxOfNewValue>
    <auxiliaryAssignments>
END
```



# Macro Example

```
From ITU-T X.219:
ERROR  MACRO  ::=
   BEGIN
    TYPE NOTATION    ::=   Parameter
    VALUE NOTATION   ::=   value (VALUE CHOICE
                        { localValue   INTEGER,
                          globalValue  OBJECT IDENTIFIER
                        })
    Parameter        ::=   ``PARAMETER" NamedType | empty
    NamedType        ::=   identifier type  |  type
   END

Usage:

BadQueueName   ERROR
               PARAMETER   QueueName
                 ::= 0
```