Overview of the Octet Encoding Rules (OER)

Alessandro Triglia, OSS Nokalva

January 2015

CONTENTS

1 Introduction

This paper is a technical overview of the Octet Encoding Rules (OER), a new, efficient set of encoding rules for ASN.1, and the most recent member of the family of ASN.1 standards developed by ISO/IEC and ITU-T.

OER was published in 2014 as ISO/IEC 8825-7 | Rec. ITU-T X.696 and can be downloaded for free from the ITU-T website (http://www.itu.int/rec/T-REC-X.696-201408-I).

OER was designed to be easy to implement and to produce encodings more compact than those produced by the Basic Encoding Rules (BER). In addition to reducing the effort of developing encoder/decoders, the use of OER can decrease bandwidth utilization (though not as much as the Packed Encoding Rules), save CPU cycles, and lower encoding/decoding latency.

A previous version of OER had been published by AASHTO, ITE and NEMA as NTCIP 1102:2004, and is used by a group of NTCIP standards. The new, international version of OER produces encodings that are identical to those produced by NTCIP 1102 when considering only the features of ASN.1 that are used in those NTCIP standards.

This paper describes the OER encoding of a few major ASN.1 types.

2 The Octet Encoding Rules

The Octet Encoding Rules (OER), like the Packed Encoding Rules (PER), produce compact encodings by taking advantage of information present in the ASN.1 schema to limit the amount of information included in each encoded message. However, in contrast to PER, OER favors encoding/decoding speed and ease of implementation over compactness of the encodings.

Unlike PER Aligned, OER is completely octet-oriented. Whereas in PER Aligned a fixed-size encoding occupying less than 8 bits may begin at any bit position within an octet and may end at any bit position within the same or another octet, in OER the encoding of every possible value of every ASN.1 type occupies a whole number of octets, and all the fields of the encoding are octet-aligned.

Another characteristic of OER is the fact that most integer types are encoded in **one**, **two**, **four**, or **eight octets**, either **signed** or **unsigned** (depending on the value range constraint specified for the type) and most enumerated types are encoded in **one** octet. Since integer types and enumerated types are a significant fraction of the elementary types that occur in an ASN.1 schema, OER encoder/decoder implementations can be simpler and faster compared to other encoding rules.

Still, OER does not reach the degree of compactness of PER (especially PER Unaligned). OER uses 8 bits in many places where PER Unaligned might use 1 to 7 bits, 16 bits in many places where PER Unaligned might use 9 to 15 bits, and 32 or 64 bits in many places where PER Unaligned might use fewer bits. When compactness is more valuable than encoding/decoding speed (e.g., when the endpoints have many spare CPU cycles but bandwidth is scarce or expensive), PER may be more adequate than OER.

The OER standard specifies two versions of OER: *Basic OER* and *Canonical OER*. In Canonical OER, as in all the other canonical ASN.1 encoding rules (DER, Canonical PER, etc.), each possible value of a type is encoded in exactly one way. This is not true, in general, in Basic OER, as in some cases there are two or more slightly different ways to encode the same value. One might consider the use of Canonical OER as a faster alternative to DER when a strict one-to-one correspondence between abstract values and encodings is required (e.g., in security applications).

The following sections describe the OER encoding of a few major ASN.1 types.

2.1 Encoding of a boolean type

A boolean type is encoded in a single octet, which is set to zero for the value `FALSE` and any nonzero binary value for the value `TRUE`.

2.2 Encoding of an integer type

The encoding of an integer type depends on the value range constraint present in the type definition (if any). Some integer types are encoded as a **variable-size** integer field preceded by a ***length prefix***; others are encoded as a **fixed-size integer** field occupying 1, 2, 4, or 8 octets. The rules are the following:

1) if the lower bound of the value range constraint is not less than 0 and the upper bound is not greater than 255 and the constraint is not extensible, the integer value is encoded as an **unsigned** binary integer in **one octet**;
2) otherwise, if the lower bound is not less than 0 and the upper bound is not greater than 65535 and the constraint is not extensible, the integer value is encoded as an **unsigned** binary integer in **two octets**;
3) otherwise, if the lower bound is not less than 0 and the upper bound is not greater than 4294967295 and the constraint is not extensible, the integer value is encoded as an **unsigned** binary integer in **four octets**;
4) otherwise, if the lower bound is not less than 0 and the upper bound is not greater than 18446744073709551615 and the constraint is not extensible, the integer value is encoded as an **unsigned** binary integer in **eight octets**;
5) otherwise, if the lower bound is not less than 0, the integer value is encoded as a ***length prefix*** as described in 2.11, followed by an unsigned binary integer occupying a **variable number of octets**; the *length prefix* contains the number of subsequent octets;
6) otherwise, if the lower and the upper bound are both in the range -128 to 127 and the constraint is not extensible, the integer value is encoded as a **signed** binary integer in **one** octet;
7) otherwise, if the lower and the upper bound are both in the range -32768 to 32767 and the constraint is not extensible, the integer value is encoded as a **signed** binary integer in **two** octets;
8) otherwise, if the lower and the upper bound are both in the range -2147483648 to 2147483647 and the constraint is not extensible, the integer value is encoded as a **signed** binary integer in **four** octets;
9) otherwise, if the lower and the upper bound are both in the range -9223372036854775808 to 9223372036854775807 and the constraint is not extensible, the integer value is encoded as a **signed** binary integer in **eight** octets;
10) otherwise, the integer value is encoded as a ***length prefix*** as described in 2.11, followed by a signed binary integer occupying a **variable number of octets**; the *length prefix* contains the number of subsequent octets.

All integer encodings are in big-endian order.

2.3 Encoding of an enumerated type

An enumerated type is encoded by encoding the **number** associated with the enumerator. (This differs from PER, where a consecutive index is assigned to each enumerator and the index goes into the encoding.) Extensible and non-extensible enumerated types are encoded in the same way. The rules are the following:

1) if the number associated with the enumerator is in the range 0 to 127, it is encoded as a **one-octet** unsigned integer, with its highest-order bit set to zero;
2) otherwise, the number associated with the enumerator is encoded as **a signed** binary integer occupying a **variable number of octets** in big-endian order, preceded by a one-octet **length prefix**; the length prefix contains the number of subsequent octets and has its highest-order bit set to one.

2.4 Encoding of a real type

OER is the first standard set of ASN.1 encoding rules that uses the IEEE 754 floating-point format to encode real types. This encoding is very efficient because in many cases the in-memory (unencoded) value is in the same format. However, the IEEE format is used only when the real type is constrained in such a way that all the possible values are representable in that format.

For example, the following real types are encoded in the IEEE floating-point format:

```
REAL (0 | WITH COMPONENTS { mantissa (-99999..99999),
      base (2), exponent (-20..20)})  --single precision

REAL (0 | WITH COMPONENTS { mantissa (-999999999999..999999999999),
      base (2), exponent (-20..20)})  --double precision
```

In contrast, the following real type is encoded in OER in the same way it is encoded in DER and PER:

```
REAL
```

2.5 Encoding of an IA5String or VisibleString (ASCII character string) type

The encoding of an `IA5String` or `VisibleString` type depends on the size constraint present in the type definition (if any). The rules are the following:

1) if the `IA5String` or `VisibleString` type has a non-extensible size constraint specifying a single (fixed) size, the encoding of the string value consists just of the encodings of the characters (there is **no *length prefix***);
2) otherwise, the encoding of the string value consists of a ***length prefix*** as described in 2.11, followed by the encodings of the characters; the *length prefix* contains the number of subsequent octets.

In both cases, each character of the string is encoded in one octet (the highest-order bit is set to zero).

2.6 Encoding of an octet string type

The encoding of an octet string type depends on the size constraint present in the type definition (if any). The rules are the following:

1) if the octet string type has a non-extensible size constraint specifying a single (fixed) size, the encoding of the octet string value consists just of the concatenation of its octets (there is **no *length prefix***);
2) otherwise, the encoding of the octet string value consists of a ***length prefix*** as described in 2.11, followed by the octets of the string.

2.7 Encoding of a bit string type

The encoding of a bit string type depends on the size constraint present in the type definition (if any). The rules are the following:

1) if the bit string type has a non-extensible size constraint specifying a single (fixed) size, the encoding of the bit string value consists only of the octets containing the bits of the value (see below); there is **no *length prefix*** and no indication of how many unused bits there are in the last octet;
2) otherwise, the encoding of the bit string value consists of a ***length prefix*** as described in 2.11 followed by a ***unused-bit count prefix*** followed by the octets containing the bits of the value (see below); the *unused-bit count prefix* is one octet containing the number of unused bit positions in the last octet (between 0 and 7).

In both cases, the bits of the bit string value are written into zero or more consecutive octets, starting from the highest-order bit position of the first octet, continuing at the highest-order bit position of each subsequent octet, and ending at some bit position within the last octet. Any unused bits in the last octet are set to zero. The length prefix indicates the total number of following octets (including the octet containing the *unused-bit count prefix*), not the number of bits in the bit string value. Thus an empty bit string has a length prefix of 1 followed by one octet (the *unused-bit count prefix*) set to 0.

2.8 Encoding of a sequence type

The encoding of a sequence type in OER is very similar to the encoding of this type in PER, both for extensible and for non-extensible sequence types.

A non-extensible sequence type is encoded as a ***presence bitmap*** followed by the concatenation of the encodings of the fields of the sequence type that are present in the value, in specification order. The *presence bitmap* is encoded as a bit string with a fixed size constraint (see 2.7, case 1), and has one bit for each field of the sequence type that has the keyword `OPTIONAL` or `DEFAULT`, in specification order.

An **extensible** sequence type is encoded in a similar way, except for the following:

- the *presence bitmap* includes an ***extension bit*** (in the first position) which indicates whether any extensions are present in the sequence value (the other bits of the *presence bitmap* are displaced by one position);
- when the *extension bit* is on, an ***extension presence bitmap*** is included before the first extension; the *extension presence bitmap* is encoded as a bit string with no size constraint (see 2.7, case 2), and has one bit for each extension specified in the sequence type definition; each bit indicates whether the corresponding extension is present in the sequence value;
- each extension that is present in the sequence value is first encoded in OER, producing a series of octets, and then that series of octets is re-encoded as an **octet string** (see 2.6).

Each part of the encoding (the *presence bitmap*, the *extension presence bitmap*, each component, and each extension) occupies a whole number of octets (zero or more).

2.9 Encoding of a sequence-of type

A sequence-of type is encoded as a ***quantity*** field followed by the encoding of each occurrence of the repeating field of the sequence-of type (zero or more). Extensible and non-extensible sequence-of types are encoded in the same way. The *quantity* field is set to the number of occurrences (not to the number of octets), and is encoded as an integer type with a lower bound of zero and no upper bound (see 2.2, case 4).

2.10 Encoding of a choice type

A choice type is encoded as the encoding of the **tag** of the chosen alternative (see 2.12), followed by the encoding of the chosen alternative. This is the only case in which tags are encoded in OER.

2.11 Encoding of a *length prefix*

The *length prefix* that is part of the encoding of certain ASN.1 types (see the sections referencing this section) specifies the length of an encoding in terms of **number of octets**. A *length prefix* consists of one or more octets, as follows:

1) If the *length* is less than 128, it is encoded in the lowest-order 7 bit positions of the first and only octet, and the highest-order bit of the octet is set to zero.
2) If the *length* is greater than 127, it is encoded in one or more subsequent octets, in big-endian order. The number of those octets is indicated in the lowest-order 7 bit positions of the first octet, and the highest-order bit of that octet is set to 1.

2.12 Encoding of a tag

In OER, a tag is encoded only when it identifies the chosen alternative of a choice type. In this case, a tag is encoded into one or more *identifier octets*, as follows:

1) the first octet has its highest-order two bits set to '00'B for a tag of the *universal* tag class, '01'B for a tag of the *application* tag class, '10'B for a tag of the *context-specific* tag class, and '11'B for a tag of the *private* tag class;
2) if the tag number is less than 63, it is encoded in the lowest-order 6 bit positions of the first and only octet;
3) if the tag number is greater than 62, the lowest-order 6 bits of the first octet are set to '111111'B; the tag number occupies the lowest-order 7 bit positions of each subsequent octet, in big-endian order; the highest-order bit of each of those octets except the last is set to 1, and the highest-order bit of the last octet is set to 0.

3 Example

Suppose we have the following ASN.1 module:

```
MyModule DEFINITIONS AUTOMATIC TAGS ::= BEGIN

A ::= SEQUENCE {
      a1    INTEGER (0..100),
      a2    INTEGER (-290..399),
      a3    INTEGER (0..60000)      OPTIONAL,
      a4    INTEGER (-5000000..5000000),
      a5    INTEGER (1000..MAX),
      a6    INTEGER (-1..MAX),
      a7    INTEGER                 OPTIONAL
}

a A ::= {
      a1 4,
      a2 4,
      a3 4,   --optional and present
      a4 4,
      a5 1024,
      a6 4,
      a7 4    --optional and present
}

B ::= SEQUENCE {
      b1    IA5String (SIZE (0..10)),
      b2    IA5String (SIZE (3)),
      b3    IA5String,
      b4    OCTET STRING,
      b5    BIT STRING (SIZE (4)),
      b6    BIT STRING
}

b B ::= {
      b1 "ABC",
      b2 "ABC",
      b3 "ABC",
      b4 '01020304'H,
      b5 '0101'B,
      b6 '0101'B
}

C ::= CHOICE {
      c1    BOOLEAN,
      c2    SEQUENCE OF ENUMERATED { a, b, c, d, e }
}
```

```
c C ::= c2 : { b, c, d, e }

END
```

In OER, the value a is encoded as follows:

		Series of octets	Notes
Presence bitmap		'1100 0000'B	• the two optional fields of this sequence type (a3 and a7) are both present • the lowest-order 6 bits of this octet are not used
Content octets	**Encoding of the a1 field**		
	Content octets	'04'H	• 8-bit unsigned integer
	Encoding of the a2 field		
	Content octets	'00 04'H	• 16-bit signed integer, big-endian
	Encoding of the a3 field		
	Content octets	'00 04'H	• 16-bit unsigned integer, big-endian
	Encoding of the a4 field		
	Content octets	'00 00 00 04'H	• 32-bit signed integer, big-endian
	Encoding of the a5 field		
	Length octets	'02'H	• length=2 octets
	Content octets	'04 00'H	• variable-size unsigned integer, big-endian
	Encoding of the a6 field		
	Length octets	'01'H	• length=1 octet
	Content octets	'04'H	• variable-size signed integer, big-endian
	Encoding of the a7 field		
	Length octets	'01'H	• length=1 octet
	Content octets	'04'H	• variable-size signed integer, big-endian

In OER, the value `b` is encoded as follows:

		Series of octets	Notes
Content octets	**Encoding of the `b1` field**		
	Length octets	'03'H	• length=3 octets
	Content octets	'41 42 43'H	• three characters
	Encoding of the `b2` field		
	Content octets	'41 42 43'H	• three characters
	Encoding of the `b3` field		
	Length octets	'03'H	• length=3 octets
	Content octets	'41 42 43'H	• three characters
	Encoding of the `b4` field		
	Length octets	'04'H	• length=4 octets
	Content octets	'01 02 03 04'H	• four octets
	Encoding of the `b5` field		
	Content octets	'01010000'B	• four significant bits ('0101'B) • four unused bits set to zero
	Encoding of the `b6` field		
	Length octets	'02'H	• length=2 octets
	Unused-bit count	'04'H	• four unused bits in the last octet
	Content octets	'0101 0000'B	• four significant bits ('0101'B) • four unused bits set to zero

In OER, the value c is encoded as follows:

			Series of octets	Notes
Identifier octets			'1000 0001'B	• The first two bits ('10'B) denote a *context-specific* tag • The next six bits ('000001'B) identify the tag number 1
Content octets	**Encoding of the c2 field**			
	Quantity	**Length octets**	'01'H	• Length=1 octet
		Content octets	'04'H	• There are four occurrences of the repeating field
	Content octets	**Encoding of the first occurrence**		
		Content octets	'01'H	• enumerator number 1
		Encoding of the second occurrence		
		Content octets	'02'H	• enumerator number 2
		Encoding of the third occurrence		
		Content octets	'03'H	• enumerator number 3
		Encoding of the fourth occurrence		
		Content octets	'04'H	• enumerator number 4