*:96 Overheads

Part 2b: Encoding using ASN.1

More about this course about Internet application protocols can be found at URL:

http://www.dsv.su.se/~jpalme/internet-course/Int-app-prot-kurs.html

Last update: 2005-09-20 13:49

ASN.1-s historia

(1) Courier, protokoll inom Xerox Corp.

(2) CCITT Recommendation X.409 1984

(3) ISO delar upp X.409 i två standarder, en (ISO 8824) för språket och en (ISO 8825) för BER.

(4) CCITT ger ut dessa 1988 som X.208 och X.209.

(5) Ny version 1994, innehåller bl.a. ny notation som ersättning för macros.

Några Internet standarder som använder ASN.1

Kerberos — Security system (Authentication, etc.) [RFC 1510]

LDAP — Lightweight Directory Access Protocol [RFC 1777]

SNMP — Simple Network Management Protocol [RFC 1303]

SMIME — Security Enhanced MIME

ASN.1 versus ABNF

Similarities

1. Both are languages for the specification of the syntax of encoded data transmitted between computers in net-based protocols.
2. Both are based on the BNF (Backus-Naur-Form) form syntax specifications, which was first used in the Algol-60 specification.

Differences

1. ABNF specifies encoding of information into text strings, ASN.1 specified encodings of information into usually binary form (octet strings). Because of this difference, ABNF encodings are easier for a human to read.
2. ASN.1 specifies a tag-length-value kind of encoding, which avoids many problems with delimiters and delimeter encoding in ABNF.

Nya datatyper definieras ur kända typer


```
Temperature ::=      REAL -- in degrees Kelvin
```

One-component data types

```
Temperature ::=      [APPLICATION 0] REAL -- in degrees Kelvin
WindVelocity ::=     [APPLICATION 1] REAL -- in m/s
Humidity ::=         [APPLICATION 2] REAL -- relative percentage
```

Two-component data types

```
ComplexNumber::= [APPLICATION 3] SEQUENCE
                 {       imaginaryPart       REAL,
                         realPart            REAL }
```

Använda tidigare definierade typer i nya typdefinitioner

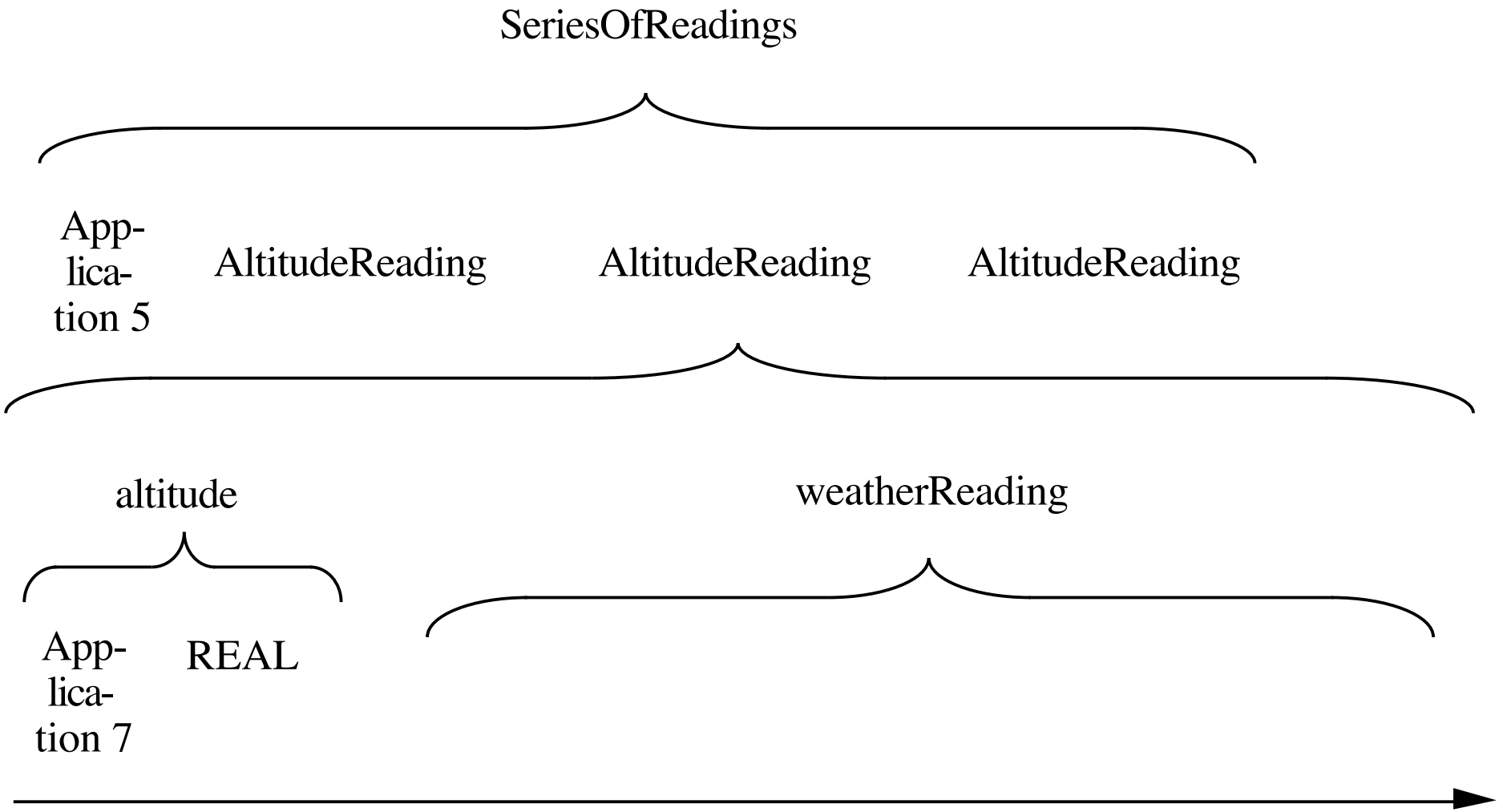
```
Temperature       ::= [APPLICATION 0] REAL - - in degrees Kelvin
WindVelocity      ::= [APPLICATION 1] REAL - - in m/s
Humidity          ::= [APPLICATION 2] REAL - - relative percentage

WeatherReading    ::= [APPLICATION 4] SEQUENCE
                  {     temperatureReading  Temperature,
                        velocityReading     WindVelocity,
                        humidityReading     Humidity }
```

Stor och liten bokstav är signifikant. Första bokstaven måste vara stor för datatyp, liten för datafält (se ovan).

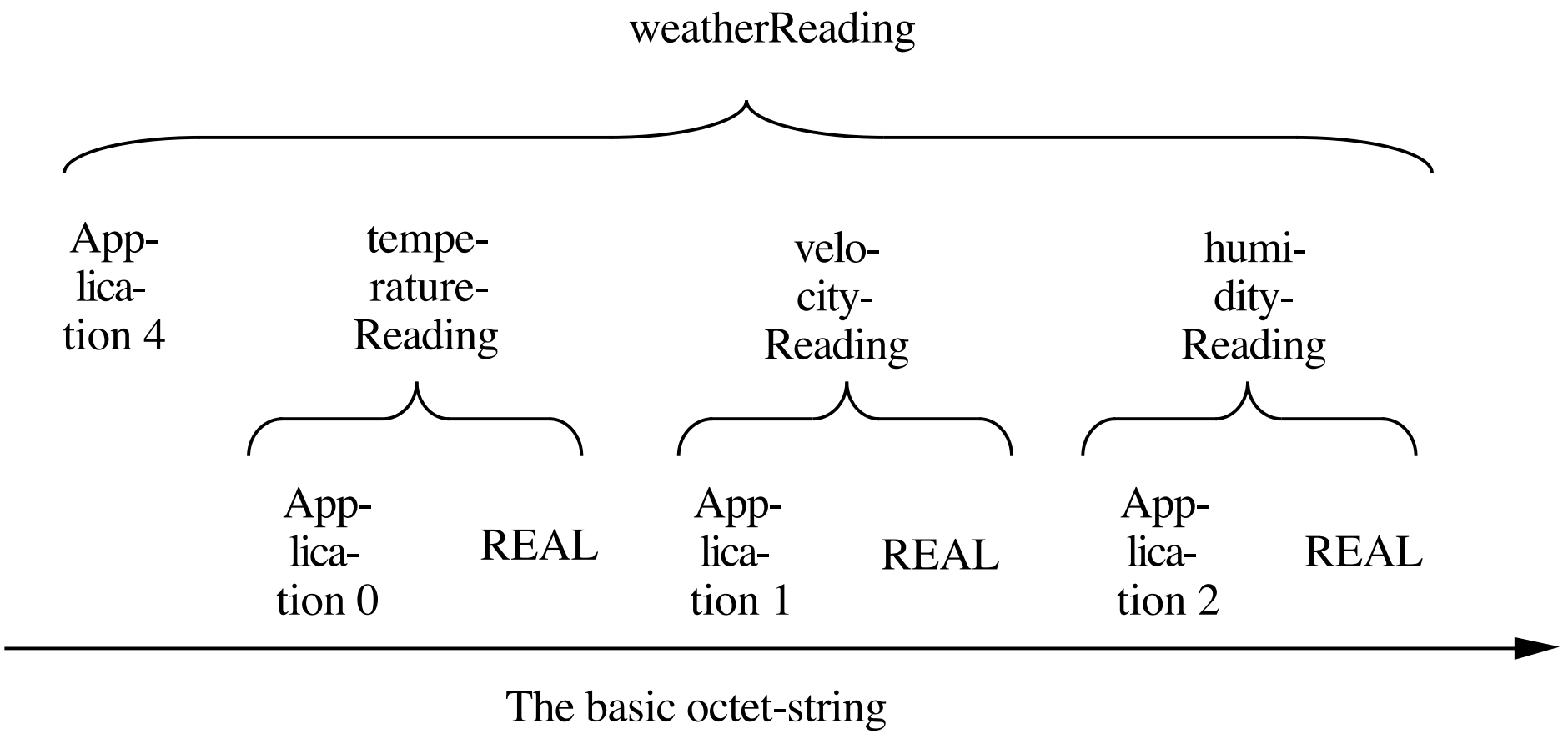
En sekvens av element av samma typ


```
Altitude            ::= [APPLICATION 7] REAL -- Meters
                                        -- above the sea

SeriesOfReadings    ::= [APPLICATION 5] SEQUENCE OF
                        AltitudeReading

AltitudeReading     ::= [APPLICATION 6] SEQUENCE
                        {       altitude            Altitude,
                                weatherReading WeatherReading }
```

Uppdelning av oktett(bit)-strömmen



Vidare uppdelning av oktettströmmen



Terminologi

En *typ* eller *datatyp* är en mängd av värden.

En typ kan definieras genom att räkna upp alla tillåtna värden, eller definieras att ha obegränsat antal värden som t.ex. typerna *Integer* och *Real*.

En ny typ, som definieras som en kombination av element av tidigare definierade typer, kallas för en *strukturerad typ*.

Exempel på strukturerad typdefinition:

```
ComplexNumber::=    [APPLICATION 3] SEQUENCE
                {   imaginaryPart   REAL,
                    realPart        REAL }
```

Abstract och Transfer syntax

Dokumentformat	**Språk**	**Användning**
Notation	Abstract Syntax (ASN.1)	Specifikationer
Kodning	Transfer Syntax (ASN.1-BER)	Kommunikation

ASN.1 = Abstract Syntax Notation 1

BER = Basic Encoding Rules

ASN.1-produktioner

En ASN.1-produktion är en regel, som definierar en typ ur andra typer. Syntaxen för en ASN.1-produktion är

(1) Namnet på den nya typen (börjar med stor bokstav)

(2) Operatorn ”::=”

(3) Definitionen av den nya typen (fältnamn börjar med små bokstäver)

Exempel:

```
ComplexNumber::=    [APPLICATION 3] SEQUENCE
                 {  imaginaryPart   REAL,
                    realPart        REAL }
```

Moduler

<modulnamn> DEFINITIONS ::= BEGIN
<modulkropp>
END

Exempel:

EmptyModule DEFINITIONS ::= BEGIN
END

Typer i ASN.1:

Enkla typer	Teckensträngs typer	Strukturerade typer	”Useful types”
BOOLEAN	NumericString	SET	GeneralizedTime
INTEGER	PrintableString	SET OF	UTCTime
ENUMERATED	TeletexString	SEQUENCE	EXTERNAL
REAL	VideotexString	SEQUENCE OF	ObjectDescriptor
BIT STRING	VisibleString	CHOICE	
OCTET STRING	IA5String	ANY	*Warning: Constraints are strongly recommended for Graphic, General, Universal, BMP and UTF8 strings*
NULL	GraphicString	[Tagged]	
OBJECT IDENTIFIER	GeneralString	*<Different variants < of ISO 10646, not < in the 1998 < version*	
	UniversalString		
	BMPString		
	UTF8String		
	CharacterString		

Reserverade ord

BOOLEAN	**INTEGER**	**BIT**	**STRING**
OCTET	**NULL**	**SEQUENCE**	**OF**
SET	**IMPLICIT**	**CHOICE**	**ANY**
EXTERNAL	**OBJECT**	**IDENTIFIER**	**OPTIONAL**
DEFAULT	**COMPONENTS**	**UNIVERSAL**	**APPLICATION**
PRIVATE	**TRUE**	**FALSE**	**BEGIN**
END	**DEFINITIONS**	**EXPLICIT**	**ENUMERATED**
EXPORTS	**IMPORTS**	**REAL**	**INCLUDES**
MIN	**MAX**	**SIZE**	**FROM**
WITH	**COMPONENT**	**PRESENT**	**ABSENT**
DEFINED	**BY**	**PLUS-INFINITY**	
MINUS-INFINITY		**TAGS**	

Stora resp. små bokstäver är signifikanta, så ett enkelt sätt att undvika kollission med reserverade ord är att använda identifierare som innehåller små bokstäver helt eller delvis.

Identifierarformat

Teckenmängd	Typdefinition	Fält, värde
"a"-"z"	Kan ingå	Kan ingå, måste börjar med
"A"-"Z"	Kan ingå, måste börja med	Kan ingå
"0"-"9"	Kan ingå	Kan ingå
"-"	Kan ingå, men aldrig två i följd	Kan ingå, men aldrig två i följd

Kommentarer

Inleds med "--", avslutas med "--" eller vid radens slut

Integer — Simple Type

Värdemängd: Alla positiva och negativa heltal inklusive 0.
OBS: Ingen maximigräns!

Typnotation:

INTEGER
INTEGER { <ident> (<num>), ...}

Exempel:

```
INTEGER
{       touristClass (-1),
        businessClass(0),
        firstClass(1) }
```

Värdenotation:

<num>
<ident>

Exempel:

4711
-4294967296
0
firstClass

Subtyper: **Single value, Contained subtype, Range**

Subtyper

Notation:

<type> <subtype-spec>

där <**subtype-spec>** har formen **(<value-set> | ...)**

där <**value-set>** kan vara

- single value
- contained subtype

Bara för vissa typer även

- value range
- size range
- permitted alphabet
- inner subtyping

Subtyper till Integer I

Enkelt värde:

StandardBase ::= INTEGER (2 | 10)

Inkluderad subtype

ExtendedBase ::= INTEGER (INCLUDES StandardBase | 8 | 16)

Value Range (bara när ordning är definierad)

Positive ::= INTEGER (1 .. MAX)
Non-negative ::= INTEGER (0 .. MAX)
Number ::= INTEGER (0 .. <1000)

MAX och **MIN** betyder obegränsat (inte samma sak som $+\infty$ och $-\infty$, kan ej användas som värde, utan bara i Range-satser)!

Subtyper till Integer II

Value Range med namngivna delfält

DayOfTheMonth ::= INTEGER { first(1), last(31) } (first .. last)

Användande av definierade konstanter

CharacterPosition ::= INTEGER { first(1), last(lineLength) }
(first .. last)

lineLength INTEGER ::= 80

Examples of use of subtyping of Integer


```
Month-number ::= INTEGER (1 .. 12)
```

```
months-of-the-year ::= 12
Month-number ::= INTEGER (1 .. months-of-the-year)
```

```
Single-digit-prime ::= INTEGER ( 2 | 3 | 5 | 7 )
```

```
Positive-number ::= INTEGER (1 .. MAX)
Non-negative-number ::= INTEGER ( 0 | INCLUDES
Postive-number)
```

```
Date ::= SEQUENCE {
        year INTEGER
        month INTEGER (1 .. 12)
        day INTEGER (1 .. 31)
        }
```

Boolean — Simple Type

Värdemängd: TRUE och FALSE	
Typnotation: **BOOLEAN**	**Värdenotation:** **TRUE** **FALSE**
Subtyper: **Single value, Contained subtype**	

Anmärkning: Jag tycker det borde vara lagligt att skriva t.ex.

Sex ::= BOOLEAN {male (TRUE), female (FALSE) }

men det lär inte vara tillåtet.

Enumerated — Simple Type

Värdemängd: Varje uppräkning av skilda, namngivna värden

Typnotation:

ENUMERATED { <ident> (<num>), ...}

Exempel:

```
ENUMERATED
{       touristClass (-1),
        businessClass(0),
        firstClass(1)
}
```

Värdenotation:

<ident>

Exempel:

touristClass
firstClass

Subtyper: Single value, Contained subtype

Tre möjliga notationer för veckodag


```
DayOfTheWeek    ::= INTEGER { monday(1), tuesday(2),
                      wednesday(3), thursday(4), friday(5),
                      saturday(6), sunday(7) }
DayOfTheWeek    ::= INTEGER { monday(1), tuesday(2),
                      wednesday(3), thursday(4), friday(5),
                      saturday(6), sunday(7) } (1..7)
DayOfTheWeek    ::= ENUMERATED { monday(1), tuesday(2),
                      wednesday(3), thursday(4), friday(5),
                      saturday(6), sunday(7) }
```

I det översta fallet tillåts alla heltal, i den mittersta fallet tillåts bara de sju heltalsvärdena från 1 till 7.

Skillnad mellan mittersta och nedre fallet: Ordning definierad för INTEGER, inte för ENUMERATED. Detta innebär att jämförelser med < och > och range-subtyper inte tillåts för Enumerated. Jämför programmeringsspråket Pascal.

Real — Simple Type

Värdemängd: ±∞ och heltal som kan uttryckas på formen
M X B^E där Mantissan M kan vara godtycklig INTEGER,
Basen B kan vara 2 eller 10,
Exponenten E kan vara godtycklig INTEGER

Typnotation:	**Värdenotation:**
REAL	**{ <num>, <num>, <num> }** **0** **PLUS-INFINITY** **MINUS-INFINITY** **Exempel:** **{ 3141592653589793238462433832 79, 10, -30}**

Subtyper: **Single value, Contained subtype, Range**

Bit String — Simple Type

Värdemängd: Ordnad följd av 0 eller fler bitar

Typnotation:	**Värdenotation:**
BIT STRING **BIT STRING { <ident> (<num>), ...}**	**'<binära siffror>' B** **'>hexadecimala siffror>' H** **{ <identifierare>, ... }**
Exempel:	**Exempel:**
BIT STRING { **oddparity (0),** **enableparity (1),** **eightdatabits (2) }**	**'0001010' B** **'00FF' H** **{ oddparity, enableparity }**
Subtyper: Single value, Contained subtype, Size range	

Anmärkning: Kodas enligt BER mer kompakt än SEQUENCE of BOOLEAN, men ej mer kompakt vid Packed Encoding Rules.

Subtyper till BITSTRING

Utöver Single value och Contained subtype finns även Size range. Exempel:

```
BIT STRING (SIZE ( 0 | 2 .. 7 | 10 ) )
```

Exempel på BITSTRING

```
BitMappedPicture    ::= BIT STRING
Characteristics     ::= BIT STRING {male(0), adult(1), blueEyed(2),
                            caucasian(3) }
Characteristics     ::= BIT STRING {male(0), adult(1), blueEyed(2),
                            caucasian(3) } (SIZE (0 .. 4))
Characteristics     ::= BIT STRING {male(0), adult(1), blueEyed(2),
                            caucasian(3) } (SIZE (4))
```

Vad är skillnaden mellan de tre definitionerna av Characteristics ovan?

Jämförelse av Bit String och Enumerated

```
DayOfTheWeek  ::= ENUMERATED { monday(0),
                      tuesday(1), wednesday(2),
                      thursday(3), friday(4), saturday(5),
                      sunday(6) } }

DaysOpen      ::= BIT STRING { monday(0), tuesday(1),
                      wednesday(2), thursday(3),
                      friday(4), saturday(5), sunday(6) }
                      (SIZE(7))
```

Vad betyder ”monday” i de två fallen ovan?

Octet String — Simple Type

Värdemängd: Ordnad följd av 0 eller fler oktetter	
Typnotation: **OCTET STRING**	**Värdenotation:** **'<binära siffror>' B** **'>hexadecimala siffror>' H** **Exempel:** **'00001010' B** **'00FF' H**
Subtyper: Single value, Contained subtype, Size range	

Exempel på Octet String

```
PackedBCDString      ::= OCTET STRING
--  the digits 0 through 9, two digits per octet,
--  each digit encoded as 0000 to 1001,
--  1111 used for padding.

twelve  PackedBCDString::= '12'H
```

Null — Simple Type

Värdemängd: Ett enda värde: **null**	
Typnotation: **NULL**	**Värdenotation:** **NULL**
Subtyper: **Single value, Contained subtype**	
Exempel: **Order ::= SEQUENCE {** **ISBN VisibleString,** **Airmail NULL OPTIONAL }**	

Anmärkning: Kan användas för att markera plats för något som skall komma, eller när enbart existensen ger information, används sällan.

Exempel på användning av SIZE


```
MonthNumber ::= NumericString (SIZE (1 .. 2))
MonthNumber ::= NumericString (SIZE (1 | 2))
Base ::= BIT STRING (SIZE ( 0 | 2 .. 7 | 10 ) )
Couple ::= SET SIZE(2) OF Human
BridgeDeal ::= SET SIZE (13) OF PlayingCard
BridgeHand ::= SET SIZE (0..13) OF PlayingCard

lineLength INTEGER 80
Line ::= VisibleString (SIZE (0 .. lineLength)
```

Character String-typer

Värdemängd: En sträng av tecken ur ett visst alfabet

Typnotation:

NumericString
PrintableString
TeletexString **T61String**
VideotexString
VisibleString **ISO646String**
IA5String
GraphicString
GeneralString
UniversalString

Värdenotation:

"<sträng>"

Exempel:

"PS example"
"Alfvén"
"αβχδεφγηιφκλ"

Subtyper: Single value, Contained subtype, Size Range,
Permitted alphabet (finns bara för teckensträngar)

Alfabet för Character String-typerna

NumericString	'0' .. '9' och ' '
PrintableString	'a'..'z', 'A'..'Z', '0'..'9' ' ( ) + , - . / : = ?
TeletexString **T61String**	Se T.61, ca 400 tecken inklusive diakritiska tecken, t.ex. "ä" = "a with diaresises", klarar alla nationella varianter av latinska alfabetet
VideotexString	Se T.100 och T.101
VisibleString **ISO646String**	Tryckbara tecken plus blanksteg ur ISO 646 (”ascii”)
IA5String	IA5 (ISO 646, ”ascii”)
GraphicString	Alla hos ISO registrerade teckenmängder (G-mängderna) plus blanksteg. ISO 2022 escape sequences can switch between sets.
GeneralString	Alla hos ISO registrerade teckenmängder (G och C-mängderna) plus blanksteg och delete. ISO 2022 escape sequences can switch sets.
UniversalString	ISDO 10646 (Unicode)

Subtyp till Character String-typer

Permitted Alphabet, alla tillåtna tecken måste räknas upp, ingen ordning gäller.

Exempel:

PrintableString (FROM("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"))

Set — Structured Type

Värdemängd: Värdet är en samling av värden från ett antal komponent-typer. Samlingen har ett värde för varje obligatorisk komponent, och noll eller ett för varje valfri komponent

Typnotation:

SET { <component> , ... }

där **<component >**är någon av

- **<identifier> <type>**
- **<identifier> <type> OPTIONAL**
- **<identifier> <type> DEFAULT <value>**
- **COMPONENTS OF <type>**

Värdenotation:

{ identifier value,

. . .

}

Sequence — Structured Type

Värdemängd: varje värde är en samling av värden från ett antal komponent-typer. Samlingen måste vara ordnad, och har ett värde för varje obligatorisk komponent och noll eller ett värde för varje valfri komponent

Typnotation:	**Värdenotation:**
SEQUENCE { <component,> ...} där <**component**> kan ta samma värden som SET-typen	**{<identifier> <value>,** **. . .** **}**

Sequence (forts.)

Exempel:

```
SEQUENCE
{       day INTEGER,
        name IA5String
                OPTIONAL
}
```

Exempel:

```
{ day 12, name "Agneta" }
{ day 5 }
{ day -4711 }
{ 5, "Mary Anne" }
{ name "Jean" day 17 } - - Fel
- - ordningen måste stämma!
```

Subtyper: Single value, Contained subtype, Inner subtyping

Observera att komponentnamnen inte behöver anges i värdenotationen, se det andra exemplet ovan.

Set-of — Structured Type

Värdemängd: Varje värde är en oordnad mängd av värden av en viss känd typ

Typnotation:

SET OF <type>
SET <size-limit> OF <type>

Exempel:

```
SET OF Name
där
Name ::= SEQUENCE
{       GivenName IA5String,
        SurName IA5String }
```

Värdenotation:

{ <value>, . . . }

Exempel:

```
{       { "John", ”Green” },
        { "Mary", "Green" },
        { ”John”, ”Green” }
}
```

Subtyper: **Single value, Contained subtype, Size range, Inner subtyping**

Sequence of — Structured Type

Värdemängd: Varje värde är en ordnad mängd av värden av en viss känd typ

Typnotation:

```
SEQUENCE OF <type>
SEQUENCE <size-limit> OF <type>
```

Exempel:

```
SEQUENCE OF City
där City ::= SEQUENCE {
        name IA5String,
        longitude INTEGER,
        latitude INTEGER }
```

Värdenotation:

```
{ <value>, . . . }
```

Exempel:

```
{       { "Stockholm", 59 ,18 },
        { "London", 51, 0 },
        { "Berlin", 52, 13 },
        { "Stockholm", 59, 18 } }

{ }
```

Subtyper: Single value, Contained subtype, Size range, Inner subtyping

Choice — Structured Type

Värdemängd: Summan av värdena för alla komponenttyperna

Typnotation:

```
CHOICE
{ <ident> <type>,
. . .
}
```

Exempel:

```
CHOICE
{     arabicNumber  NumericString,
      romanNumber  PrintableString
      }
```

Värdenotation:

(1988 års version)

```
<ident> <value>
```

(1992 års version)

```
<ident> : <value>
```

Exempel:

```
arabicNumber  "6" --  1988
romanNumber "VI" --  1988
romanNumber : "VI" --  1992
```

Subtyper: Single value, Contained subtype, Inner subtyping

Any — Structured Type

Värdemängd: Alla värden hos alla typer

Typnotation:

ANY

ANY DEFINED BY <identifier>

<identifier> kan vara t.ex. ett heltal eller en objektidentifierare

Exempel:

```
SEQUENCE
{   type INTEGER,
    value ANY DEFINED BY type  }
```

Värdenotation:

(1988) <type> <value>

(1992) <type> : <value>

Exempel (1988 års notation):

```
BOOLEAN TRUE
BIT STRING '101' B
```

Subtyper: **Single value, Contained subtype**

Kan typen härledas ur sammanhanget?


```
Name        ::= SEQUENCE
            {   givenName       [0]  VisibleString OPTIONAL,
                surName         [1]  VisibleString OPTIONAL }

Name        ::= SET
            {   givenName       [0]  VisibleString,
                surName         [1]  VisibleString }

Name        ::= CHOICE
            {   numericName     NumericString,
                alphabeticName  VisibleString }
```

Alla alternativ måste ha olika typer för:

- Komponenter i ett SET
- Komponenter i en SEQUENCE med OPTIONAL
- Komponenter i en CHOICE

Om bastyper inte är olika, kan de göras olika med etiketter (tags)

OBS att man ändå kan ta bort de två förekomsterna av **[0]** i exemplet ovan, därför att då får det ena objektet Universal-taggen för VisibleString, det andra context-taggen **[1]** och det räcker för att de skall anses vara olika.

Etiketter (Tags)

Tags (etiketter) används för att skilja på olika typer. Tags är nödvändiga i sådana situationer där typen inte framgår av sammanhanget, t.ex. i en oordnad, blandad mängd av objekt av olika typer. Men tags får användas även när det inte är absolut nödvändigt, och det anses numera vara god ASN.1 att använda tags även när det inte är nödvändigt.

Orsak: Om man har ett element med en Tag, så kan man i framtida nya versioner av protokollet tillåta nya värden med andra Tags. Har man ingen Tag på ett element, får man det mycket svårare när man i framtiden skall definiera en utvidgad version av ett protokoll.

En Tag består av två komponenter:

- Tag class
- Tag number

Fyra klasser av etiketter(tags)

Klass	Exempel	Beskrivning
Universal	**[UNIVERSAL 1]**	I ASN.1-standarden definierad tag. [Universal 1] är t.ex. definierad som standard-tag för typen Boolean.
Application	**[APPLICATION 3]**	Har samma betydelse överallt inom en applikations-modul.
Private	**[PRIVATE 4]**	Om ett företag eller en organisation vill göra egna utvidgningar av ASN.1
Context	**[7]**	En omgivningsberoende (context dependent) tag har sitt värde bara i just det sammanhang där den används.

Anmärkning 1: I 1994 års ASN.1 avråds från användning av Application och Private tags.

Anmärkning 2: Med Automatic tagging behöver man sällan ange taggar.

Exempel på omgivningsberoende (context-dependent) tags:

```
Name                ::= SET {
                            given name      [0] VisibleString,
                            surname         [1] VisibleString }

PersonellRecord     ::= SET {
                            name            [0] Name,
                            wage            [1] INTEGER }
```

I detta exempel betyder tag-värdena [0] och [1] olika saker i de två exemplen på ASN.1-produktioner. I det övre fallet betyder [1] efternamn, i det undre fallet betyder [1] ett lönefält.

Vilka Tag-klasser används mest?

UNIVERSAL	Kräver att de i standarden inbyggda typerna räcker.
APPLICATION	Ger problem vid export och import. Ej rekommenderad i 1994 års ASN.1.
PRIVATE	Ger problem vid hopkoppling av olika tillämpningar, ANY och EXTERNAL kan göra samma sak bättre. Ej rekommenderad i 1994 års ASN.1.
CONTEXT	Entydiga i givet sammanhang.
(AUTOMATIC	Görs om till CONTEXT eller UNIVERSAL alltefter behov)

I ASN.1 fördefinierade etiketter: Universal Tags

Simple types

1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
9	REAL
10	ENUMERATED

Character String Types

12	UTF8String
18	NumericString
19	PrintableString
20	TeletexString
21	VideotexString
22	IA5String
25	GraphicString
26	VisibleString
27	GeneralString
28	UniversalString
29	CharacterString
30	BMPString

Structured types

16	SEQUENCE
16	SEQUENCE OF
17	SET
17	SET OF
(a)	CHOICE
(b)	ANY

(a) = Alla tags för de tillåtna alternativen

(b) = Samtliga möjliga tags

UsefulTypes

7	ObjectDescriptor
8	EXTERNAL
23	UTCTime
24	GeneralizedTime

Tagged — Structured Type

Värdemängd: Samma som någon annan typ, men med en ny, användargiven Tag

Typnotation:

[<tagclass> <tagnumber>] <type>
[<tagclass> <tagnumber>] IMPLICIT <type>
[<tagclass> <tagnumber>] EXPLICIT <type>

där **<tagclass>** kan vara
UNIVERSAL, APPLICATION eller
PRIVATE
om **<tagclass>** inte anges antas
"Context-specific"

Exempel:

[APPLICATION 7] INTEGER (0..9)
[0] IMPLICIT REAL
[PRIVATE 7] EXPLICIT BOOLEAN

Värdenotation:

Samma som för den underliggande typen

IMPLICIT och EXPLICIT får bara anges direkt efter en tag, och anger om den nya taggen skall ersätta eller komplettera tag för underliggande typ. Om ingendera anges antas förvalt värde för denna modul.

Subtyper: Samma som för underliggande typ

Explicit och Implicit tags

I modulhuvudet kan man ange

DEFINITIONS ::=

DEFINITIONS IMPLICIT TAGS ::=

DEFINITIONS EXPLICIT TAGS ::=

DEFINITONS AUTOMATIC TAGS ::= (I 1994 års ASN.1)

Om varken IMPLICIT TAGS eller EXPLICIT TAGS anges antas EXPLICIT TAGS. AUTOMATIC TAGS innebär EXPLICIT för CHOICE and Open Types, IMPLICIT otherwise.

Man kan sedan i texten ange t.ex.

Height [0] IMPLICIT REAL

Height [0] EXPLICIT REAL

Height [0] REAL

När varken IMPLICIT eller EXPLICIT anges gäller förval för hela modulen, enligt modulhuvudet.

Example of the same module specified with IMPLICIT and EXPLICIT tags


```
PersonnelFile { 2 3 4 4711 6 }
    DEFINITIONS IMPLICIT TAGS ::=

BEGIN

PersonRecord ::= SET {
    name IA5String,
    wage [0] EXPLICIT INTEGER,
    age [1] INTEGER }

Person ::= [APPLICATION 1]
    PersonRecord

Robot ::= [APPLICATION 2]
    EXPLICIT PersonRecord

END - - of module PersonnelFile
```

```
PersonnelFile { 2 3 4 4711 6 }
    DEFINITIONS EXPLICIT TAGS ::=

BEGIN

PersonRecord ::= SET {
    name IA5String,
    wage [0] INTEGER,
    age [1] IMPLICIT INTEGER }

Person ::= [APPLICATION 1]
    IMPLICIT PersonRecord

Robot ::= [APPLICATION 2]
    PersonRecord

END - - of module PersonnelFile
```

Which tags can be removed?

Record ::= SEQUENCE { **GivenName [0]** **PrintableString** **SurName [1] PrintableString }**	Both tags can be removed
Record ::= SET { **GivenName [0]** **PrintableString** **SurName [1] PrintableString }**	One of the tags can be removed
Record ::= SEQUENCE { **GivenName [0]** **PrintableString OPTIONAL** **SurName [1] PrintableString** **OPTIONAL }**	One of the two tags can be removed

Exercise 22

Which tags below can be removed in correct ASN.1, not using automatic tagging?

```
Colour ::= [APPLICATION 0] CHOICE {
                                    rgb [1] RGB-Colour,
                                    cmg [2] CMG-Colour,
                                    freq [3] Frequency
                                    }
RGB-Colour ::= [APPLICATION 1] SEQUENCE {
                                    red [0] REAL,
                                    green [1] REAL OPTIONAL,
                                    blue [2] REAL
                                    }
CMG-Colour ::= SET {     cyan [1] REAL,
                                    magenta [2] REAL,
                                    green [3] REAL
                                    }
Frequency ::= SET {      fullness [0] REAL,
                                    freq [1] REAL }
```

GeneralizedTime

Datum och tidpunkt med olika precision enligt ISO-standarder. Formatet är i huvudsak

YYYYMMDDHHMMSS.SSS±HHMM eller YYYYMMDDHHMMSS.SSSZ

Samma tidpunkt, 5 minuter och 33,8 sekunder efter 7 på morgonen den 2 januari 1982 i New York City kan anges som

eventtime GeneralizedTime ::= "19820102070533.8" eller

eventtime GeneralizedTime ::= "19820102070533.8Z" eller

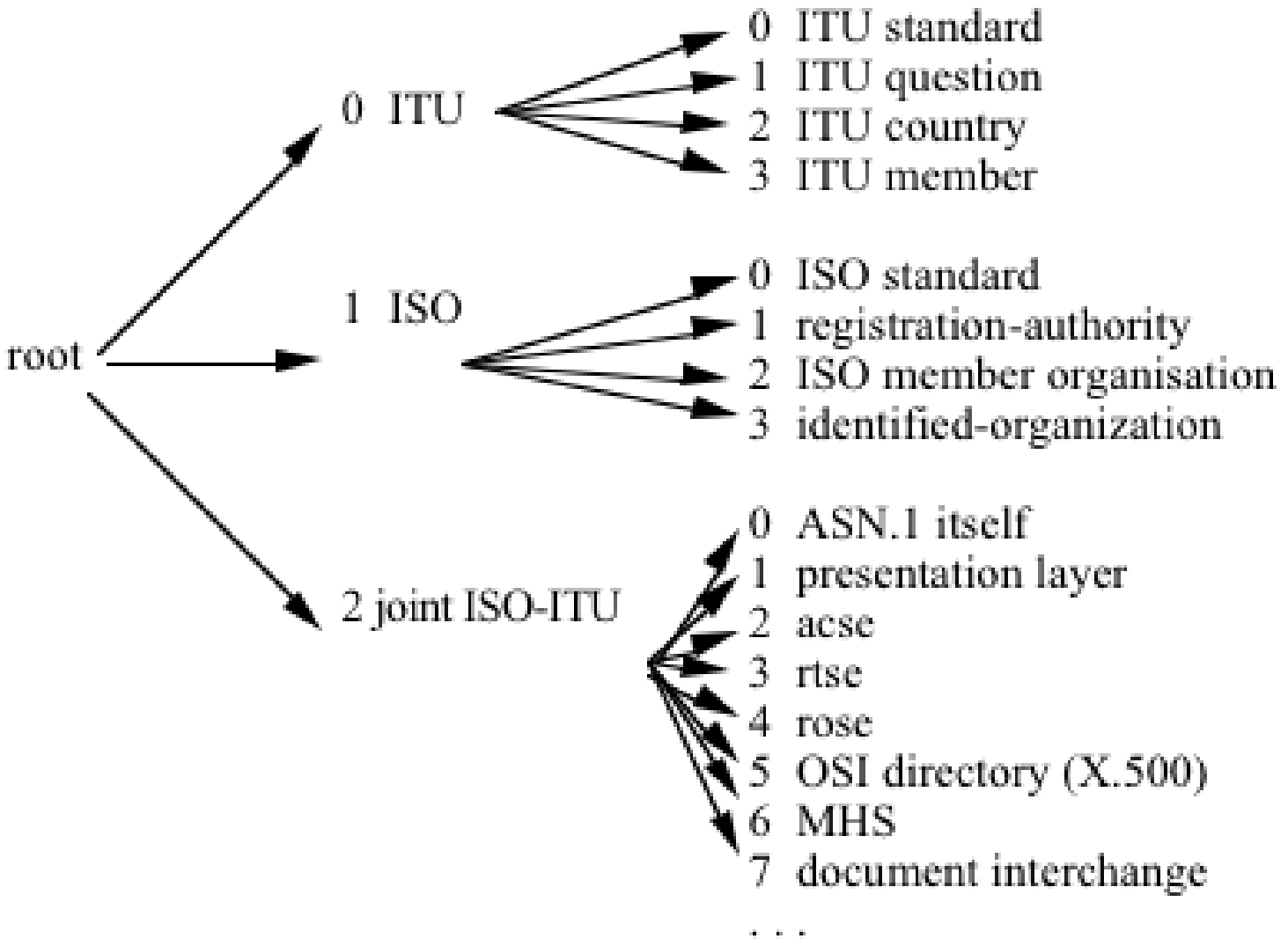
eventtime GeneralizedTime ::= "19820102070533.8-0500"

UTCTime (Universal time)

Något enklare och mer kortfattat än Generalized time. Årtal anges med bara två siffror, tiden anges i antingen hela minuter eller hela sekunder. Exempel:

eventtime UTCTime ::= "820102070534-0500"

Object identifier



External — Useful Type

Liknar typen Any, men en External kan innehålla ett värde som inte är i ASN.1-format. Externals yttre format kan definieras i ASN.1 på följande sätt:

```
EXTERNAL ::= [ UNIVERSAL 8 ] IMPLICIT SEQUENCE
{    direct-reference          OBJECT-IDENTFIER   OPTIONAL,
     indirect-reference        INTEGER            OPTIONAL,
     data-value-descriptor     ObjectDescriptor   OPTIONAL,
     encoding  CHOICE
     {         single-ASN1-type    [0]   ANY,
               octet-aligned       [1]   IMPLICIT OCTET STRING,
               arbitrary           [2]   IMPLICIT BIT STRING
     }
}
```

Minst en av tre OPTIONAL-alternativen måste ha ett värde.

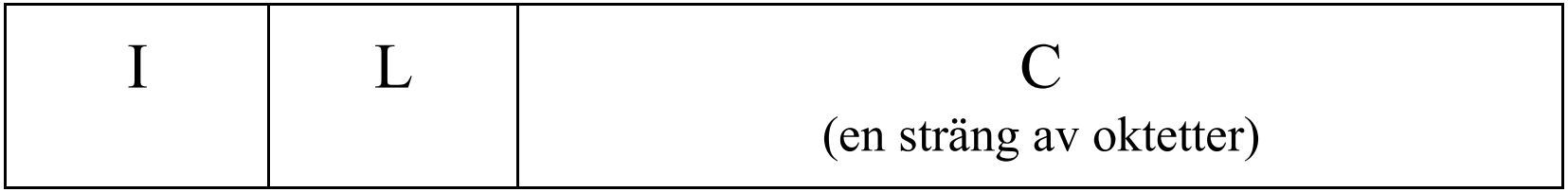
Innehållets typ kan alltså anges antingen genom en OBJECT-IDENTFIER (direct-reference) eller genom en INTEGER (indirect-reference). I det senare fallet tilldelas detta heltal ett värde i presentationslagret.

Exempel på modulnotation med IMPORTS

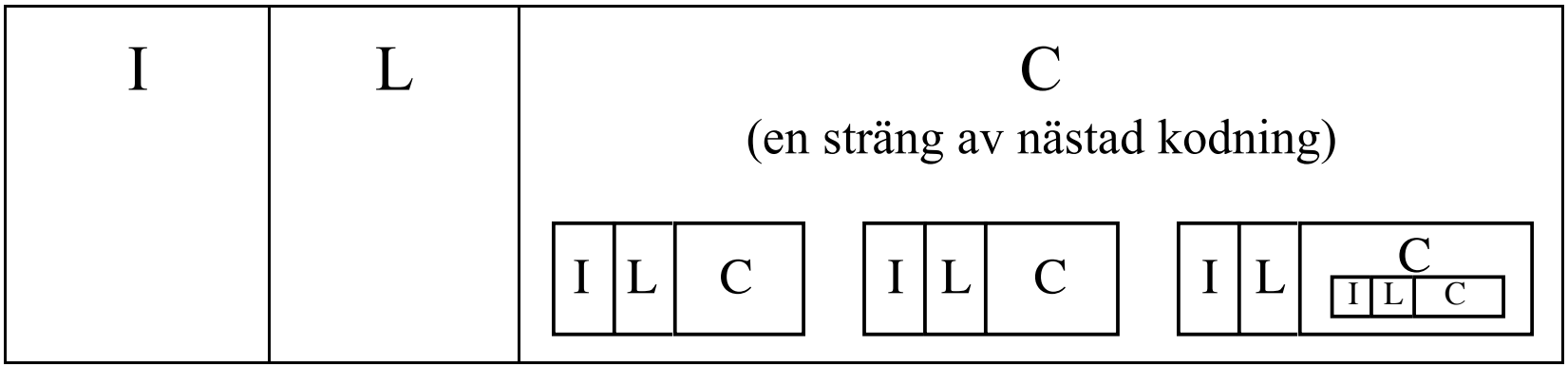

```
CargoHandling { 1  2  4711  17 } DEFINITIONS EXPLICIT TAGS ::=
BEGIN
EXPORTS Box, Container ;
Box ::= SEQUENCE {
                                height INTEGER, - -  in centimeters
                                width INTEGER, - -  in centimeters
                                length INTEGER } - -  in centimeters
Container ::= SEQUENCE
                                weight INTEGER, - -  in kilograms
                                volume Box }
END - -  of CargoHandling

TrainCargo { 1 2 4711 18 } DEFINITIONS EXPLICIT TAGS ::=
BEGIN
IMPORTS Box, Container FROM CargoHandling { 1  2  4711  17 };
TrainContainer ::= Container
                                ( WITH COMPONENTS
                                        {     weight ( 0 .. 5000 ), volume }
                                )
Carriage ::= SET SIZE (2..4) OF Container
END - -  of TrainCargo
```

Basic Encoding Rules (BER)

Primitive:



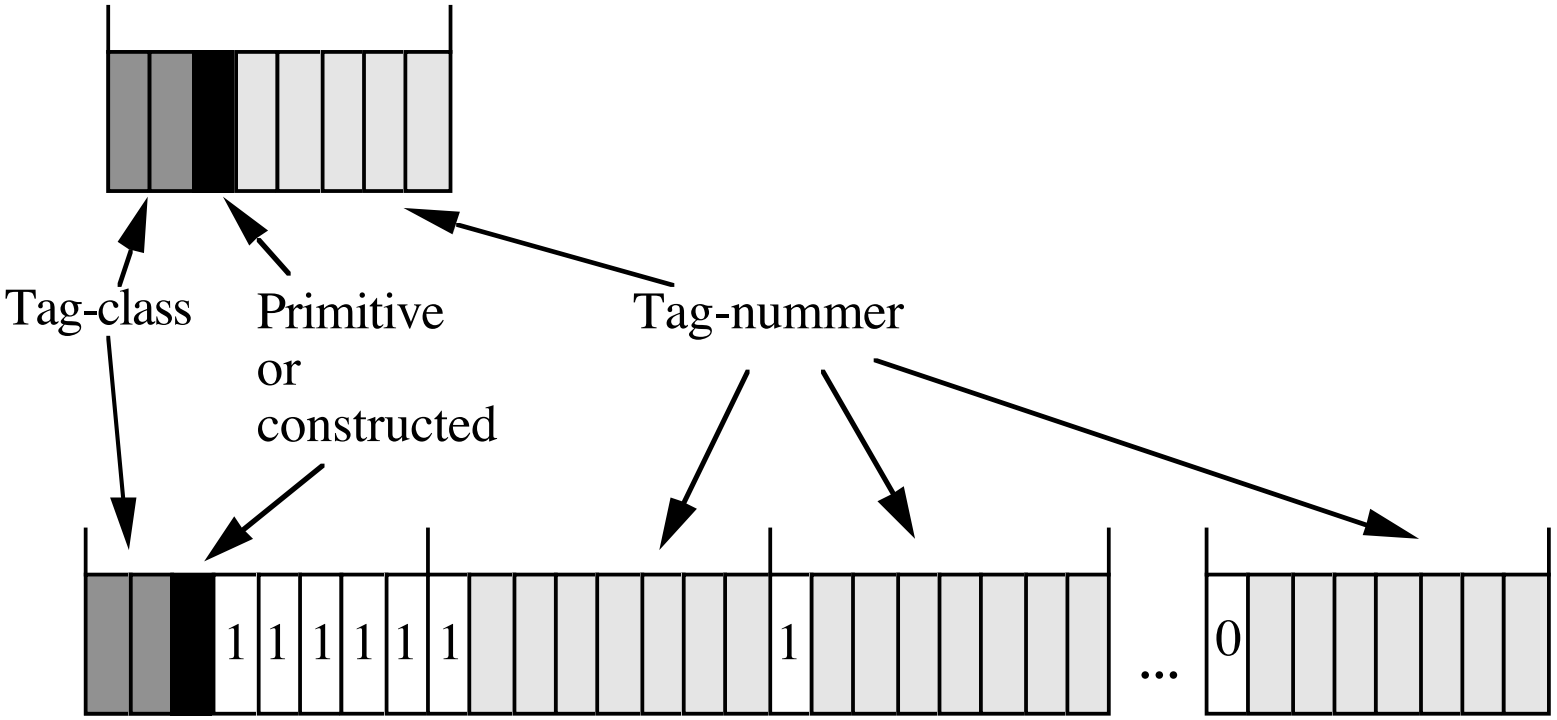
Constructed:



I = Identifier octets
L = Length octets
C = Contents octets

Identifierar-fältet i BER

En-oktett-varianten



Fler-oktett-varianten

Tag-class
(bit 8-9):

00 Universal

01 Application

10 Context

11 Private

P/C (bit 7)

0 Primitive

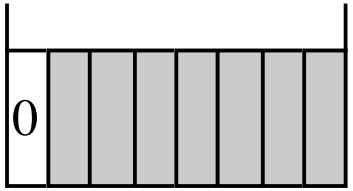
1 Constructed

Tag-nummer

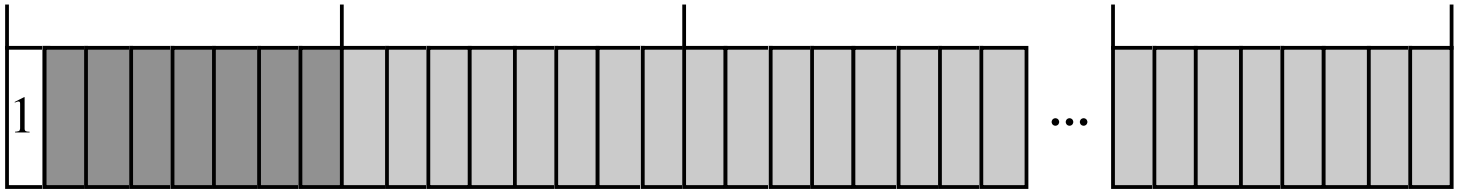
0 till 30 kan kodas i en-oktett-varianten med fem bitar

Längd-fältet i BER

Korta formen

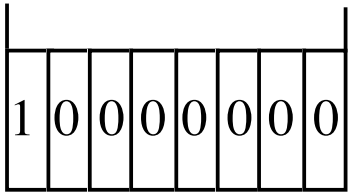


Långa formen

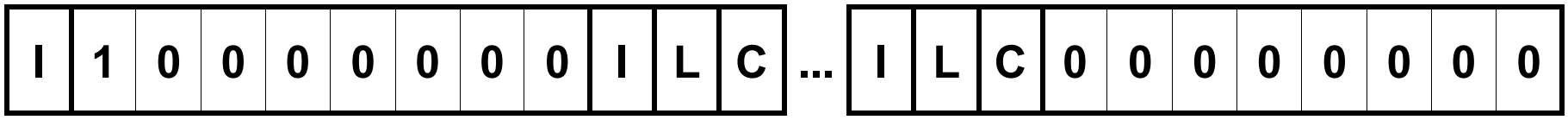


0 < n < 127 0 1 n

Obegränsade formen, avslutas med oktett med enbart 0-or



Avslutande av obegränsade formen



Observera att 0-oktetter mycket väl kan finnas inuti I, L och C-fältena i de underordnade fälten. Det är bara när avtolkaren väntar sig ett I-fält på samma nivå som startfältet, som avslutning sker med en 0-oktett. Eftersom tag-värdet 0 är reserverat för detta ändamål, kan en 0-oktett bara finnas när den avser en avslutning av den obegränsade formen.

Contents Octets

Boolean	En enda oktett. FALSE = 00000000 TRUE = alla andra värden
Integer	Tvåkomplementform, kodat i minsta nödvändiga antal oktetter.
Enumerated	Som för Integer.
Null	Ingen Contents Octet alls.
Object Identifier	A packet sequence of integers. Första motsvarar de första två linje-etiketterna, därefter en integer per etikett.
Set, Sequence, Set-of, Sequence-of	Nästade kodningar av komponenterna. Ordning är signifikant för sequence och sequence-of, inte för set och set-of
Choice, Any	Samma kodning som för utvald typ och värde

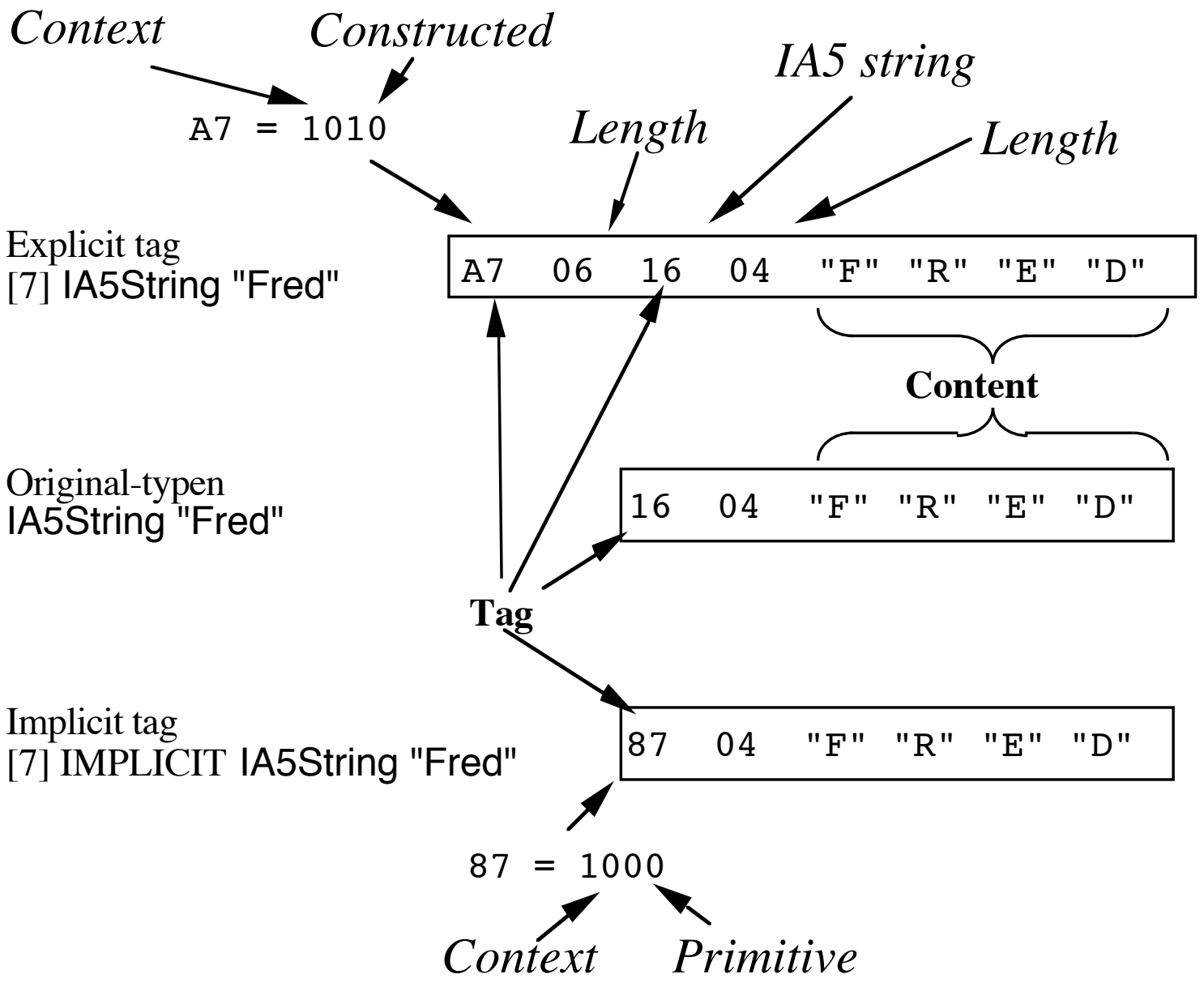
Contents Octets: String

I primitive form: Bitarna, oktetterna eller de kodade tecknen utom för Bit String, där första oktetten anger hur många bitar i sista oktetten som skall ignoreras.

I constructed form, som om ASN.1 hade haft definitionerna:

```
BIT STRING ::=      [UNIVERSAL 3] IMPLICIT SEQUENCE OF BIT
                        STRING
OCTET STRING ::=    [UNIVERSAL 4] IMPLICIT SEQUENCE OF OCTET
                        STRING
IA5String ::=       [UNIVERSAL 22] IMPLICIT SEQUENCE OF OCTET
                        STRING
```

Implicit och explicit Tagging



Exempel på kodning av en SEQUENCE

```
HeadOfState ::= [APPLICATION 17] SEQUENCE
{           name IA5 STRING,
type ENUMERATED {                   president (0),
                                    kejsare(1),
                                    kung(2) }
birthyear INTEGER OPTIIONAL }

swedishKing ::= {
name "Carl XVI Gustav",
type kung,
birthyear 1946 }
```

$22_{10} = 16_{16}$= class universal(00),
form primitive(0), tag number IA5(22)

0	0	0	1	0	1	1	0

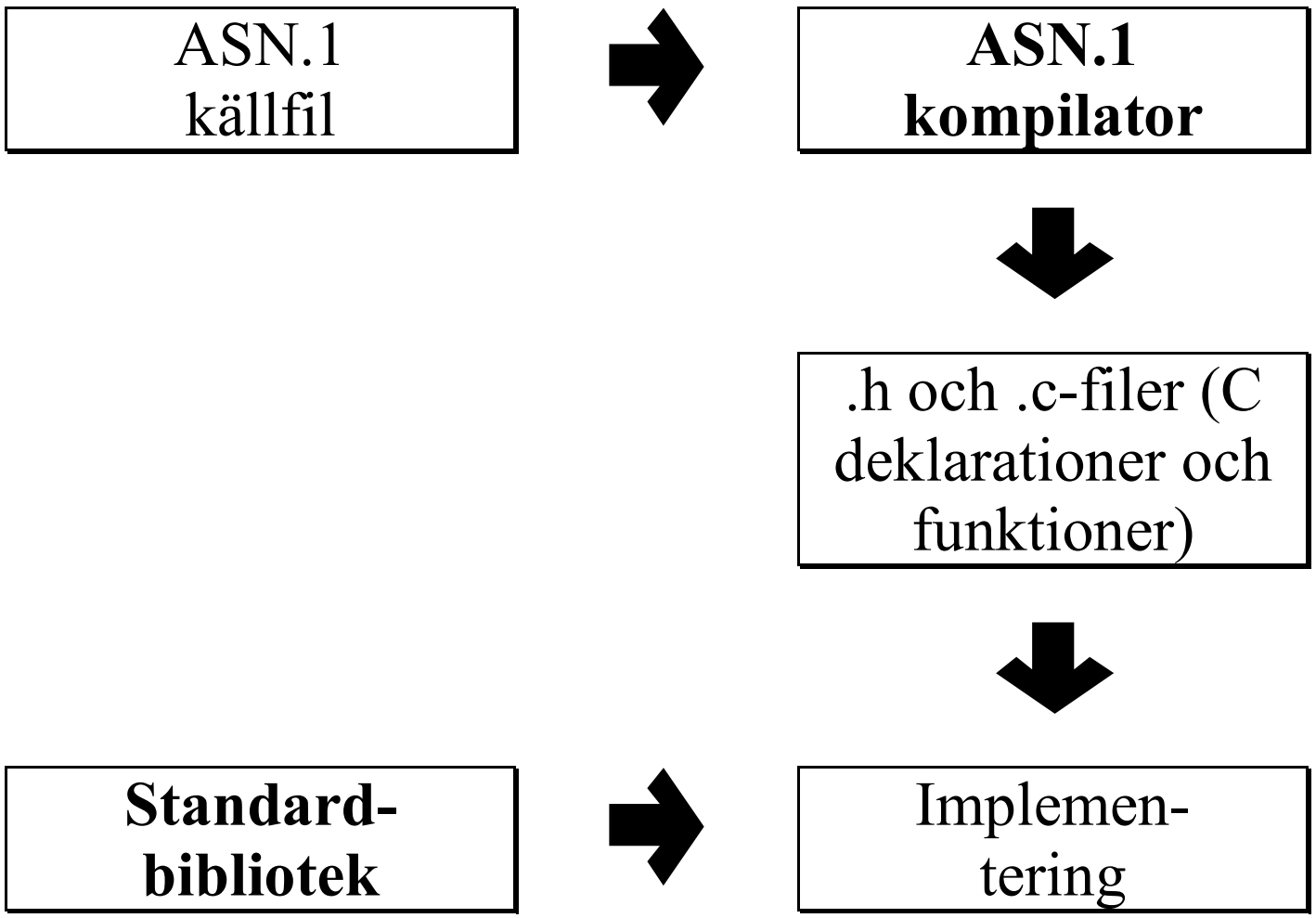
30	18	Hexadecimala tal

16	0F	C	a	r	l		X	V	I		G	u	s	t	a	v
0A	01	02														
02	02	1E	14													

Alternativa kodningar

BER = Basic Encoding Rules	Not very efficient, much redundancy, good support for extensions
DER = Distinguished Encoding Rules	No encoding options (for security hashing), always use definite length encoding
CER = Canonical Encoding Rules	No encoding options (for security hashing), always use indefinite length encoding
PER = Packed Encoding Rules	Very compact, less extensible
LWER = Light Weight Encoding Rules	Almost internal structure, fast encoding/deconding

ASN.1-kompilatorer



Exempel på ASN.1 (X.420 sid 548-551)


```
IPM ::= SEQUENCE {
                    heading      Heading
                    body         Body }

Heading ::= SET {
            this-IPM                ThisIPMField,
            originator              [0] OriginatorField OPTIONAL,
            authorizing-users       [1] AuthorizingUsersField OPTIONAL,
            primary-recipients      [2] PrimaryRecipientsField DEFAULT {},
            copy-recipients         [3] CopyRecipientsField DEFAULT {},
            . . .
```

Fråga: Varför är det ingen TAG angiven för This-IPM ovan???

```
ThisIPMField ::= IPMIdentifier

IPMIdentifier ::= [APPLICATION 11] SET {
            user                        ORAddress OPTIONAL,
            user-relative-identifier    LocalIPMIdentifier }

LocalIPMIdentifier ::= PrintableString (Size (0..ub-local-ipm-identifier))
```

From: Marshall T. Rose <mrose@dbc.mtview.ca.us>
Date: 12 jul 1995 05:12
... ...

Combining ASN.1 and high-performance is oxymornonic.

ASN.1 is probably the greatest failure of the OSI effort, it led hundreds of engineers, including myself, to devise data structures that were far too complicated for their own good.

(Oxymoron = Self-contradiction)

(Marshall T. Rose is a well-known previous OSI expert who has turned into one of the most vocal OSI enemies.)

From: Colin Robbins <c.robbins@nexor.co.uk>
Date 13 Jul 1995 16:58

Let me see if I have understood this debate.
X.400 is a brontosarus, because it uses ASN.1.
SMTP is a monkey because it does not.

Where does that leave the SNMPv2 Protocol, desgined by the Internet community, co-auther one Marshall T. Rose. It uses ASN.1. I thought leopards didn't change their spots!

There are plenty or reasons to knock X.400, but the use of ASN.1 is not one of them. Sure it has its faults, but BOTH the Internet and OSI communities are using it.

Litteratur för den somn vill lära sig mera

Douglas Steedman: Abstract Syntax Notation One ASN.1 The tutorial & Reference. Technology Appraisals 1990. *(Kan köpas i bokhandeln, boken är mycket dyr.)*

X.208 (ASN.1)

X.209 (BER)

X.219 (ROS)

X.420 (Interpersonal Messaging Service)

X.500 (Directory System)