Abstract Syntax Notation One
ASN.1

Dr. Andreas Steffen
©2000-2002 Zürcher Hochschule Winterthur

ASN.1 – Abstract Syntax Notation One
• Standards and applications using ASN.1
• Abstract syntax and transfer syntax
• Simple types – basic types, character string types
• Date and time types
• Type definitions
• Subtype definitions
• Value Assignments
• Object identifier type (OID)
• Structured types – sequence, set
• Context-specific tags

BER – Basic Encoding Rules
• Type–length–value rule
• Type tags
• Encoding of identifier field
• Encoding of tag classes
• Encoding of tag numbers
• Encoding of length field
• Encoding of integers
• Encoding of sequences
• Encoding of object identifiers
• Alternative encoding rules – PER, DER, CER
Standards and Applications using ASN.1

- X.400 Message Handling System / X.500 Directory Services
  - LDAP based Directories / X.509 Digital Certificates
- RSA Public Key Cryptography Standards
  - Storage and Transmission of Keys / Certificates (PKCS #12)
- Secure Electronic Transaction (SET)
  - Visa / Mastercard
- H.323 / T.120 Multimedia Communication Standards
  - Definition and Exchange of Multimedia Terminal Capabilities
- Telecommunications Management Network (TMN)
  - Performance / Fault and Configuration Management Information
- Unicode Worldwide Character Standard
  - Universal 16-bit Character Set comprising all written languages
- Simple Network Transfer Protocol (SNMP)
  - Management Information Base (MIB) / SNMP PDUs

Abstract Syntax

- An abstract syntax provides a set of formal rules for describing the structure of objects independent of both the user presentation and the machine-specific encoding techniques used for storage and transmission of the objects.

Transfer Syntax

- The transfer syntax is the actual representation of data as it is transmitted over a network. Often data is represented by a stream of octets.

Encoding Rules

- Encoding rules define the mapping of the abstract syntax used to represent data objects into the transfer syntax used to transmit the objects over a physical transmission channel.
Abstract Syntax Notation One

- ASN.1 is especially well suited for the representation of the complex, variable and extensible data structures used in modern communications applications.

Basic Encoding Rules of ASN.1

- BER encodes ASN.1 objects into a series of octet strings that can be transmitted over an octet oriented communications channel
ASN.1 Simple Types I

- **Basic Types**
  - BOOLEAN
  - INTEGER
  - ENUMERATED
  - REAL
  - BIT STRING
  - OCTET STRING

- **Character String Types (various subsets of ISO 10646-1)**
  - NumericString (0-9,<space>)
  - PrintableString (0-9,A-Z,a-z,<space>,<special>)
  - VisibleString
  - GraphicString
  - TeletexString
  - UTF8String
  - IA5String
## ASN.1 Simple Types II

### Object Types
- OBJECT IDENTIFIER
- ObjectDescriptor

### Miscellaneous Types
- NULL
- UTCTime
  - yymmdd hhmm[ss] <local offset from UTC>
- GeneralizedTime
  - yyyyymmdd hhmm[ss] <local offset from UTC>

### NULL
- The NULL type denotes a null value.

### OBJECT IDENTIFIER
- The OBJECT IDENTIFIER type denotes an object identifier, a sequence of integer components that identifies an object such as an algorithm, an attribute type, or perhaps a registration authority that defines other object identifiers. An OBJECT IDENTIFIER value can have any number of components, and components can generally have any nonnegative value.

### UTCTime
- yy = 50..99 : 1950-1999
- yy = 00..49 : 2000-2049

### GeneralizedTime
- Mandatory starting with the year 2050
ASN.1 Type Definitions

Syntax: `<type name> ::= <type>`

Examples: Counter ::= INTEGER

IpAddress ::= OCTET STRING

Months ::= ENUMERATED {january (1),
february (2),
march (3),
april (4),
may (5),
june (6),
july (7),
august (8),
september (9),
october (10),
november (11),
december (12)}
ASN.1 Subtype Definitions

Syntax:  <subtype name> ::= <type> ( <constraint> )

Examples:

Counter  ::= INTEGER ( 0..4294967295 )
IpAddress ::= OCTET STRING ( SIZE(4) )
Spring    ::= Months ( march | april | may )
Summer    ::= Months ( june | july | august )
SmallPrime ::= INTEGER ( 2 | 3 | 5 | 7 | 11 )
ExportKey ::= BIT STRING ( SIZE(40) )
ASN.1 Value Assignments I

Syntax: `<value name> <type> ::= <value>`

Examples:

- `ipInReceives Counter ::= 2450`
- `ipRouteMask IpAddress ::= 'FFFFFF00'H`
- `currentMonth Months ::= february`
- `currentTime UTCTime ::= "000204075015+0100"`
- `givenName VisibleString ::= "Andreas"`
- `married BOOLEAN ::= TRUE`
- `faxMessage BIT STRING ::= '01100001101'B`
- `encryptionKey ExportKey ::= 'A1B2C3D4E5'H`
Hierarchical Tree Structure

- ASN.1 object identifiers are organized in a hierarchical tree structure to make it possible to give any object a unique global identifier.

SNMP Objects

- The official SNMP objects defined by various RFCs are attached below the node

  \{ iso(1) org(3) dod(6) internet(2) mgmt(1) \}

- Enterprise specific SNMP objects are attached below the node

  \{ iso(1) org(3) dod(6) internet(2) private(4) enterprise(1) \}

SNMP OIDs of some well-known enterprises

- IBM: \{ enterprises 2 \}
- Cisco: \{ enterprises 9 \}
- Hewlett-Packard: \{ enterprises 11 \}
- Sun Microsystems: \{ enterprises 42 \}
- Microsoft: \{ enterprises 311 \}
- Intel: \{ enterprises 343 \}

Security OIDs of some well-known enterprises

- Sun Microsystems: \{ iso(1) member-body(2) US(840) 113536 \}
- RSA Data Security Inc: \{ iso(1) member-body(2) US(840) 113549 \}
- Microsoft: \{ iso(1) member-body(2) US(840) 113556 \}
- Netscape: \{ joint-iso-ccitt (2) country (16) usa (840) org (1) 113730 \}
- Verisign: \{ joint-iso-ccitt (2) country (16) usa (840) org (1) 113733 \}
- Intel: \{ joint-iso-ccitt (2) country (16) usa (840) org (1) 113741 \}
Use: Collection of a moderate number of variables that may be of different type and whose order is significant.

Type Definition:  

```plaintext
UserAccount ::= SEQUENCE {
    username    VisibleString,
    password    VisibleString,
    accountNr   INTEGER
}
```

Value Assignment:  

```plaintext
myAccount UserAccount ::= {
    username    "steffen",
    password    "jane51",
    accountNr   4711
}
```
**USE:** Collection of a large number of variables of the same type and whose order is significant.

**Type Definition:**

MemberCountries ::= SEQUENCE OF VisibleString

AccountRegistry ::= SEQUENCE OF UserAccount

**Value Assignment:**

euMembers MemberCountries ::= {
    "Austria", "Belgium", "Denmark",
    "Finland", "France", "Germany",
    "Greece", "Ireland", "Italy",
    "Luxembourg", "The Netherlands",
    "Portugal", "Spain", "Sweden",
    "United Kingdom"}
ASN.1 Structured Types III

SET

**Use:** Collection of a moderate number of variables that may be of different type and whose order is insignificant.

**Type Definition:**

\[
\text{UserAccount} ::= \text{SET} \{ \\
\quad \text{username} \ [0] \ \text{VisibleString}, \\
\quad \text{password} \ [1] \ \text{VisibleString}, \\
\quad \text{accountNr} \ [2] \ \text{INTEGER}
\}
\]

**Value Assignment:**

\[
\text{myAccount} \ \text{UserAccount} ::= \{ \\
\quad \text{accountNr} \ 4711, \\
\quad \text{username} \ "steffen", \\
\quad \text{password} \ "jane51"
\}
\]

**Context-Specific Tags**

- Since they can occur in arbitrary order, the members of a set need tags in order to uniquely identify them. Tags can either be defined explicitly by the system designer or are assigned automatically by the ASN.1 parser if the automatic tagging option is used.
Use: Collection of variables that are the same type and whose order is insignificant.

Type Definition: Keywords ::= SET OF VisibleString

Value Assignment:

someASN1Keywords Keywords ::= 
  {"INTEGER", "BOOLEAN", "REAL"}
Basic Encoding Rules of ASN.1
BER
General Encoding Rule for ASN.1 Values
Type - Length - Value

1. Primitive, definite-length encoding
   - simple types
2. Constructed, definite-length encoding
   - structured types (SEQUENCE[OF], SET[OF])
3. Constructed, indefinite-length encoding
   - structured types (SEQUENCE[OF], SET[OF])
### Universal Class Tags

<table>
<thead>
<tr>
<th>Universal Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>2</td>
<td>INTEGER</td>
</tr>
<tr>
<td>3</td>
<td>BIT STRING</td>
</tr>
<tr>
<td>4</td>
<td>OCTET STRING</td>
</tr>
<tr>
<td>9</td>
<td>REAL</td>
</tr>
<tr>
<td>10</td>
<td>ENUMERATED</td>
</tr>
<tr>
<td>6</td>
<td>OBJECT IDENTIFIER</td>
</tr>
<tr>
<td>7</td>
<td>ObjectDescriptor</td>
</tr>
<tr>
<td>26</td>
<td>VisibleString</td>
</tr>
<tr>
<td>5</td>
<td>NULL</td>
</tr>
<tr>
<td>23</td>
<td>UTCTime</td>
</tr>
<tr>
<td>24</td>
<td>GeneralizedTime</td>
</tr>
<tr>
<td>16</td>
<td>SEQUENCE [OF]</td>
</tr>
<tr>
<td>17</td>
<td>SET [OF]</td>
</tr>
</tbody>
</table>

...
ASN.1 Type Tags II
Application Class Tags

**Explicit Tagging:** (strong type checking)

```
IpAddress ::= [APPLICATION 0]
            OCTET STRING (SIZE(4))

Counter ::= [APPLICATION 1]
            INTEGER (0..4294967295)
```

**Implicit Tagging:** (shorter encoding)

```
IpAddress ::= [APPLICATION 0] IMPLICIT -- RFC 1155
            OCTET STRING (SIZE(4))

Counter ::= [APPLICATION 1] IMPLICIT -- RFC 1155
            INTEGER (0..4294967295)
```
### BER Encoding of Identifier Field I

**Tag numbers < 31**

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bits 8, 7, 6, 5**: Define the Class
- **Bits 4, 3, 2, 1**: Define the P/C (Primitive/Constructed)

- **0 0**: Universal
- **0 1**: Application
- **1 0**: Context-specific
- **1 1**: Private

---

A. Steffen, 22.01.2002, KSy_ASN1.ppt 19
BER Encoding of Identifier Field II
Tag numbers $\geq 31$

Leading octet 2nd octet Last octet
Class P/C 1 1 1 1 1 1 0

+ ... + + = Tag number
### BER Encoding of Length Field

- **Short definite form (L < 128 octets)**
  - One octet: Length L
  - L octets: Contents field

- **Long definite form (128 ≤ L < 2^{1008} octets)**
  - First octet: K octets:
  - K octets: Length L
  - L octets: Contents field

- **Indefinite form; content field terminated by EOC**
  - One octet: Length L
  - Two octets: Contents field
  - Two octets: EOC '0000'H

---

**Short definite form**
- Used either for **primitive** or **constructed** types with content lengths smaller than 128 octets.

**Long definite form**
- Used either for **primitive** or **constructed** types with content lengths usually equal or greater than 128 octets.

**Indefinite form**
- Used for **constructed** types only.
- The end-of-contents octets (EOC) can be considered as the encoding of a value whose tag is universal class, whose form is primitive, whose tag number is zero, and whose contents are absent, i.e. it has zero length.
BER Encoding Examples I

INTEGER

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>00</td>
<td>00010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>two's complement</td>
</tr>
</tbody>
</table>

DayOfYear ::= [APPLICATION 17] IMPLICIT INTEGER

today DayOfYear ::= 128

BER coding of two’s complement integers

-129: 1111 1111 0111 1111 = 02 02 FF 7F
-128: 1111 1111 1000 0000 = 02 01 80
-127: 1111 1111 1000 0001 = 02 01 81
-1: 1111 1111 1111 1111 = 02 01 FF
0: 0000 0000 0000 0000 = 02 00
1: 0000 0000 0000 0001 = 02 01 01
127: 0000 0000 0111 1111 = 02 01 7F
128: 0000 0000 1000 0000 = 02 02 00 80
129: 0000 0000 1000 0001 = 02 02 00 81
**BER Encoding Examples II**

**SEQUENCE**

```plaintext
Birthday ::= SEQUENCE {
  name VisibleString,
  day   DayOfYear
}
```

**Type Definition**

```plaintext
myBirthday Birthday ::= {
  name   "Jane",
  day    128
}
```

**Value Assignment**

<table>
<thead>
<tr>
<th>Birthday Length</th>
<th>Contents</th>
<th>BER Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0A</td>
<td></td>
</tr>
<tr>
<td>VisibleString</td>
<td>Length Contents</td>
<td>&quot;Jane&quot;</td>
</tr>
<tr>
<td>1A</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>DayOfYear</td>
<td>Length Contents</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>02</td>
<td>00 80</td>
</tr>
</tbody>
</table>
```
Coding of OID Root
- ccitt(0): \( Z = Y \quad \{ 0 ..39 \} \)
- iso(1): \( Z = 40 + Y \quad \{40 ..79 \} \)
- joint-isi-ccitt(2): \( Z = 80 + Y \quad \{80 ..119 \} \)

Coding of OID node numbers
- Similar coding as tag numbers in identifier field
- Range (0..127): 0xxxx xxxx
- Range (128..16383): 1xxxx xxxx 0xxxx xxxx
- Range (16384..2097151): 1xxxx xxxx 1xxxx xxxx 0xxxx xxxx
Alternative Encoding Rules

- **Packed Encoding Rules (PER)**
  - Very compressed encoding based on ASN.1 subtype information. Example: subtype `INTEGER (998..1001)` is encoded using two bits only. Used e.g. in H.323 multimediastandard.

- **Distinguished Encoding Rules (DER)**
  - Subset of BER that gives exactly one way to represent any ASN.1 value. Used e.g. in X.509 certificates where computation of digital hash sums must be unique. Uses definite-length encoding.

- **Canonical Encoding Rules (CER)**
  - Subset of BER that gives exactly one way to represent any ASN.1 value but based on indefinite-length encoding.