1.1. A comparison of ABNF, ASN.1-BER/PER and DTD-XML

Table 1 shows an example of the same information as encoded with ABNF, ASN.1-BER and DTD-XML.

Table 2 compares some properties of the three encoding methods.

Table 1: The same information with ABNF, ASN.1 and XML

<table>
<tr><td>ABNF specification:</td><td>ASN.1 specification:</td><td>DTD specification:</td></tr>
<tr><td><pre>
Family = "Family"
       CRLF *(Person)
      "End of Family"

Person = "Person" CRLF
      " Name: " 1*A CRLF
      " Birthyear: " 4D CRLF
      " Gender: "
     ("Male"/"Female") CRLF
     " Status: "
     ("unmarried"/ "married"/
     "divorced"/ "widow"/
     "widower")
</pre></td><td><pre>
= SEQUENCE OF Person

:= SEQUENCE {
 name VisibleString,
 birthyear INTEGER,
 gender Gender,
 status Status }

:= ENUMERATED {
 male(0), female(1) }

= ENUMERATED {
 unmarried(0), married(1), divorced(2),
 widow(3), widower(4) }
</pre></td><td><pre>
<!ELEMENT family
    (person+)>
<!ELEMENT person (name,
     birthyear)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthyear
    (#PCDATA)>
<!ATTLIST person
  gender (male | female)
          #REQUIRED
  status (unmarried |
  married | divorced |
  widow | widower)
   #REQUIRED
>
</pre></td></tr>
<tr><td>Example of textual encoding:</td><td>Example of BER encoding:</td><td>Example of XML encoding:</td></tr>
<tr><td><pre>
Family
Person
  Name: John Smith
  Birthyear: 1958
  Gender: Male
  Status: Married
Person
  Name: Eliza Tennyson
  Birthyear: 1959
  Gender: Female
  Status: Married
End of Family
</pre></td><td>(Each box represents one octet. Two-character codes are hexadecimal numbers, one character codes are characters)
30 34
30 16
1A 0A J o h n ␣ S m i t h
02 02 07 A6
0A 01 00
0A 01 01
30 1A
1A 0E E l i z a ␣ T e n n y s o n
02 02 07 A7
0A 01 01
0A 01 01</td><td><pre>
<?xml version="1.0" ?>
<!DOCTYPE family SYSTEM
"family.dtd">
<family>
 <person gender="male"
  status="married">
  <name>John Smith</name>
  <birthyear>1958
  </birthyear>
 </person>
 <person gender="female"
  status="married">
  <name>Eliza
  Tennyson</name>
  <birthyear>1959
  </birthyear>
 </person>
</family>
</pre></td></tr>
<tr><td>169 octets (excluding newlines)</td><td>54 octets</td><td>258 octets (excluding newlines and leading spaces)</td></tr>
<tr><td>18 % efficiency[1]</td><td>57 % efficiency[1]</td><td>12 % efficiency[1]</td></tr>
</table>

[1] As compared to PER.

The PER (unaligned variant) encoding of the same ASN.1 and the same data would be the following 31 octets:

00000010	(number of persons in family)	000011 10	(14 characters)
00001010	(10 characters)	100010 1	E
1001010	J	1101100	l
1 101111	o	1101001	i
11 01000	h	1 111010	z
110 1110	n	11 00001	a
0100 000		010 0000	
10100 11	S	1010 100	T
110110 1	m	11001 01	e
1101001	i	110111 0	n
1110100	t	1101110	n
1 101000	h	1111001	y
00 000010	(2 octets)	1 110011	s
00 00011110	100110 (1958)	11 01111	o
0	(male)	110 1110	n
0 01	(married)	0000 0010	(2 bytes)
		0000 01111010 0111	(1959)
		1	(female)
		001	(married)

Note 1: Many thanks to Jean-Paul Lemaire, who helped me with the BER and PER encodings.

Note 2: The success of many Internet application layer protocols with very inefficient textual encodings apparently indicates that the efficiency is not a very important factor in determining the success of an application layer protocol.

Note 3: Compression programs (like zip, gz, etc.) can compress almost any textual encoding to near-maximal efficiency. This, however, only works for large files. Small files are not compressed very efficiently with compression programs. To test this, I tried to compress the XML encoding above using the Zip encoding. It actually becaome 14 % larger after compression. I also tested a file where I repeated the XML encoding above 11 times, with the same XML elements and tags, but different content. This larger file, after compression with Zip encoding, became 53 % as efficient as the PER encoding, or about as high efficiency as with the BER encoding.

Table 2: Comparison of ABNF, ASN.1-BER and DTD-XML

	ABNF	ASN.1	DTD+XML
Level	Low level, can specify almost any textual encoding.	High level, strongly typed, you define the exact data types to use .	High level, but not as good type facilities as ASN.1.
Encoded format	Text.	With for example Basic Encoding Rules (BER), a binary format, or Packed Encoding Rules (PER), a very efficient binary format, or other encoding rules.	Text.
Readability of meta-language	OK.	Good.	Acceptable.
Readability of encoded data	Very good.	Very bad unless special reader program is used.	Very good.
Efficiency of data packing, as compared to maximum efficiency.	Usually not so good.	About 50 % with BER, almost 100 % with PER.	Not so good.
Binary data	Must be encoded, for example using BASE64, which however adds 33 % redundancy.	Can easily be included as is.	Must be encoded, for example using BASE64, or sent as separate files.
Layout facilities	None, but the high freedom allows specification of rather readable formats.	None.	Can be combined with layout languages to produce highly readable output (comparable to HTML-based web documents).

1.1.1. Comparing RFC822-style headings with XML and ASN.1

Many standards have used the so-called RFC822-style header format, which is usually specified using ABNF. Below is an example of how the same information can be encoded in this format as compared to XML:

RFC822 example (54 characters):

```
From: Father Christmas <fchristmas@northpole.arctic>
```

XML encoding of the same information (248 characters):

```
<from>
  <user-friendly-name>Father Christmas</user-friendly-name>
  <e-mail-address>
    <localpart>fchristmas</localpart>
    <domainpart>
      <domainelement>northpole</domainelement>
      <domainelement>arctic</domainelement>
  </domainpart>
</from>
```

Besides noting that XML in this example requires about five times as many characters, another difference is that XML uses the same characters for framing in all levels, while the RFC822 example uses three different notations in five levels:

Level 1: Newline between headers.

Level 2: ":" between header name and header value.

Level 3: "<" and ">" to separate localpart from e-mail address.

Level 4: "@" to separate localpart from domainlist.

Level 5: “.” to separate the domain component in the list of domain elements.

It is of course an advantage with XML that you do not have to invent new framing characters at each level, and also maybe new rules about forbidden characters or characters that need to be quoted at each level.