Agile Network Service Development

ASN.1

Gusztáv Adamis
BME TMIT

Data Specification

- High level data structure definition
  - Only structure
  - Can be complicated
- Abstract Data Types (~classes)
  - Data structure + operations
  - Axiomatic
    - ACT ONE in SDL
  - Pragmatic, constructive
    - C++, JAVA, …

ASN.1

- Abstract Syntax Notation One
  - High level structures
  - Originally for Application layer protocols
  - Now mainly for mobile protocols
- Two parts
  - Data Specification
  - Encoding Rules
    - Binary encoding
    - BER – Basic Encoding Rules
      - CER – Canonical Encoding Rules
      - DER – Distinguished Encodig Rules
      - variations of BER
    - PER – Packed Encoding Rules
      - compact encoding
      - for radio interface protocols

ASN.1 Syntax

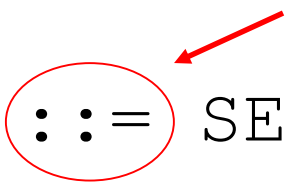
- UPPERCASE and lowercase characters different
- Keywords – all capital
- Identifiers – start with letter, contains letters, numbers, and -;
  - two consequitive - not allowed
- Types – first letter: capital
- Other identifiers (e.g. record fields, enumerated literals) – first letter: lowercase
- -- comment till the end of line

ASN.1 Data Type Specification

- Universal Types
  - INTEGER
  - BOOLEAN
  - REAL
  - NULL
  - BIT STRING – ‘1010’B (length=4)
  - OCTET STRING – ‘AB12’O (length=2)
  - IA5String – ”Hello”
    - other string types: GraphicalString, NumericalString, PrintableString, etc.

Construction Rules

- SEQUENCE ~structure, record
- SET – similar, but fields can be transmitted in any order
- OPTIONAL – field may be omitted
- DEFAULT
- CHOICE – union (only one of the fields)




```
PersonalData ::= SEQUENCE {
  age          INTEGER DEFAULT 10,
  married      BOOLEAN OPTIONAL }
```

Construction Rules ctd.

- SEQUENCE OF / SET OF
  - arrays
  - SEQUENCE OF INTEGER
  - SET OF PersonalData

- ENUMERATED

```
Colors ::= ENUMERATED {blue, green, red}
```

Sub-Types

- Small-Integer ::= INTEGER(0..9)
- SmallPrimes ::= INTEGER(2|3|5|7)
- Array64 ::= SEQUENCE SIZE (64..64) OF INTEGER
  - or
- Array64 ::= SEQUENCE SIZE (64) OF INTEGER
- ArrayMax64::= SEQUENCE SIZE (1..64) OF INTEGER
- Bit8 ::= BIT STRING SIZE(8)
- Telephone-Number ::= IA5String(FROM('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'0'|'*'|'#'))

BER – Basic Encoding Rules

- Each data is sent as a triplet:
- TAG + Length + Value (TLV)
  - TAG ~ Type Code
  - Simple Types
    - ‘Value’ is really the value
  - Structured Types
    - At ‘Value’: new TAG+Length+Value triplet(s) stand recoursively
    - E.g.: Value of a SEQUENCE is the list of its fields

Encoding of a TAG

TAG

Class:

- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:

- **0 = Primitive**
- **1 = Structured**

TAG values for Universal Types:

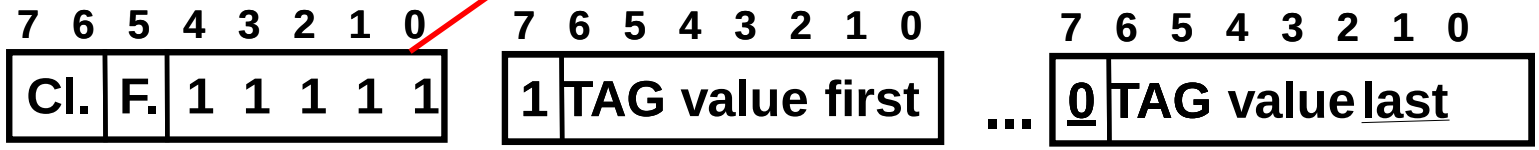
- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

Encoding of a TAG

TAG short form:

Max short value = 30 (11110)

TAG long form:



Class:

- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:

- **0 = Primitive**
- **1 = Structured**

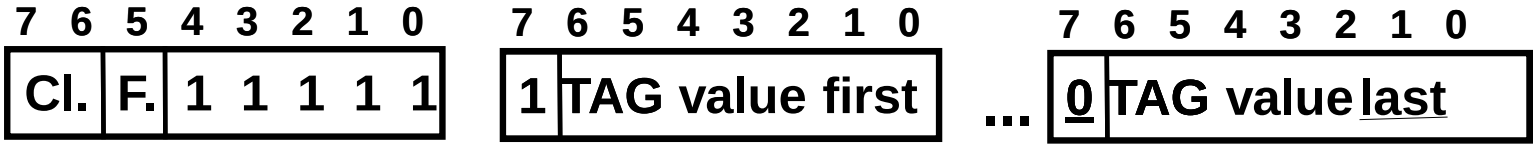
TAG values for Universal Types:

- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

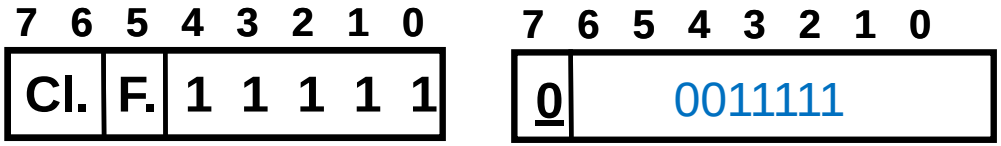
Encoding of a TAG

TAG short form:

TAG long form:



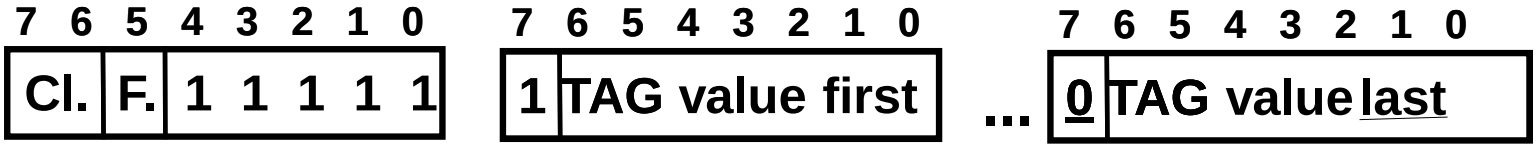
TAG value = 31



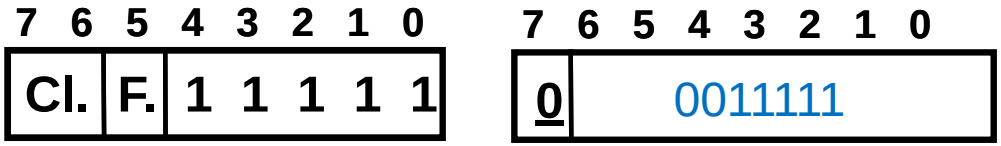
Encoding of a TAG

TAG short form:

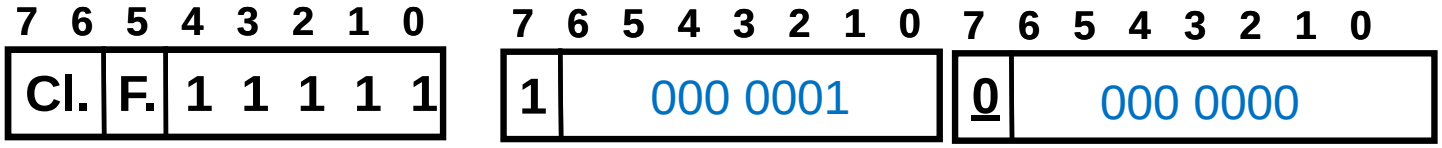
TAG long form:



TAG value = 31

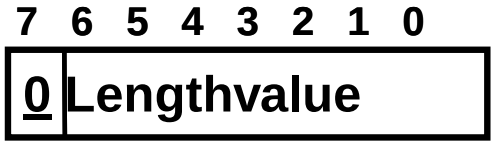


TAG value = 128 (1000 0000)

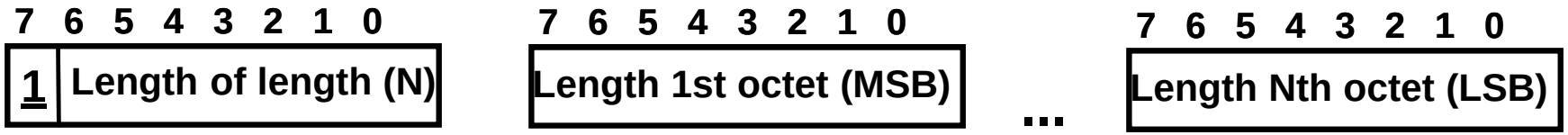


Encoding of Length

- Length short form (Length <= 127):

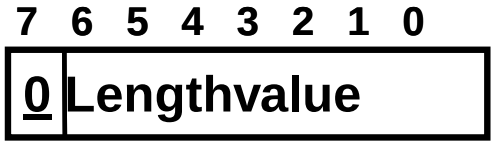


- Length long form (Length > 127)

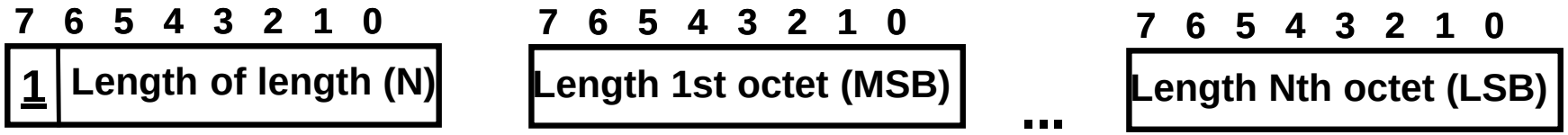


Encoding of Length

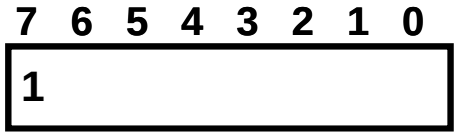
- Length short form (Length <= 127):



- Length long form (Length > 127)

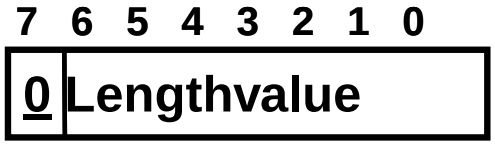


Length = 256

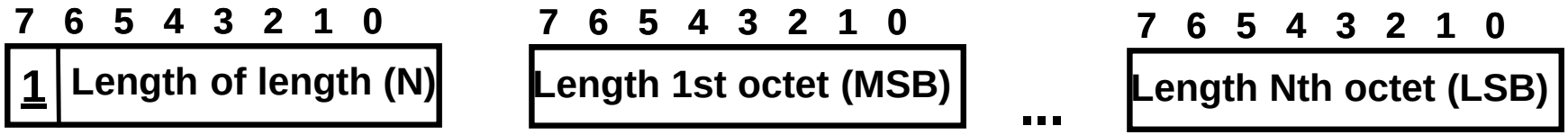


Encoding of Length

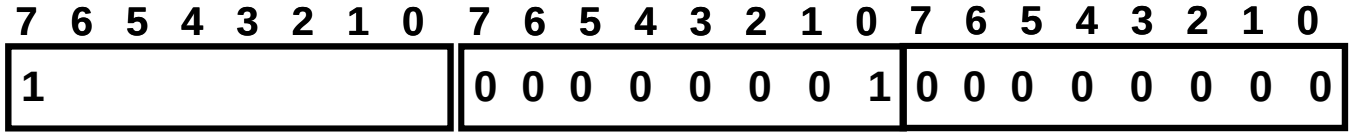
- Length short form (Length <= 127):



- Length long form (Length > 127)

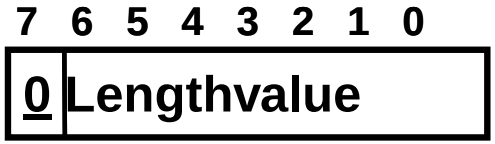


Length = 256

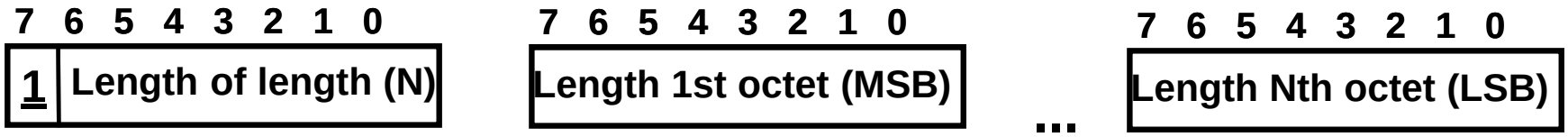


Encoding of Length

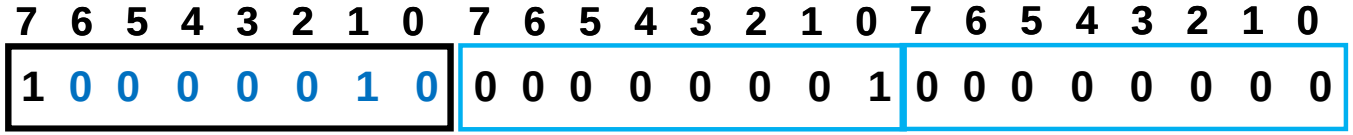
- Length short form (Length <= 127):



- Length long form (Length > 127)

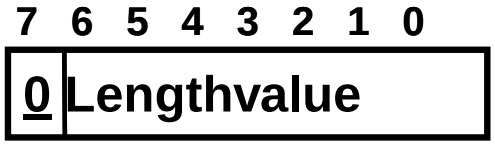


Length = 256

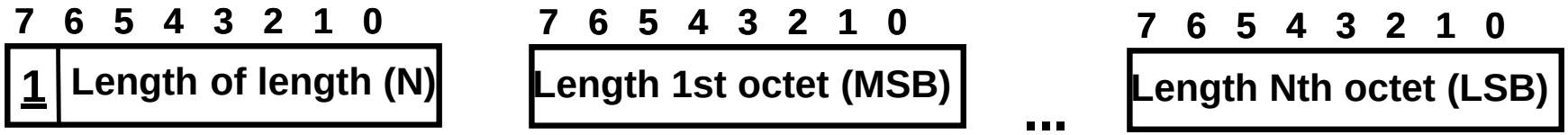


Encoding of Length

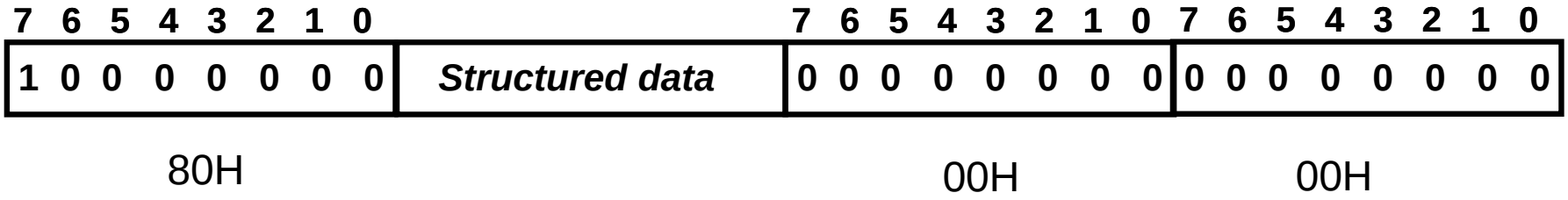
- Length short form (Length <= 127):



- Length long form (Length > 127)



- Length indefinite form:



BER Example – NULL, BOOLEAN

NULL

TAG	Length	~~V~~
00 **0** 00101	**0** 0000000	

BOOLEAN (TRUE)

TAG	Length	Value
00 **0** 00001	**0** 0000001	00000001

Class:

- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:

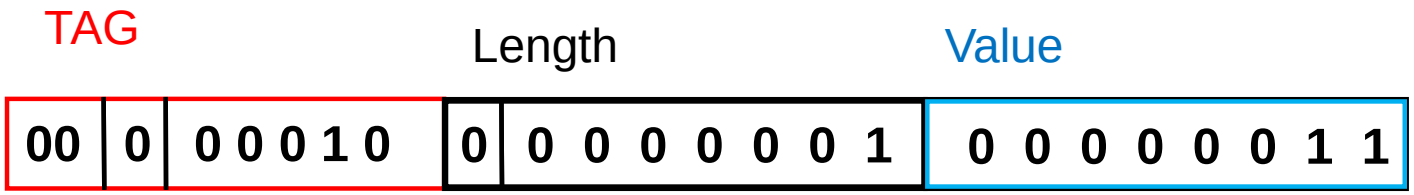
- **0 = Primitive**
- **1 = Structured**

TAG values for Universal Types:

- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

BER Example – INTEGER

- INTEGER with value of 3



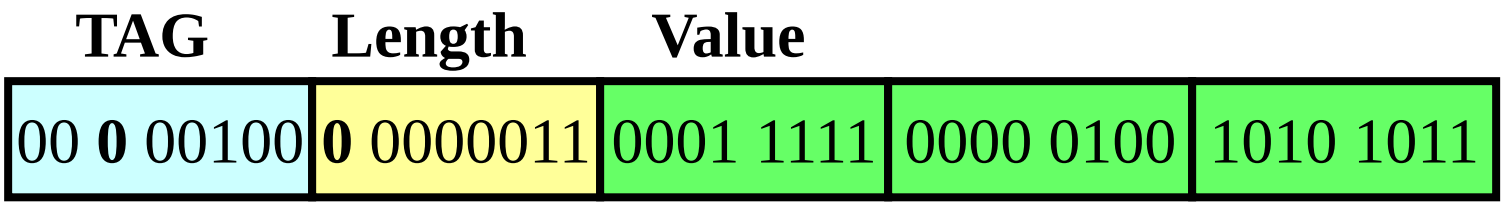
Class:
- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:
- **0 = Primitive**
- **1 = Structured**

TAG values for Universal Types:
- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

BER Example – OCTET STRING

- OCTET STRING with value of ‘1F04AB’O



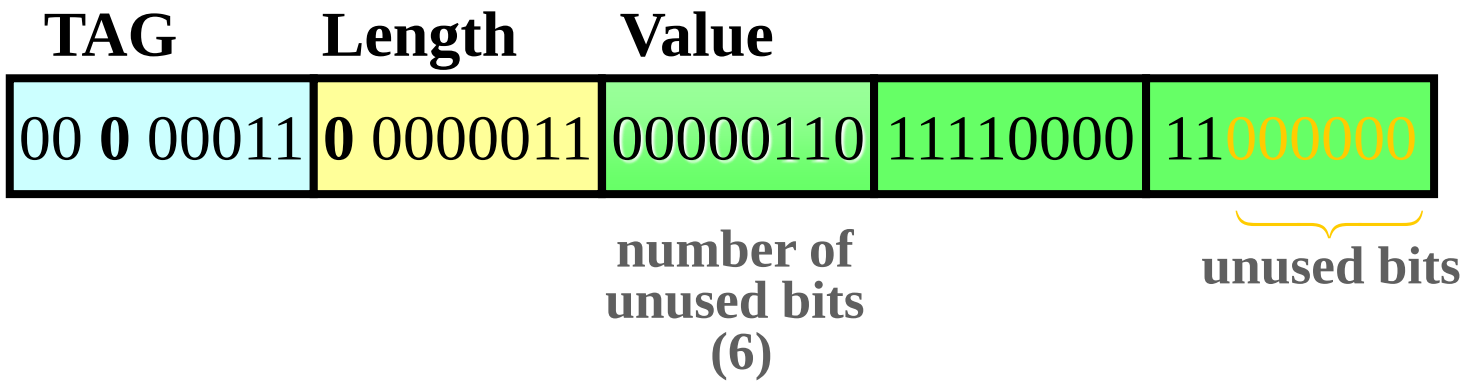
Class:
- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:
- **0 = Primitive**
- **1 = Structured**

TAG values for Universal Types:
- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

BER Example – BIT STRING

- BIT STRING with value of '1111000011'B



Class:

- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:

- **0 = Primitive**
- **1 = Structured**

TAG values for Universal Types:

- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

BER Example – Array

- Array of 2 INTEGERS: 5 and 10
  - SEQUENCE OF INTEGER

TAG [SEQUENCE OF]	Length	
00 1 10000=30H	06H	
TAG [INTEGER]	Length	Value
00 0 00010=02H	01H	05H
TAG [INTEGER]	Length	Value
00 0 00010=02H	01H	0AH

Class:
- **00 = Universal**
- **01 = Application wide**
  (Standardised type)
- **10 = Context-specific**
  (SEQUENCE/SET/CHOICE field)
- **11 = Private**
  (Non standardised type)

Format:
- **0 = Primitive**
- **1 = Structured**

TAG values for Universal Types:
- **1 = BOOLEAN**
- **2 = INTEGER**
- **3 = BIT STRING**
- **4 = OCTET STRING**
- **5 = NULL**
- **10 = ENUMERATED**
- **16 = SEQUENCE / SEQUENCE OF**
- **17 = SET / SET OF**
- **22 = IA5String**

User defined TAG - Introduction

```
Coordinate ::= SET {
   x INTEGER,
   y INTEGER
   }
```

- in which order the fields transmitted?

```
Coordinate ::= SEQUENCE {
   x INTEGER OPTIONAL,
   y INTEGER OPTIONAL
   }
```

- if only one field transmitted: which?

User defined TAG values

```
Coordinate ::= SET {
    x [0] INTEGER,
    y [1] INTEGER
    }
```

- to interpret: x is type 0, which is actually an INTEGER

```
Coordinate ::= SEQUENCE {
    x [0] INTEGER OPTIONAL,
    y [1] INTEGER OPTIONAL
    }
```

Encoding of User Defined TAG values

- x [0] INTEGER
  - Meaning: x is of a type 0, that is actually an INTEGER
- Encoding:
  - [0] – Format: 1, since the value contains the type code of the actual type (INTEGER in this example)
  - TAG Value: the value in [ ] (0 in this example)
  - TAG [0] + Length of the whole data value + TAG of the actual data type (INTEGER) + Length of the actual data + Value of the actual data

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG **Length**

[APPLICATION 3]

01 1 00011=63H

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG **Length**
[APPLICATION 3]
01 1 00011=63H

TAG **Length**
[SEQUENCE]
00 1 10000=30H

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG **Length**
[APPLICATION 3]
01 1 00011=63H

TAG **Length**
[SEQUENCE]
00 1 10000=30H

TAG **Length**
[Context Specific 0]
10 1 00000=A0H

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG	Length			
[APPLICATION 3]				
01 1 00011=63H				
TAG	Length			
[SEQUENCE]				
00 1 10000=30H				
TAG	Length			
[Context Specific 0]				
10 1 00000=A0H				
TAG	Length	Value		
[INTEGER]				
00 0 00010=02H	01H	04H		

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG **Length**
[APPLICATION 3]
01 1 00011=63H

TAG **Length**
[SEQUENCE]
00 1 10000=30H

TAG **Length**
[Context Specific 0]
10 1 00000=A0H **03H**

TAG	Length	Value
[INTEGER]		
00 0 00010=02H	01H	04H

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG	Length			
[APPLICATION 3]				
01 1 00011=63H				
TAG	Length			
[SEQUENCE]				
00 1 10000=30H				
TAG	Length			
[Context Specific 0]				
10 1 00000=A0H	03H			
	TAG	Length	Value	
	[INTEGER]			
	00 0 00010=02H	01H	04H	
TAG	Length			
[Context Specific 1]				
10 1 00001=A1H				

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG **Length**
[APPLICATION 3]
01 1 00011=63H

TAG **Length**
[SEQUENCE]
00 1 10000=30H

TAG **Length**
[Context Specific 0]
10 1 00000=A0H **03H**

TAG	Length	Value
[INTEGER]		
00 0 00010=02H	01H	04H

TAG **Length**
[Context Specific 1]
10 1 00001=A1H

TAG	Length	Value
[INTEGER]		
00 0 00010=02H	01H	05H

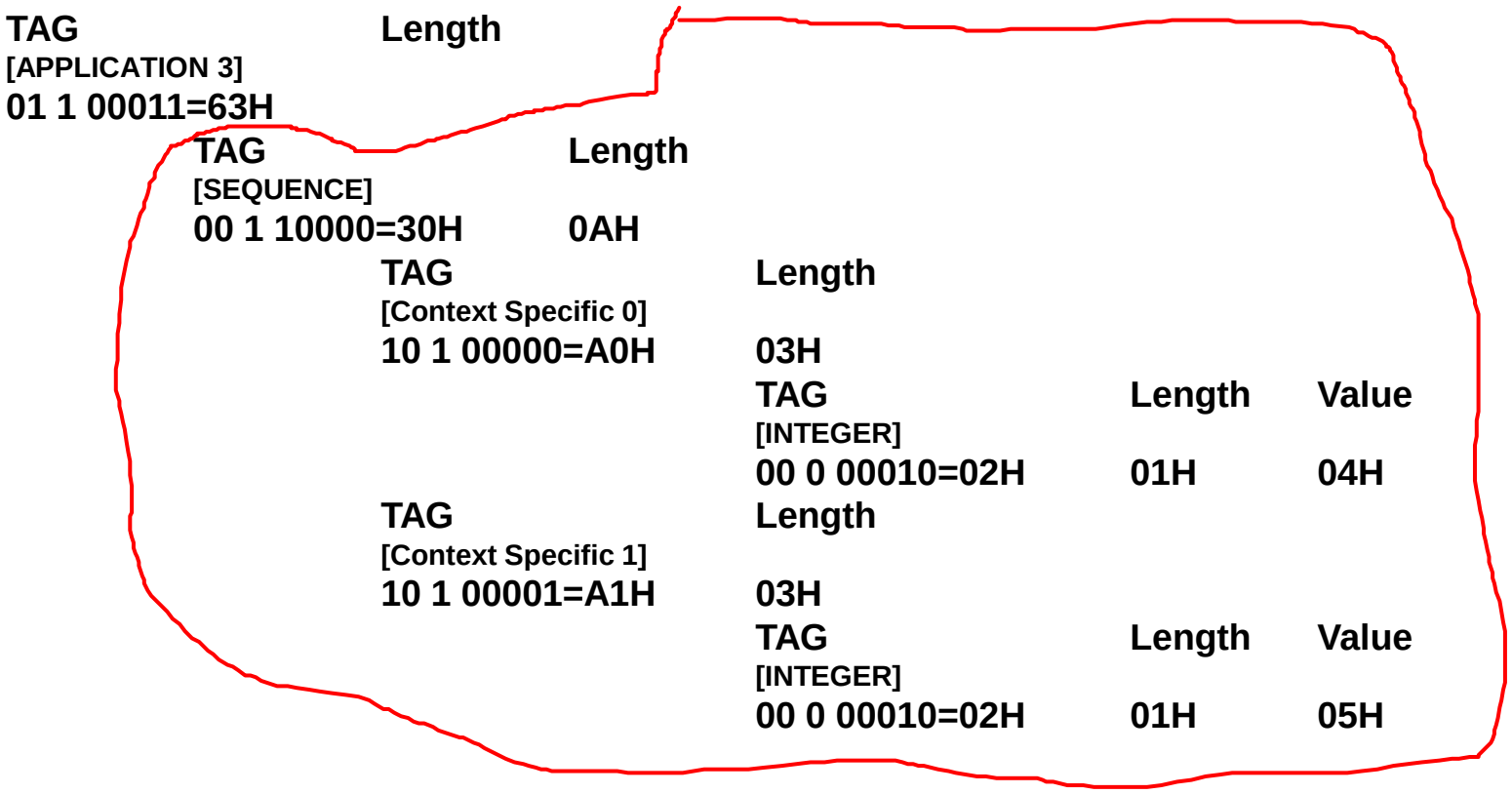
BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG	Length		
[APPLICATION 3]			
01 1 00011=63H			
TAG	Length		
[SEQUENCE]			
00 1 10000=30H			
TAG	Length		
[Context Specific 0]			
10 1 00000=A0H	03H		
	TAG	Length	Value
	[INTEGER]		
	00 0 00010=02H	01H	04H
TAG	Length		
[Context Specific 1]			
10 1 00001=A1H	03H		
	TAG	Length	Value
	[INTEGER]		
	00 0 00010=02H	01H	05H

BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```



BER Example – SEQUENCE

```
Coordinate ::= [3] SEQUENCE{
x [0] INTEGER OPTIONAL, -- 4
y [1] INTEGER OPTIONAL  -- 5
}
```

TAG	Length			
[APPLICATION 3]				
01 1 00011=63H	0CH			
TAG	Length			
[SEQUENCE]				
00 1 10000=30H	0AH			
TAG	Length			
[Context Specific 0]				
10 1 00000=A0H	03H			
	TAG	Length	Value	
	[INTEGER]			
	00 0 00010=02H	01H	04H	
TAG	Length			
[Context Specific 1]				
10 1 00001=A1H	03H			
	TAG	Length	Value	
	[INTEGER]			
	00 0 00010=02H	01H	05H	

IMPLICIT encoding

- x [0] **IMPLICIT** INTEGER

- For user TAGged fields, the actual type is not transmitted
- More compact code, but to decode, the data structure shall be known by the decoder
- Encoding:
  - [0] – Format: format of the actual type, since the type code of the actual type (INTEGER in this example) NOT transmitted (Format: 0 in this example)
  - TAG [0] + Length of the actual data + Value of the actual data

BER Example – Implicit SEQUENCE

```
Coordinate ::= [3] IMPLICIT SEQUENCE{
x [0] IMPLICIT INTEGER OPTIONAL, -- 4
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

BER Example – Implicit SEQUENCE

```
Coordinate ::= [3] IMPLICIT SEQUENCE{
x [0] IMPLICIT INTEGER OPTIONAL, -- 4
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG [APPLICATION 3]

Length

01 1 00011=63H

BER Example – Implicit SEQUENCE

```
Coordinate ::= [3] IMPLICIT SEQUENCE{
x [0] IMPLICIT INTEGER OPTIONAL, -- 4
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length	
[APPLICATION 3] 01 1 00011=63H		
TAG	Length	Value
[Context Specific 0] 10 0 00000=80H	01H	04H

BER Example – Implicit SEQUENCE

```
Coordinate ::= [3] IMPLICIT SEQUENCE{
x [0] IMPLICIT INTEGER OPTIONAL, -- 4
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length	Value
[APPLICATION 3]		
01 1 00011=63H		
TAG	Length	Value
[Context Specific 0]		
10 0 00000=80H	01H	04H
TAG	Length	Value
[Context Specific 1]		
10 0 00001=81H	01H	05H

BER Example – Implicit SEQUENCE

```
Coordinate ::= [3] IMPLICIT SEQUENCE{
x [0] IMPLICIT INTEGER OPTIONAL, -- 4
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length	
[APPLICATION 3]		
01 1 00011=63H	06H	
TAG	Length	Value
[Context Specific 0]		
10 0 00000=80H	01H	04H
TAG	Length	Value
[Context Specific 1]		
10 0 00001=81H	01H	05H

BER Example – Indefinite Length

- Only for Structured types
- Same example as the previous

TAG	Length		
[APPLICATION 3]			
01 1 00011=63H	06H		
	TAG	Length	Value
	[Context Specific 0]		
	10 0 00000=80H	01H	04H
	TAG	Length	Value
	[Context Specific 1]		
	10 0 00001=81H	01H	05H

BER Example – Indefinite Length

- Only for Structured types
- Same example as the previous

TAG	Length		
[APPLICATION 3]	(indefinite)		
01 1 00011=63H	80H		
TAG	Length	Value	
[Context Specific 0]			
10 0 00000=80H	01H	04H	
TAG	Length	Value	
[Context Specific 1]			
10 0 00001=81H	01H	05H	
TAG	"Length "		
[End of Content]			
00 0 00000=00H	00H		

BER Complex Example

```
Coordinate ::= [5] IMPLICIT SEQUENCE{ -- Indefinite
length encoding
x [0] INTEGER OPTIONAL, -- 4 + Indef. length
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

BER Complex Example

```
Coordinate ::= [5] IMPLICIT SEQUENCE{ -- Indefinite
length encoding
x [0] INTEGER OPTIONAL, -- 4 + Indef. length
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length
[APPLICATION 5]	(indefinite)
01 1 00101=65H	80H

TAG	"Length"
[End of Content - SEQUENCE]	
00 0 00000=00H	00H

BER Complex Example

```
Coordinate ::= [5] IMPLICIT SEQUENCE{ -- Indefinite
length encoding
x [0] INTEGER OPTIONAL, -- 4 + Indef. length
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length
[APPLICATION 5]	(indefinite)
01 1 00101=65H	80H

TAG	Length
[Context Specific 0]	(indefinite)
10 1 00000=A0H	80H

TAG	"Length"
[End of Content - First Field]	
00 0 00000=00H	00H

TAG	"Length"
[End of Content - SEQUENCE]	
00 0 00000=00H	00H

BER Complex Example

```
Coordinate ::= [5] IMPLICIT SEQUENCE{ -- Indefinite
length encoding
x [0] INTEGER OPTIONAL, -- 4 + Indef. length
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length
[APPLICATION 5]	(indefinite)
01 1 00101=65H	80H

TAG	Length
[Context Specific 0]	(indefinite)
10 1 00000=A0H	80H

TAG	Length	Value
[INTEGER]		
00 0 00010=02H	01H	04H

TAG	"Length"
[End of Content - First Field]	
00 0 00000=00H	00H

TAG	"Length"
[End of Content - SEQUENCE]	
00 0 00000=00H	00H

BER Complex Example

```
Coordinate ::= [5] IMPLICIT SEQUENCE{ -- Indefinite
length encoding
x [0] INTEGER OPTIONAL, -- 4 + Indef. length
y [1] IMPLICIT INTEGER OPTIONAL  -- 5
}
```

TAG	Length
[APPLICATION 5]	(indefinite)
01 1 00101=65H	80H

TAG	Length
[Context Specific 0]	(indefinite)
10 1 00000=A0H	80H

TAG	Length	Value
[INTEGER]		
00 0 00010=02H	01H	04H

TAG	"Length"
[End of Content - First Field]	
00 0 00000=00H	00H

TAG	Length	Value
[Context Specific 1]		
10 0 00001=81H	01H	05H

TAG	"Length"
[End of Content - SEQUENCE]	
00 0 00000=00H	00H

How ASN.1 used in 3GPP specifications

- Operations
  - Parameters of request message: ARGUMENT
  - Parameters of reply message: RESULT
  - List of possible errors: ERRORS
  - Operation code: CODE
- Errors
  - (Optional) parameters of error: PARAMETER
  - Error code: CODE

Semi-Formal Definitions

```
name_of_operation OPERATION ::= {
   ARGUMENT
        list of parameters at invocation
   RESULT
        list of parameters at return
   ERRORS {
        list of error types may occur
        during operation execution
   }
   CODE local: operation_code
}


error_type_name ERROR ::= {
   PARAMETER
        optional further information
   CODE local: error_code
}
```

MAP Example

```
checkIMEI OPERATION ::= {
    ARGUMENT
      imei OCTET STRING (SIZE (3..8))
    RESULT
      equipmentStatus ENUMERATED {
               whiteListed (0),
               blackListed (1),
               greyListed (2)}
    ERRORS {
      systemFailure, dataMissing,
    unexpectedDataValue, unknownEquipment }
    CODE local : 43
}

systemFailure ERROR ::= {
    PARAMETER
      networkResource ENUMERATED{
               plmn     (0),
               hlr              (1),
               ....
               }
    CODE local : 34
}
dataMissing ERROR ::= {
    CODE local :35
}
```

```
mo-ForwardSM OPERATION  ::=  {
  ARGUMENT     SEQUENCE {
    sm-RP-DA            CHOICE {
      imsi                  [0] IMPLICIT OCTET STRING ( SIZE( 3 .. 8 ) ),
      lmsi                  [1] IMPLICIT OCTET STRING ( SIZE( 4 ) ),
      serviceCentreAddressDA     [4] IMPLICIT OCTET STRING ( SIZE( 1 .. 20 ) ),
      noSM-RP-DA                 [5] IMPLICIT NULL},
    sm-RP-OA            CHOICE {
      msisdn                [2] IMPLICIT OCTET STRING ( SIZE( 1 .. 20 ) ) ( SIZE( 1 .. 9 ) ),
      serviceCentreAddressOA     [4] IMPLICIT OCTET STRING ( SIZE( 1 .. 20 ) ),
      noSM-RP-OA                 [5] IMPLICIT NULL},
    sm-RP-UI            OCTET STRING ( SIZE( 1 .. 200 ) ),
    extensionContainer   SEQUENCE {
      privateExtensionList   [0] IMPLICIT SEQUENCE  ( SIZE( 1 .. 10 ) ) OF
        SEQUENCE {
          extId      MAP-EXTENSION .&extensionId  ( {
            ,
            ...} ) ,
          extType    MAP-EXTENSION .&ExtensionType  ( {
            ,
            ...} { @extId   }  )  OPTIONAL} OPTIONAL,
      pcs-Extensions        [1] IMPLICIT SEQUENCE {
        ... } OPTIONAL,
      ... } OPTIONAL,
    ... ,
    imsi                OCTET STRING ( SIZE( 3 .. 8 ) ) OPTIONAL}
  RESULT     SEQUENCE {
    sm-RP-UI            OCTET STRING ( SIZE( 1 .. 200 ) ) OPTIONAL,
    extensionContainer   SEQUENCE {
      privateExtensionList   [0] IMPLICIT SEQUENCE  ( SIZE( 1 .. 10 ) ) OF
        SEQUENCE {
          extId      MAP-EXTENSION .&extensionId  ( {
            ,
            ...} ) ,
          extType    MAP-EXTENSION .&ExtensionType  ( {
            ,
            ...} { @extId   }  )  OPTIONAL} OPTIONAL,
      pcs-Extensions        [1] IMPLICIT SEQUENCE {
        ... } OPTIONAL,
      ... } OPTIONAL,
    ... }
  ERRORS     {
    systemFailure |
    unexpectedDataValue |
    facilityNotSupported |
    sm-DeliveryFailure }
  CODE     local      : 46
}
```

```
sm-DeliveryFailure ERROR  ::=  {
  PARAMETER     SEQUENCE {
    sm-EnumeratedDeliveryFailureCause   ENUMERATED {
      memoryCapacityExceeded        ( 0 ),
      equipmentProtocolError        ( 1 ),
      equipmentNotSM-Equipped       ( 2 ),
      unknownServiceCentre          ( 3 ),
      sc-Congestion                 ( 4 ),
      invalidSME-Address            ( 5 ),
      subscriberNotSC-Subscriber    ( 6 ) },
    diagnosticInfo                  OCTET STRING ( SIZE( 1 .. 200 ) ) OPTIONAL,
    extensionContainer              SEQUENCE {
      privateExtensionList   [0] IMPLICIT SEQUENCE  ( SIZE( 1 .. 10 ) ) OF
        SEQUENCE {
          extId      MAP-EXTENSION .&extensionId  ( {
            ,
            ...} ) ,
          extType    MAP-EXTENSION .&ExtensionType  ( {
            ,
            ...} { @extId   }  )  OPTIONAL} OPTIONAL,
      pcs-Extensions        [1] IMPLICIT SEQUENCE {
        ... } OPTIONAL,
      ... } OPTIONAL,
    ... }
  CODE     local      : 32
  }
```