# JL's Web Tutorials

# Encoding Control Notation (ECN)
# (An early tutorial - Dec 2000)



***First ECN Tutorial***

# Copyright notice

- This material can be freely copied and used for non-commercial purposes.
- It should not be used for commercial purposes without prior permission.

# What is ECN?

- Formally-defined notation.
- Tools support.
- ASN.1+ECN is the phrase to use.
- Completely determines the encoding for ASN.1 types to which it is applied.
- Can be used without existing encoding rules, but usually supplements them.

*First ECN Tutorial*

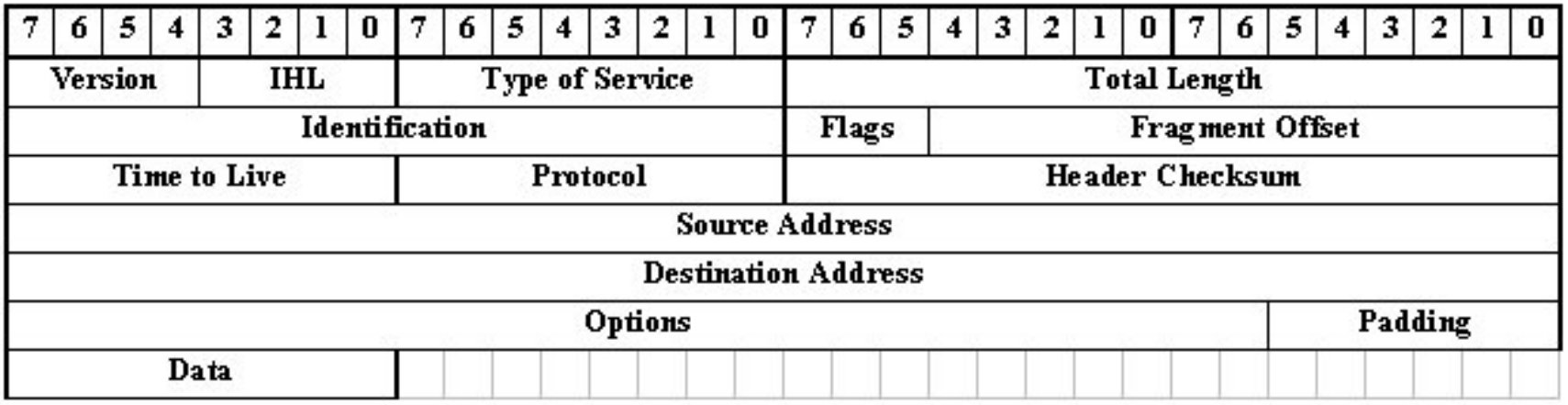
# Applications of ASN.1+ECN

- Specialised encodings of simple fields - for example Huffman encoding of integers.
- Specialised encodings of constructors, e.g.
  - more-bit sequence-of
  - optionals with preceding presence bit
  - tail-end optionals
- Specialized mechanisms for extensibility.
- Re-definition of legacy protocols.

# Legacy protocols (1)

## Diagrams of bits and bytes - IPv4 etc

**More relevant examples can be found in ISDN specifications! But this will do as an illustration!**




Tool support not possible

***First ECN Tutorial***

# Legacy protocols (2)

- Each parameter has
  Parameter ID (or type), length, value
- Tables list each parameter: Tabular Notation

**Connect Message format**

| Parameter ID | Length | Optionality | Semantics |
|---|---|---|---|
| Version | 1 octet | Mandatory | See para 14.2 |
| Priority | 1 octet | Optional (default 0) | See para 14.3 |
| Called address | Variable | Mandatory | See para 14.4 |
| Calling address | Variable | Mandatory | See para 14.5 |
| Additional information | Variable | Optional | See para 14.6 |

Tool support not possible

# Power, simplicity, clarity, brevity

- ECN has a robust framework.
- Anything it can't do can easily be added within the framework.
- A lot of the work on ECN was to ensure that it could express simply what had to be expressed.
- Keywords chosen to aid readability.
- Attempts to avoid verbose specifications.

# Early history

- Began with MBER work in the 1980s.
- Problem of auxiliary fields acting as determinants.
- Producing the bits is easy, identifying the determinant linkages is harder.
- MBER was incorporated into PER.

# Application fields and auxiliary fields

- Presence determinants.
- Length determinants.
- Choice determinants.
- ECN has to support many different ways of providing determination, including use of application fields.

# Recent history (1)

- Frank Schramm's and Colin Wilcock's initiative in Jan 1999.
- Proposed EC (later ECN) for ITU-T and ISO/IEC standardization.
- Originally "1-1 model", similar to early workers, but ….
- Recognised the need for notation to link determinants to the determined field.

*First ECN Tutorial*

# Recent history (2)

- Strong ETSI involvement during 2000 (and into 2001?).
- Initially, work on the base standard.
- Extended to include “ETSI Case Studies” of legacy protocols and the “ETSI ECN Tutorial”.
- Rejected re-definition of PER and BER.

*First ECN Tutorial*

# Status of ITU-T Recommendation and ISO/IEC Standard

- Currently under ballot as an ISO/IEC Final Committee Draft.
- Scheduled for ITU-T “consent” in January 2001.
- Will be followed by ISO/IEC Final Draft International Standard (FDIS) ballot and ITU-T “decision”, probably in early Summer 2001.

# Encodable item - beginnings of understanding

- Part of an ASN.1 definition whose encoding can be independently specified.
- Originally just the definition of encodings of bit-fields (boolean, integer, etc).
- Extended to procedures for encoding sequence-of, choices, optionality.

# Encoding classes - the end result

- The set of all possible encodings for a part of the procedures for encoding ASN.1 types.
- Includes encoding definitions for bit-fields (boolean and integer classes).
- Includes definitions for handling sequence-of, optionality, etc.
- The definitions are encoding objects for the encoding class.

# More on encoding classes

- Encoding classes start with a “#”.
- Examples: #INTEGER, #SEQUENCE, #My-defined-type.
- Are synonyms for primitive classes #INT, #CONCATENATION.
- Form the building blocks of encoding structure definitions.

# Encoding structure definitions

- Specify the structure of fields (and construction mechanisms) in an encoding.
- Similar to ASN.1 type definition, but underlying concepts very different.
- Fundamental to ECN work.
- CSN.1 considered, but a new ASN.1-like notation eventually developed.
- More later.

# Structure of an ASN.1+ECN specification

- Composed of modules.
- Export import between the modules.
- Similar outer structure to ASN.1 modules.
- Many ASN.1 modules - unchanged
- Many Encoding Definition Modules (EDM modules)
- One Encoding Link Module (ELM)

# EDM modules

**EDM-module-my-name**

**{joint-iso-itu-t(2)** ***To be supplied*****}**

**ENCODING-DEFINITIONS ::=**

**BEGIN**

**EXPORTS ......**

**GENERATES-AND-EXPORTS ......**

**IMPORTS ......**

**.....**

**END**

# The ELM module

**ELM-module-my-name**

**{joint-iso-itu-t(2)** ***To be supplied*****}**

**LINK-DEFINITIONS ::=**

**BEGIN**

**IMPORTS .....**

**.....**

**END**

# Applying encodings

- Define encoding objects for encoding classes.
- Form an encoding object set (only one object for each class).
- “Apply” (in the ELM) an encoding object set to an ASN.1 type.
- Not quite true - to a structure generated from the ASN.1 type.

# Syntax to define encoding objects

- **Encoding objects are to encoding classes as ASN.1 values are to ASN.1 types:**

  **my-int INTEGER ::= .....**

  **my-boolean-encoding #BOOLEAN ::= ……**

- **Governor concept carries over.**

# Encoding structures

- Describe the structure of an encoding.
- Implicitly generated structures derived from ASN.1 type definitions.
- Explicitly generated structures transform these into the actual fields needed in the final encoding.
- Encoding objects both encode fields and perform transformations.

# Encoding structure features

- Encoding structures are simpler than ASN.1 types, but still have primitive fields and constructors.
- Primitive fields could be just bit strings, but easy mapping from ASN.1 requires richer primitive fields.
- Constructors have different names from ASN.1 because they operate on bit-fields, not on abstract values.

# Encoding structure definition (1)

```
#Structure1 ::=  #CONCATENATION {
        field1  #INT,
        -- Comment can be embedded, as in ASN.1
        field2  #BOOL,
        field3  #NUL,
        field4  #PAD,
        field5  #ALTERNATIVES {
            alt1  #CHARS,
            alt2  #BITS },
        field6  #REPETITION {
            #OCTETS },
        field7  #Structure2 }
```

# Encoding structure definition (2)

```
#Structure2 ::= #CONCATENATION {
                field1  #OID,
                field2  #REL-OID,
                field3  #OPEN-TYPE,
                field5  #REAL,
                field6  #Structure3 }
```

# Encoding structure definition (3)

```
#Structure3 ::= #TAG #CONCATENATION {
        field1  #BOOL,
        field2  #INT #OPTIONAL,
        field3  #TAG #Structure4,
        field4  #INT (0..7),
        field5  #OCTETS (SIZE 6) }
```

# Encoding structure definition (4)

```
#Structure4 ::= #CONCATENATION {
        field1 #INT,
        #EXTENSIONS {
            #VERSION-BRACKET {
                field1   #INT,
                field2   #BOOL } ,
            #VERSION-BRACKET {
                field1   #CHARS,
                field2   #OCTETS } } ,
        field2 #BOOL,
        field3 #Structure5 }
```

# Encoding structure definition (5)

```
#Structure5 ::=  #SEQUENCE {
        field1  #INTEGER,
        -- Looks rather like #Structure1, yes?
        -- But the names are more like ASN.1
        field2  #BOOLEAN,
        field3  #NULL,
        field5  #CHOICE {
                alt1  #UTF8String,
                alt2  #BIT-STRING },
        field6  #SEQUENCE-OF {
                #OCTET-STRING } }
```

# Built-in synonyms for classes

| | |
|---|---|
| #SEQUENCE | ::= #CONCATENATION |
| #INTEGER | ::= #INT |
| #BOOLEAN | ::= #BOOL |
| #NULL | ::= #NUL |
| #CHOICE | ::= #ALTERNATIVES |
| #UTF8String | ::= #CHARS |
| #BIT-STRING | ::= #BITS |
| #SEQUENCE-OF | ::= #REPETITION |
| #OCTET-STRING | ::= #OCTETS |

**Note the assignment of synonyms for constructor classes.**

# Implicitly generated encoding structures

- ASN.1 types => Implicit structure.
- Value references resolved.
- Named bits ignored.
- Names of enumerations lost.
- Parameterization resolved.
- Components of and Selection types expanded.
- etc.

# ASN.1 example

```
Color ::= [1] INTEGER {red(10), orange(20), yellow(30),
                       green(40), blue(50), indigo(60), violet(70)}

Version ::= [2] BIT STRING {version1(0), version2(2)}

My-type ::= SEQUENCE {
   field1      Version,
   field2      Color,
   field3      CHOICE {
       alt1    [3] INTEGER (0..15),
       alt2    [4] INTEGER (16..MAX) },
   field4      SEQUENCE OF
               OCTET STRING (SIZE 10) DEFAULT ''B}
```

# Corresponding implicitly generated structure

```
#Color ::= #TAG #INTEGER

#Version ::= #TAG #BIT-STRING

#My-type ::= #SEQUENCE {
    field1      #Version,
    field2      #Color,
    field3      #CHOICE {
        alt1            #TAG #INTEGER (0..15),
        alt2            #TAG #INTEGER (16..MAX) },
    field4      #SEQUENCE-OF {
                    #OCTET-STRING (SIZE 10) #OPTIONAL }
```

# Explicitly generated encoding structures

- Sometimes called “coloring” an implicitly generated structure.
- Replacing some classes with synonyms to identify places needing different encodings.
- Under discussion: The concept of “attributes” of fields/classes (colours) which cannot be easily recorded by changing the class.

# Coloring syntax

```
EDM-coloring-module
    {joint-iso-itu-t(2) To be supplied}
    ENCODING-DEFINITIONS ::=
    BEGIN
    EXPORTS My-encodings
      -- An object set containing all specialized encodings
    GENERATES-AND-EXPORTS
      REPLACING
        #OPTIONAL WITH #My-Optional
          IN ALL
      REPLACING
        #SEQUENCE-OF WITH #My-Sequence-of
          IN ALL EXCEPT #Group-Identity-Uplink, #Proprietary
  etc
END
```

# The heavy stuff!

- It is (fairly) easy to define special encodings (adding determinants, specifying encodings of fields and constructions) for a given structure.
- But defining rules for encoding a long ASN.1 definition in a special way requires rules for adding fields, applying encodings, etc. that apply to any structure.

# Parameterisation is key

- **Adding a preceding presence bit for optionality:**

**Replace**

```
#Any-class #OPTIONAL
```

**with**

```
#New-component {< #Any-class >} ::=
        #SEQUENCE {
                presence-bit  #BOOL,
                component     #Any-class #My-Inner-Optional }
```

# Getting replacement done is the second key

- Definition of an encoding object can specify replacement of optional or mandatory components (or - perhaps- “red” components).
- Needs provision of a parameterized encoding object for the structure.
- Can require a parameter for the global encoding object set.

# Defining a legacy protocol

- **Write the ASN.1 - no auxiliary fields, no encoding influence!**
- ***To be pure or not to be pure, that is the question!***
- **Coloring the implicitly generated structure.**
- **Defining the needed encoding objects and an object set.**
- **Applying the encodings.**

# The need for enhanced coloring

- **Replacement control.**
  - **Optional, mandatory.**
  - **“Type 1”, “Type 2”, etc.**
- **Insertion point for determinant fields.**
  - **Head, before first optional, tail?**
  - **At point X, Y, X.**
- **Add “encoding attributes” and “insertion markers” as part of “coloring”.**
- **Bit-map?**

# Model for defining simple bit-field encoding objects

- See picture in ECN Tutorial
- Encoding-space
- Value-encoding

# Functionality and syntax for defining a #BOOL encoding object

- See Annex A of ECN Specification

# Example #BOOL encoding object (1)

```
my-bool-encoding #BOOL ::=
        {ENCODING-SPACE
            SIZE 1
            MULTIPLE OF bit
        VALUE
            TRUE-PATTERN  bits:'0'B
            FALSE-PATTERN bits:'1'B }
```

# Example #BOOL encoding object (2)

```
#BOOL2 ::= #BOOL

my-bool2-encoding #BOOL2 ::=
    {ENCODING-SPACE
        ALIGNED TO octet
        SIZE 1
        MULTIPLE OF octet
     VALUE
        TRUE-PATTERN octets:'00'H
        FALSE-PATTERN other }
```

# Defining encoding objects (1)

- **Use an existing encoding object**

second-bool2-object #BOOL2 ::= my-bool2-encoding

- **Use "defined syntax"**

  **(See Annex A example of #BOOL)**

# Defining encoding objects (2)

- **Use an existing encoding object set**

```
per-int-encoding #INTEGER ::=
        { ENCODE WITH PER-BASIC-UNALIGNED }
```

or

```
per-structure1-encoding #Structure1 ::=
        { ENCODE WITH DER }
```

# Defining encoding objects (3)

- **Use value mappings**

```
encoding-for-old-class #Old-class ::=
        { USE #Replacement-structure
          MAPPING
          ......
          WITH
           { ENCODED WITH PER-BASIC-UNALIGNED }
           -- The above line is defining an encoding object
           -- for the class #Replacement-structure
```

# Defining encoding objects (4A)

- **Encoding a structure component by component.**

**Structure:**

```
#Sequence1 ::= #SEQUENCE {
        a           #Structure1,
        inserted    #BOOLEAN,
        b           #INTEGER #OPTIONAL,
        c           #OCTET-STRING }
```

# Defining encoding objects (4B)

- **An encoding object could be:**

```
sequence1-encoding #Sequence1 ::= {
        ENCODE STRUCTURE {
          b OPTIONAL-ENCODING
              {ENCODING-SPACE AS aux-determinant
               DETERMINED BY inserted } }
        WITH { ENCODE WITH PER-BASIC-UNALIGNED }
```

# Defining encoding objects (5)

- **Differential encode/decode**

```
object-for-tail-end-additions  #PAD ::=
        { ENCODE-DECODE
            {ENCODING-SPACE SIZE empty}
            -- Specifies an empty encoding for ENCODE
          DECODE AS IF
            {SIZE uses-determination-mechanism
             DETERMINED BY container
             CONTAINED IN end-of-encoding:NULL
             PADDING encoder-option } }
```

# Defining encoding objects (5)

- **User-defined functions**

```
encoding-of-funny-type #Funny-type ::=
        USER-FUNCTION-BEGIN
        {joint-iso-itu-t(2) etc } -- specifies the notation used
        -- Pseudo-code for encoding values of #Funny-type
          ......
        -- Pseudo-code for decoding values of #Funny-type
          ......
        USER-FUNCTION-END
```

# Mapping Values (1)

- **Mapping single values**

```
encoding-for-old-class #UTF8String ::=
        { USE #INTEGER
          MAPPING VALUES {
            "0" TO 0, "1" TO 1, "2" TO 2, "3" TO 3,
            "4" TO 4, "5" TO 5, "6" TO 6, "7" TO 7,
            "8" TO 8, "9" TO 9, "*" TO 10, "#" TO 11 }
          WITH
          encoding-of-integer }
```

# Mapping Values (2)

- **Mapping by matching fields**

```
encoding-for-old-class #Old-class ::=
        { USE #Replacement-class
          MAPPING FIELDS
          WITH
           encoding-of-replacement }
```

# Mapping Values (3)

- **Mapping using #TRANSFORM objects**

```
encoding-for-old-class #INTEGER1 ::=
        { USE #INTEGER2
          MAPPING TRANSFORMS
          { {INT-TO-INT divide:2} }
          WITH
           encoding-of-integer2 }
```

# Mapping Values (4)

- **Mapping by abstract value ordering**

```
encoding-for-old-class #Old-class ::=
        { USE #Replacement-class
          MAPPING ORDERED VALUES
          WITH
          encoding-of-replacement }
```

# Mapping Values (5)

- **Mapping by distributing values**

```
encoding-for-old-class #Old-class ::=
        { USE #Replacement-class
          MAPPING DISTRIBUTION
          0 .. 9 TO field1,
          10 .. 99 TO field2,
          REMAINDER TO field3
          WITH
          encoding-of-replacement }
```

# Mapping Values (6)

- Mapping integer values into bits
- Huffman encodings
- Word macro support
- See ECN Specification

# Defining encoding object sets

**My-encodings #ENCODINGS ::=**

**{ obj1 | obj2 | obj3 | Their-encodings }**

# Applying encodings in the ELM

**ELM-module-my-name**

**{joint-iso-itu-t(2) *To be supplied*}**

**LINK-DEFINITIONS ::=**

**BEGIN**

**IMPORTS My-encodings FROM EDM-Encodings-module { ..... }**

**#My-Messages FROM EDM-Coloring-module { ..... }**

**ENCODE #My-Messages WITH My-encodings**

**COMPLETED BY PER-BASIC-UNALIGNED**

**END**

***First ECN Tutorial***

# Further reading

- The ETSI Case Studies
- The ECN Specification

# Electronic fora

- asn1ecn@oss.com
- www.elibel.fr

# We are done!

*First ECN*