OSS Nokalva

ASN.1, A SCHEMA LANGUAGE FOR XML

OSS Nokalva, Inc., April 2004.

OSS Nokalva

CONTENTS:

1 INTRODUCTION

The purpose of this document is to provide a basic understanding of the Extended XML Encoding Rules of ASN.1 (EXTENDED-XER). The meaning of XER encoding instructions and their effect on EXTENDED-XER encodings is shown by using examples. This document is not intended to provide a formal (or even semi-formal) description of EXTENDED-XER and of XER encoding instructions. For a formal description please refer to ITU-T Rec. X.693 | ISO/IEC 8825-4 Amd. 1.

A value notation "value" is provided in each of the examples in this document. All the examples show the EXTENDED-XER (hereafter called E-XER) encoding of a "value" together with the BASIC-XER encoding of the same "value". The BASIC-XER encodings are provided in order to show how the EXTENDED-XER encodings differ from them. In these examples, use of automatic tagging is assumed (the clause AUTOMATIC TAGS present in the module header). The clause XER INSTRUCTIONS is assumed to be present in the module header as well.

```
M DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::= BEGIN
...
...
END
```

There are two ways of assigning an encoding instruction to an ASN.1 type. It can be assigned either in-line in the type definition, or in a separate encoding control section.

1. 1 Assigning XER encoding instructions using a type prefix

The following example shows how an encoding instruction can be applied to an ASN.1 type using a type prefix. In this example, the ATTRIBUTE encoding instruction is assigned to the component "name" of type "Product", and the LIST encoding instruction is assigned to the component "availableColors" of type "Product".

```
X1 DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::= BEGIN
Product ::= SEQUENCE {
    name             [ATTRIBUTE] IA5String (FROM ("a".."z")),
    availableColors [LIST] SEQUENCE OF IA5String (FROM ("a".."z")),
    price            INTEGER
}
END
```

1.2 Assigning XER encoding instructions using an encoding control section

Alternatively, encoding instructions can be assigned to the ASN.1 types using an XER encoding control section. The following example is similar to the previous example. However, in this example, the encoding instruction is written in an encoding control section.

```
X1 DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::= BEGIN
Product ::= SEQUENCE {
    name IA5String (FROM ("a".."z")),
    availableColors SEQUENCE OF IA5String (FROM ("a".."z")),
    price INTEGER
```

```
}
ENCODING-CONTROL XER
    ATTRIBUTE name IN Product
    LIST availableColors IN Product
END
```

An alternative way of writing the above syntax using an encoding control section is the following:

```
X1 DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::= BEGIN
Product ::= SEQUENCE {
    name IA5String (FROM ("a".."z")),
    availableColors SEQUENCE OF IA5String (FROM ("a".."z")),
    price INTEGER
}
ENCODING-CONTROL XER
    ATTRIBUTE Product.name
    LIST Product.availableColors
END
```

In the encoding control section, built-in ASN.1 type names can also be used as targets of encoding instructions. This will cause the encoding instruction to be applied to all ASN.1 types of that built-in type. In the following example, the ATTRIBUTE encoding instruction is applied to all the ASN.1 INTEGER and REAL types in the module, and the DECIMAL encoding instruction is applied to all the ASN.1 REAL types in the module.

Example 1:

```
Product ::= SEQUENCE {
    size  INTEGER,
    price REAL (WITH COMPONENTS {..., base(10)})
              (ALL EXCEPT (-0 | MINUS-INFINITY | PLUS-INFINITY | NOT-A-NUMBER))
}

value Product ::= { size 44, price 19.95 }

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
    ATTRIBUTE INTEGER, REAL
    DECIMAL REAL
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product size="44" price="19.95"/>
```

Example 2:

```
List ::= SEQUENCE OF [UNTAGGED] SEQUENCE {
    name IA5String,
    color ENUMERATED {red(0), green(1), blue(2), yellow(3), black(4)}
}

value List ::= {
    {name "shirt", color red},
    {name "shoes", color black},
    {name "trousers", color yellow}
```

```
}

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
    TEXT List.*.color:red, List.*.color:green, List.*.color:blue AS CAPITALIZED
    TEXT List.*.color:black, List.*.color:yellow AS UPPERCASED
```

E-XER encoding:

```
<List>
  <name>shirt</name>
  <color>Red</color>
  <name>shoes</name>
  <color>BLACK</color>
  <name>trousers</name>
  <color>YELLOW</color>
</List>
```

1.3 Negating Encoding Instructions

The assignment of an encoding instruction can be cancelled by using the `NOT` keyword. In following example the field `"color"` of type `"Shirt"` is an attribute, while for type `"Shoes"`, it is not an attribute. The `ELEMENT` encoding instruction is a synonym for `NOT UNTAGGED`. It cannot be cancelled using a NOT keyword.

```
ColorType ::= IA5String (FROM("a".."z"))

Shirt ::= SEQUENCE {
    color ColorType,
    size  INTEGER
}

Shoes ::= SEQUENCE {
    color ColorType,
    size  INTEGER
}

value Shirt ::= { color "red", size 44 }
anotherValue Shoes ::= { color "black", size 10 }

ENCODING-CONTROL XER
    ATTRIBUTE ALL IN ALL
    NOT ATTRIBUTE Shoes.color
```

E-XER encoding of `value`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Shirt color="red" size="44"/>
```

E-XER encoding of `anotherValue`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Shoes size="10">
  <color>black</color>
```

```
</Shoes>
```

1.4 What are character-encodable types?

The term “character-encodable” is used in many places in this document. An ASN.1 type is called character-encodable if the encodings of all its possible values consist of a sequence of legal characters and does not include child elements (see ITU-T Rec. X.693 | ISO/IEC 8825-4 Amd. 1, 3.2.2 ter for a more-precise definition of "character-encodable"). A character is an atomic unit of text as specified by ISO/IEC 10646. Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646.

There are certain XER encoding instructions that can only be assigned to character-encodable ASN.1 types. For example, the ATTRIBUTE encoding instruction can only be assigned to character-encodable types. Similarly, the LIST encoding instruction can only be applied to “SEQUENCE OF”/”SET OF” types where the component of the “SEQUENCE OF”/”SET OF” type is a character-encodable type. Finally, the USE-UNION encoding instruction can only be applied to choice types where all the alternatives of the choice are character-encodable.

It is recommended to define a separate character-encodable type, so that the character-encodable constraint definition is not repeated too many times in your ASN.1 syntax. For example, you can create another type “XMLCompatibleString” as shown below, and use this type in your ASN.1 syntax wherever a character-encodable type is needed.

Example:

```
XMLCompatibleString ::= UTF8String (FROM(
        {0, 0, 0, 9} |
        {0, 0, 0, 10} |
        {0, 0, 0, 13} |
        {0, 0, 0, 32} .. {0, 0, 215, 255} |
        {0, 0, 224, 0} .. {0, 0, 255, 253} |
        {0, 1, 0, 0} .. {0, 16, 255, 253}))

Attr ::= XMLCompatibleString

Color ::= XMLCompatibleString ("red" | "blue" | "green")

S ::= [LIST] SEQUENCE OF Color

C ::= [USE-UNION] CHOICE {
    c1  [0] XMLCompatibleString,
    c2  [1] XMLCompatibleString,
    c3  [1] XMLCompatibleString,
    c4  [2] XMLCompatibleString
}
```

1.5 Tagging and E-XER encoding instructions

Like E-XER encoding instructions, tags (combinations of a tagging class and a tag number) are also specified with in square brackets (“[]”) as type prefixes. In order to differentiate tags from E-XER encoding instructions, tags must be prefixed with the “TAG” keyword and a colon (“:”). In the following example, the second ASN.1 syntax is enhanced to declare type “A” and “B” as attributes for E-XER encoding.

Example:

```
M DEFINITIONS ::= BEGIN
A ::= [10] VisibleString
B ::= [PRIVATE 11] VisibleString
S ::= SEQUENCE {
    a A,
    b B
}
END

M DEFINITIONS XER INSTRUCTIONS ::= BEGIN
A ::= [TAG: 10] [ATTRIBUTE] VisibleString
B ::= [TAG: PRIVATE 10] [ATTRIBUTE] VisibleString
S ::= SEQUENCE {
    a A,
    b B
}
END
```

2 ANY-ATTRIBUTES

This encoding instruction allows flexibility as to what attributes may appear in the E-XER encoding. A namespace restriction, using the FROM and EXCEPT clauses followed by a list of URIs, may also be added in encoding instruction to specify what namespaces the attribute names may belong to.

The FROM clause restricts attribute names to be qualified from one of the namespaces specified in the list of URIs. The keyword ABSENT may be added in the list of URIs to allow unqualified attributes. The attribute "available" in Example-1 doesn't belong to any namespace.

The EXCEPT clause allows attribute names to be qualified from any namespace except those are in the URI list. If the keyword ABSENT is present in the URI list, unqualified attribute names cannot be used. In Example-2, since ABSENT is present in the list of URIs, the attribute "available" must be qualified.

This encoding instruction can only be assigned to SEQUENCE OF or SET OF types with a UTF8String component, whose value provides zero, one, or more attribute names and values (one in each UTF8String), each of which is subject to any namespace restriction that is present.

Example 1:

```
AttrList ::= SEQUENCE OF UTF8String

Product ::= SEQUENCE {
    info        [ANY-ATTRIBUTES FROM "http://www.example.com/A"
                "http://www.example.com/B" ABSENT] AttrList (CONSTRAINED BY {}),
    name        VisibleString,
    price       INTEGER
}

value Product ::= {
    info {"http://www.example.com/A size=""small""",
        "http://www.example.com/B color=""red""",
        "available=""yes"""},
    name "Trousers",
```

```
    price 20
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product
  xmlns:b0="http://www.example.com/A" b0:size="small"
  xmlns:b1="http://www.example.com/B" b1:color="red" available="yes">
  <name>Trousers</name>
  <price>20</price>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <info>
    <UTF8String>http://www.example.com/A size="small"</UTF8String>
    <UTF8String>http://www.example.com/B color="red"</UTF8String>
    <UTF8String>available="yes"</UTF8String>
  </info>
  <name>Trousers</name>
  <price>20</price>
</Product>
```

Example 2:

```
AttrList ::= SEQUENCE OF UTF8String

Product ::= SEQUENCE {
    info        [ANY-ATTRIBUTES EXCEPT "http://www.example.com/C" ABSENT]
                AttrList (CONSTRAINED BY {}),
    name        VisibleString,
    price       INTEGER
}

value Product ::= {info {"http://www.example.com/A size=""small""",
        "http://www.example.com/B color=""red""",
        "http://www.example.com/B available=""yes"""}, name "Trousers", price 20}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product xmlns:b0="http://www.example.com/A"
  b0:size="small" xmlns:b1="http://www.example.com/B" b1:color="red"
  b1:available="yes">
  <name>Trousers</name>
  <price>20</price>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <info>
```

```
    <UTF8String>http://www.example.com/A size="small"</UTF8String>
    <UTF8String>http://www.example.com/B color="red"</UTF8String>
    <UTF8String>available="yes"</UTF8String>
  </info>
  <name>Trousers</name>
  <price>20</price>
</Product>
```

3 ANY-ELEMENT

This encoding instruction enables a UTF8String type to provide the specification of a single XML element.

The FROM clause restricts the element name to be qualified from one of the namespaces specified in the list of URIs. The keyword ABSENT can be added in the list of URIs to allow unqualified elements.

On the contrary, the EXCEPT clause allows the element name to be qualified from any namespace except those are in the list of URIs. If ABSENT is present in the list of URIs, unqualified element names cannot be used. In Example-2, since ABSENT is present in the list of URIs, the element "color" must be qualified.

Example 1:

```
Product ::= SEQUENCE {
    name    IA5String,
    price   INTEGER,
    info    [ANY-ELEMENT FROM "http://www.example.com/A"
"http://www.example.com/B" ABSENT] UTF8String (CONSTRAINED BY {})
}

value Product ::= { name "Trousers",
    price 20,
    info "<xyz:color xmlns:xyz=""http://www.example.com/A"" available=""true"">r
ed</xyz:color>"
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>Trousers</name>
  <price>20</price>
  <xyz:color xmlns:xyz="http://www.example.com/A" available="true">red</xyz:color>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>Trousers</name>
  <price>20</price>
  <info>&lt;xyz:color xmlns:xyz="http://www.example.com/A"
available="true"&gt;red&lt;/xyz:color&gt;</info>
</Product>
```

Example 2:

```
Product ::= SEQUENCE {
    name    IA5String,
    price   INTEGER,
    info    [ANY-ELEMENT EXCEPT "http://www.example.com/A" "http://www.example.com
/B" ABSENT] UTF8String (CONSTRAINED BY {})
}

value Product ::= { name "Trousers",
    price 20,
    info "<xyz:color xmlns:xyz=""http://www.example.com/C"" available=""true"">r
ed</xyz:color>"
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>Trousers</name>
  <price>20</price>
  <xyz:color xmlns:xyz="http://www.example.com/C" available="true">red</xyz:color>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>Trousers</name>
  <price>20</price>
  <info>&lt;xyz:color xmlns:xyz="http://www.example.com/C"
available="true"&gt;red&lt;/xyz:color&gt;</info>
</Product>
```

4 ATTRIBUTE

This encoding instruction specifies that a “character-encodable” ASN.1 type is to be encoded as an XML attribute. An ASN.1 type with this encoding instruction can only be used as a component of SEQUENCE or SET type. The type must be constrained so that it allows only those values which are valid as XML attribute values. For example, there must not be any control characters or XML tags in any of its abstract values.

If this encoding instruction is applied to an ASN.1 type, the type must not have the UNTAGGED encoding instruction also applied to it, or applied to the enclosing type that contains it as a component.

Example :

```
Color ::= [ATTRIBUTE] IA5String (FROM ("a".."z"))

Product ::= SEQUENCE {
    color   Color,
    available [ATTRIBUTE] UTF8String ("true" | "false"),
    name    IA5String,
    price   INTEGER
}
```

```
value Product ::= {
    color "red",
    available "false",
    name "Shoes",
    price 25
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product color="red" available="false">
  <name>Shoes</name>
  <price>25</price>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <color>red</color>
  <available>false</available>
  <name>Shoes</name>
  <price>25</price>
</Product>
```

5 BASE64

This encoding instruction can be assigned to an `OCTET STRING`, to an open type, or to any restricted character string type. Application of this encoding instruction removes the option of a hexadecimal encoding, but allows the option of a base64 encoding.

Example:

```
T1 ::= SEQUENCE {
    comp [BASE64] OCTET STRING
}

value T1 ::= {
    comp '0123456789ABCDEF0123456789ABCDEF'H
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<T1>
  <comp>ASNFZ4mrze8BI0VniavN7w==</comp>
</T1>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<T1>
  <comp>01234567 89ABCDEF 01234567 89ABCDEF</comp>
```

```
</T1>
```

6 DECIMAL

The purpose of this encoding instruction is to modify the encoding of a REAL type so that the exponential notation is forbidden and a hyphen followed by "0" denotes the value plus zero instead of the value minus zero. The type must be restricted so that the values minus 0, `MINUS-INFINITY`, `PLUS-INFINITY`, and `NOT-A-NUMBER` are not permitted and the base is 10.

Example:

```
Product ::= SEQUENCE {
    color   [ATTRIBUTE] UTF8String (FROM ("a".."z")),
    name    IA5String,
    price   [DECIMAL] REAL (WITH COMPONENTS {..., base(10)})
                (ALL EXCEPT (-0 | MINUS-INFINITY | PLUS-INFINITY | NOT-A-NUMBER)
)
}

value Product ::= {
    color "red",
    name "shirt",
    price 12.33
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product color="red">
  <name>shirt</name>
  <price>12.33</price>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <color>red</color>
  <name>shirt</name>
  <price>1.233E1</price>
</Product>
```

7 DEFAULT-FOR EMPTY

This encoding instruction specifies an abstract value that can be encoded as an empty element in the `E-XER` encoding. It is different from the use of `ASN.1 DEFAULT`, which relates to the absence of a component of a `SEQUENCE` or `SET`.

In the following example, the value of `currency` is "`US Dollars`". That is why it is encoded as an empty element by the encoder.

Example:

```
Product ::= SEQUENCE {
    color    [ATTRIBUTE] UTF8String (FROM ("a".."z")),
    name     IA5String,
    price    [DECIMAL] REAL (WITH COMPONENTS {..., base(10)})
             (ALL EXCEPT (-0 | MINUS-INFINITY | PLUS-INFINITY | NOT-A-NUMBER)),
    currency [DEFAULT-FOR-EMPTY AS "US Dollars"] UTF8String (FROM ("A".."Z" |
"a".."z" | " "))
}


value Product ::= {
    color "red",
    name "shirt",
    price 12.33,
    currency "US Dollars"
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product color="red">
  <name>shirt</name>
  <price>12.33</price>
  <currency/>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <color>red</color>
  <name>shirt</name>
  <price>1.233E1</price>
  <currency>US Dollars</currency>
</Product>
```

8 ELEMENT

This encoding instruction only negates an `UNTAGGED` encoding instruction, and doesn't otherwise affect encodings. This encoding instruction should not be used with `ANY-ATTRIBUTES`, `ANY-ELEMENT`, or `ATTRIBUTE` instructions to avoid any confusion.

Example:

```
I ::= [UNTAGGED] INTEGER
S ::= [ELEMENT] I

value S ::= 20

ENCODING-CONTROL XER
```

```
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>20</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>20</S>
```

9 EMBED-VALUES

This encoding instruction is used to produce mixed content in the E-XER encodings. It enables the first component of a sequence type to provide character strings to be inserted before the first XML element, after the last XML element, and between the XML elements. The `SEQUENCE` must not have an `UNTAGGED` encoding instruction.

This encoding instruction cannot be assigned unless there is a `GLOBAL-DEFAULTS MODIFIED-ENCODINGS` encoding instruction in the encoding control section.

E-XER encoding:

```
Product ::= [EMBED-VALUES] SEQUENCE {
    embed-values SEQUENCE OF UTF8String,
    companyName UTF8String,
    productColor UTF8String,
    productName UTF8String
} (CONSTRAINED BY {})

value Product ::= {
  embed-values {"My Company", "produces", "", "which is very popular"},
  companyName "ABC",
  productColor "red",
  productName "shirt"
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>My
Company<companyName>ABC</companyName>produces<productColor>red</productColor><prod
ctName>shirt</productName>which is very popular</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Product>
  <embed-values>
    <UTF8String>My Company</UTF8String>
    <UTF8String>produces</UTF8String>
    <UTF8String/>
    <UTF8String>which is very popular</UTF8String>
  </embed-values>
  <companyName>ABC</companyName>
  <productColor>red</productColor>
  <productName>shirt</productName>
</Product>
```

10 GLOBAL-DEFAULTS

This encoding instruction serves two purposes. First, it can be used to specify a control namespace. The namespace of the type identification attribute and of the nil identification attribute is the control namespace. This is, by default, the ASN.1 namespace, unless a different namespace is specified by using this encoding instruction (see Example-1 and Example-2). The ASN.1 namespace has a namespace name of `"urn:oid:2.1.5.2.0.1"` and a recommended namespace prefix of `"asn1".`

The other purpose of the `GLOBAL-DEFAULTS` encoding instruction (when used with the keyword `MODIFIED-ENCODINGS`) is to change the E-XER encodings in the following ways:

- for `BOOLEAN` types, it causes the values to be encoded as “true” or “false” instead of the empty element tags <true/> and <false/> (see Example-3);
- for `INTEGER` types with named values, it allows the names of the values to appear in the E-XER encodings as well as the values. In Example-1, “price” is encoded as "twenty" (see Example-3). In the absence of this encoding instruction, the only valid encoding of “price” would be 20 (numeric format);
- for `ENUMERATED` types, it causes the values to be encoded as text strings instead of empty element tags (see Example-3);
- for `BIT STRING` types with named bits, it causes the names of the named bits to be encoded as text strings instead of a bits pattern (see Example-3);
- for `REAL` type values it causes `INF`, `-INF`, and `NaN` to appear in the E-XER encoding instead of the corresponding empty element tags for these special real values (see Example-4).

Example 1:

```
Product ::= [ELEMENT] SEQUENCE {
    size INTEGER,
    number [USE-TYPE] CHOICE {
        shortProductNumber INTEGER (1..100),
        longProductNumber INTEGER (1..10000)
    }
}

value Product ::= { size 10, number longProductNumber:1000}

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product xmlns:asn1="urn:oid:2.1.5.2.0.1">
  <size>10</size>
  <number asn1:type="longProductNumber">1000</number>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <size>10</size>
  <number>
    <longProductNumber>1000</longProductNumber>
  </number>
</Product>
```

Example 2:

```
Product ::= [ELEMENT] SEQUENCE {
    size INTEGER,
    number [USE-TYPE] CHOICE {
        shortProductNumber INTEGER (1..100),
        longProductNumber INTEGER (1..10000)
    }
}

value Product ::= { size 10, number longProductNumber:1000}

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
    GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <size>10</size>
  <number xsi:type="longProductNumber">1000</number>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <size>10</size>
  <number>
    <longProductNumber>1000</longProductNumber>
  </number>
</Product>
```

Example 3:

```
Product ::= SEQUENCE {
    avail   BOOLEAN,
```

```
    color   ENUMERATED { red(0), blue(1), green(2), yellow(3) },
    name    IA5String,
    price   INTEGER { ten(10), twenty(20), thirty(30) },
    atStore BIT STRING {store1(0), store2(1), store3(2)}
}

value Product ::= {
    avail TRUE,
    color red,
    name "shirt",
    price twenty,
    atStore { store1, store3 }
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com">
  <avail>true</avail>
  <color>red</color>
  <name>shirt</name>
  <price>twenty</price>
  <atStore>store1 store3</atStore>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <avail><true/></avail>
  <color><red/></color>
  <name>shirt</name>
  <price>20</price>
  <atStore>101</atStore>
</Product>
```

Example 4:

```
R ::= REAL

value R ::= MINUS-INFINITY

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<R xmlns="http://www.example.com">-INF</R>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<R><MINUS-INFINITY/></R>
```

11 LIST

This encoding instruction can only be applied to “SEQUENCE OF” and “SET OF” types. The component of “SEQUENCE OF” and “SET OF” cannot itself be a “SEQUENCE OF” or “SET OF” type, or contain a nested “SEQUENCE OF” or “SET OF” type with a LIST encoding instruction at any depth.

The component of the “SEQUENCE OF” or “SET OF” type: must be a character-encodable type; and must be such that, for all of its abstract values, there is at least one encoding that is not empty (“”) and that does not contain white-spaces.

Example 1:

```
S ::= [LIST] SEQUENCE OF INTEGER
value S ::= {10, 20, 34, 67, 98}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  10 20 34 67 98
</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <INTEGER>10</INTEGER>
  <INTEGER>20</INTEGER>
  <INTEGER>34</INTEGER>
  <INTEGER>67</INTEGER>
  <INTEGER>98</INTEGER>
</S>
```

Example 2:

```
E ::= ENUMERATED { red(0), green(1), blue(2), yellow(4) }
S ::= [LIST] SEQUENCE OF E
value S ::= {red, green, yellow, green, blue}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  red green yellow green blue
</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <red/><green/><yellow/><green/><blue/>
</S>
```

12 NAME

This encoding instruction can be used for five different purposes:

- to change the element name, the attribute name, or the value of type identification attribute of the target;
- to change the case (or the case of the initial letter) of the element name, the attribute name, or the value of a type identification attribute of the target(s);
- to change the element name used in an empty-element tag normally derived from a specified identifier used in the type definition of the target;
- to change the case (or the case of the initial letter) of the element name used in an empty-element tag normally derived from a specified identifier used in the type definition of the target(s);
- to change the case (or the case of the initial letter) of the element names used in the encoding derived from any identifier used in the type definition of the target(s).

Example 1:

```
S ::= SEQUENCE {
    r [NAME AS "Red"] IA5String,
    blue [NAME AS UPPERCASED] IA5String,
    black [NAME AS CAPITALIZED] IA5String
}

value S ::= { r "shirt", blue "trousers", black "shoes" }

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <Red>shirt</Red>
  <BLUE>trousers</BLUE>
  <Black>shoes</Black>
</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <r>shirt</r>
  <blue>trousers</blue>
```

```
  <black>shoes</black>
</S>
```

Example 2:

```
C ::= [USE-TYPE] CHOICE {
    first [NAME AS "One"] BOOLEAN,
    second [NAME AS "Two"] INTEGER
}

value C ::= second:123

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<C xmlns="http://www.example.com" xmlns:asn1="urn:oid:2.1.5.2.0.1"
asn1:type="Two">
  123
</C>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<C>
  <second>123</second>
</C>
```

13 NAMESPACE

The NAMESPACE encoding instruction enables a namespace name and recommended namespace prefix to be assigned to type reference names and component identifiers.

Example 1:

```
S ::= SEQUENCE {
    firstName IA5String,
    lastName IA5String,
    middleInitial IA5String
}

value S ::= {
    firstName "John",
    lastName "Doe",
    middleInitial "M"
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.org/test" PREFIX "tst"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<tst:S xmlns:tst="http://www.example.org/test">
  <firstName>John</firstName>
  <lastName>Doe</lastName>
  <middleInitial>M</middleInitial>
</tst:S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
  <middleInitial>M</middleInitial>
</S>
```

Example 2:

```
S ::= [NAMESPACE AS "http://www.example.org/B"] IA5String

value S ::= "Robert"

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.org/A" PREFIX "tst"
END
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S xmlns="http://www.example.org/B">Robert</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>Robert</S>
```

Example 3:

```
FirstName ::= IA5String
LastName ::= IA5String
MiddleInitial ::= IA5String

S ::= SEQUENCE {
    first FirstName,
    last LastName,
    middle MiddleInitial
}

value S ::= {
    first "John",
    last "Doe",
    middle "M"
```

```
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL, ALL IN ALL AS "http://www.example.org/data"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S xmlns="http://www.example.org/data">
  <first>John</first>
  <last>Doe</last>
  <middle>M</middle>
</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <first>John</first>
  <last>Doe</last>
  <middle>M</middle>
</S>
```

14 PI-OR-COMMENT

This encoding instruction causes specified XML processing instructions and/or comments to be inserted before or after the content of element, or before or after the associated tags.

Example:

```
S ::= [PI-OR-COMMENT AS {"<!--Personal Data-->", {0, 0, 0, 10}} BEFORE-TAG]
SEQUENCE {
    firstName [PI-OR-COMMENT AS "<?proc-inst?>" AFTER-VALUE] IA5String,
    lastName IA5String,
    middleInitial IA5String
}

value S ::= {
    firstName "John",
    lastName "Doe",
    middleInitial "M"
}
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Personal Data-->
<S>
  <firstName>John<?proc-inst?></firstName>
  <lastName>Doe</lastName>
  <middleInitial>M</middleInitial>
</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
  <middleInitial>M</middleInitial>
</S>
```

15 TEXT

The purpose of this encoding instruction is to enable boolean and enumerated values to be specified as character strings in the content of an XML element. It also enables the character strings that are used for each of those values to be independently modified, or to be the identifiers in an enumerated type definition (or to be "true" and "false" for the boolean type).

This encoding instruction can only be applied to `BOOLEAN` types, `BIT STRING` types with named bits, `ENUMERATED` types, and `INTEGER` types with named numbers.

Example 1:

```
ProductColor ::= ENUMERATED {red(0), light-green(1), blue(2)}

Availability ::= BOOLEAN

S ::= SEQUENCE {
    color ProductColor,
    available Availability
}

value S ::= {
    color light-green,
    available TRUE
}

ENCODING-CONTROL XER
  TEXT ProductColor:red AS UPPERCASED
  TEXT ProductColor:light-green AS "Light Green"
  TEXT ProductColor:blue AS CAPITALIZED
  TEXT Availability:ALL AS UPPERCASED
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <color>Light Green</color>
  <available>TRUE</available>
</S>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S>
  <color><light-green/></color>
```

```
  <available><true/></available>
</S>
```

16 UNTAGGED

This encoding instruction is used to remove the XML start-tag that precedes the encoding of the "Type" to which it is applied and the XML end-tag that follows it.

This encoding instruction can only be applied to the components of SET, SEQUENCE, CHOICE types, or component of "SET OF" and "SEQUENCE OF" ASN.1 constructed types.

Example:

```
ProductColor ::= ENUMERATED {red(0), green(1), blue(2)}

ProductInfo ::= SEQUENCE {
    name IA5String,
    color ProductColor
}

AdditionalInfo ::= SEQUENCE {
    size INTEGER,
    available BOOLEAN
}

Product ::= SEQUENCE {
    color [UNTAGGED] SEQUENCE OF ProductColor,
    info [UNTAGGED] ProductInfo,
    addInfo [UNTAGGED] AdditionalInfo,
    priceInfo [UNTAGGED] CHOICE {
        usd [NAME AS UPPERCASED] [TAG: 0] INTEGER,
        euro [NAME AS UPPERCASED] [TAG: 1] INTEGER
    }
}

value Product ::= {
    color { red, green, blue },
    info {
        name "shirt",
        color red
    },
    addInfo {
        size 44,
        available FALSE
    },
    priceInfo usd:25
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Product>
  <ProductColor>red</ProductColor>
  <ProductColor>green</ProductColor>
  <ProductColor>blue</ProductColor>
  <name>shirt</name>
  <color>red</color>
  <size>44</size>
  <available>false</available>
  <USD>25</USD>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <color>
    <red/><green/><blue/>
  </color>
  <info>
    <name>shirt</name>
    <color><red/></color>
  </info>
  <addInfo>
    <size>44</size>
    <available><false/></available>
  </addInfo>
  <priceInfo>
    <usd>25</usd>
  </priceInfo>
</Product>
```

17 USE-NIL

This encoding instruction is used to optimize the encoding of an OPTIONAL component of a SEQUENCE using a "nil" attribute in the start tag of an element. This encoding instruction can only be applied to a SEQUENCE type with following conditions:

- It can have at most one OPTIONAL component without an ATTRIBUTE encoding instruction.
- All the other components of the SEQUENCE, if any, must have an ATTRIBUTE or ANY-ATTRIBUTE encoding instruction,
- The SEQUENCE type must not have an UNTAGGED encoding instruction.

If the OPTIONAL component is absent in an abstract value then the "nil" attribute is set to "true" or "1", and added to the list of attributes of the enclosing element.

Example 1:

```
ProductColor ::= ENUMERATED {red(0), green(1), blue(2)}

Size ::= [USE-NIL] SEQUENCE {
    size INTEGER OPTIONAL
}
```

```
Product ::= SEQUENCE {
    name IA5String,
    color ProductColor,
    size Size
}


value Product ::= {
    name "shirt",
    color red,
    size { }
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product xmlns:asn1="urn:oid:2.1.5.2.0.1">
  <name>shirt</name>
  <color>red</color>
  <size asn1:nil="true"/>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>shirt</name>
  <color><red/></color>
  <size/>
</Product>
```

If the `OPTIONAL` component is present, the `"nil"` attribute can either be removed, or it can be added to the list of attributes with a value of either "`false`" or "`0`".

Example 2:

```
ProductColor ::= ENUMERATED {red(0), green(1), blue(2)}

Size ::= [USE-NIL] SEQUENCE {
    size INTEGER OPTIONAL
}

Product ::= SEQUENCE {
    name IA5String,
    color ProductColor,
    size Size
}

value Product ::= {
    name "shirt",
    color red,
    size { size 10 }
}
```

```
ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product xmlns:asn1="urn:oid:2.1.5.2.0.1">
  <name>shirt</name>
  <color>red</color>
  <size asn1:nil=”false”>10</size>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>shirt</name>
  <color><red/></color>
  <size>
    <size>10</size>
  </size>
</Product>
```

18 USE-NUMBER

The purpose of this encoding instruction is to modify the encoding of an `ENUMERATED` type so that the numbers in the enumeration are used instead of the identifiers. This encoding instruction cannot be assigned unless `GLOBAL-DEFAULTS MODIFIED-ENCODINGS` encoding instruction is present in the encoding control section.

Example:

```
ProductColor ::= [USE-NUMBER] ENUMERATED {red(0), green(1), blue(2)}

Size ::= [USE-NIL] SEQUENCE {
    size INTEGER OPTIONAL
}

Product ::= SEQUENCE {
    name IA5String,
    color ProductColor,
    size Size
}


value Product ::= {
    name "shirt",
    color red,
    size { size 10 }
}

ENCODING-CONTROL XER
```

```
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>shirt</name>
  <color>0</color>
  <size>10</size>
</Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>shirt</name>
  <color><red/></color>
  <size>
    <size>10</size>
  </size>
</Product>
```

19 USE-ORDER

The purpose of this encoding instruction is to provide a special E-XER encoding of a SEQUENCE type in which there is a SEQUENCE OF component that determines the semantic order of the values of the following components of the sequence type that are encoded as elements.

The component determining the semantic order is required to be a SEQUENCE OF type with a component that is an ENUMERATED type, and it must be the first component of sequence unless there is another sequence-of component supporting a final EMBED-VALUES encoding instruction on the sequence type. In this case, the SEQUENCE OF component supporting the EMBED-VALUES encoding instruction must precede the SEQUENCE OF component supporting the USE-ORDER encoding instruction.

Example 1:

```
Product ::= [USE-ORDER] SEQUENCE {
    order SEQUENCE OF ENUMERATED { id, name, price, color },
    id INTEGER,
    name IA5String,
    price REAL,
    color IA5String
} (CONSTRAINED BY {})

value Product ::= {
    order { id, color, price, name },
    id 100,
    name "shirt",
    price 12.23,
    color "red"
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
```

```
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com">
  <id>100</id>
  <color>red</color>
  <price>1.223E1</price>
  <name>shirt</name>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <order>
    <id/><color/><price/><name/>
  </order>
  <id>100</id>
  <name>shirt</name>
  <price>1.223E1</price>
  <color>red</color>
</Product>
```

Example 2:

```
Product ::= [USE-ORDER] [EMBED-VALUES] SEQUENCE {
    embed-value SEQUENCE OF UTF8String,
    order SEQUENCE OF ENUMERATED { id, name, price, color },
    id INTEGER,
    name IA5String,
    price   [DECIMAL] REAL (WITH COMPONENTS {..., base(10)})
            (ALL EXCEPT (-0 | MINUS-INFINITY | PLUS-INFINITY | NOT-A-NUMBER)),
    color IA5String
} (CONSTRAINED BY {})

value Product ::= {
    embed-value {"Order Id ", "", "", "US Dollars", "" },
    order { id, color, price, name },
    id 100,
    name "shirt",
    price 12.23,
    color "red"
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com">Order Id
<id>100</id><color>red</color><price>12.23</price>US
Dollars<name>shirt</name></exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <embed-value>
    <UTF8String>Order Id </UTF8String>
    <UTF8String/>
    <UTF8String/>
    <UTF8String>US Dollars</UTF8String>
    <UTF8String/>
  </embed-value>
  <order>
    <id/><color/><price/><name/>
  </order>
  <id>100</id>
  <name>shirt</name>
  <price>1.223E1</price>
  <color>red</color>
</Product>
```

It is possible to apply both USE-ORDER and USE-NIL to a type. In this case, the OPTIONAL component supporting the USE-NIL encoding instruction must be a SEQUENCE, and the SEQUENCE OF component supporting the USE-ORDER encoding instruction will determine the semantic order of the components of that OPTIONAL SEQUENCE.

Example 3:

```
Product ::= [USE-ORDER] [USE-NIL] SEQUENCE {
    order SEQUENCE OF ENUMERATED { id, name, price, color },
    info ProductInfo OPTIONAL
} (CONSTRAINED BY {})

ProductInfo ::= SEQUENCE {
    id INTEGER,
    name IA5String,
    price REAL,
    color IA5String
}

value Product ::= {
    order { id, color, price, name },
    info {
    id 100,
    name "shirt",
    price 12.23,
    color "red"
    }
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com">
  <id>100</id>
  <color>red</color>
  <price>1.223E1</price>
  <name>shirt</name>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <order>
    <id/><color/><price/><name/>
  </order>
  <info>
    <id>100</id>
    <name>shirt</name>
    <price>1.223E1</price>
    <color>red</color>
  </info>
</Product>
```

Example 4:

```
Product ::= [USE-ORDER] [USE-NIL] SEQUENCE {
    order SEQUENCE OF ENUMERATED { id, name, price, color },
    info ProductInfo OPTIONAL
} (CONSTRAINED BY {})

ProductInfo ::= SEQUENCE {
    id INTEGER,
    name IA5String,
    price REAL,
    color IA5String
}

value Product ::= {
    order {}
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Product>
  <order/>
</Product>
```

20 USE-QNAME

The USE-QNAME encoding instruction can be applied to a SEQUENCE type representing a name (either a namespace-qualified name or an unqualified name). This encoding instruction can only be applied to a SEQUENCE with exactly two components, both of the type UTF8String, with the first component marked OPTIONAL. The first component must be restricted to represent a URI and second component must be restricted to contain an "NCName" as specified in W3C XML Namespaces.

Example 1:

This example shows the E-XER encoding when the URI of the name (i.e., the optional component of the SEQUENCE) is present and is different from the target namespace.

```
ProductName ::= [USE-QNAME] SEQUENCE {
  uri   UTF8String (CONSTRAINED BY {}) OPTIONAL,
  name  UTF8String (CONSTRAINED BY {})
}

Product ::= SEQUENCE {
  name ProductName,
  cost REAL
}

value Product ::= {
  name {
    uri "http://www.furniture.com",
    name "table"
  },
  cost 199.95
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
END
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com">
  <name xmlns:b0="http://www.furniture.com">b0:table</name>
  <cost>1.9995E2</cost>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>
    <uri>http://www.furniture.com</uri>
```

```
    <name>table</name>
  </name>
  <cost>1.9995E2</cost>
</Product>
```

Example 2:

This example shows the E-XER encoding when the URI of the name (i.e., the optional component of the SEQUENCE) is missing.

```
ProductName ::= [USE-QNAME] SEQUENCE {
  uri   UTF8String (CONSTRAINED BY {}) OPTIONAL,
  name  UTF8String (CONSTRAINED BY {})
}

Product ::= SEQUENCE {
  name ProductName,
  cost REAL
}

value Product ::= {
  name {
    name "table"
  },
  cost 199.95
}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com">
  <name>Table</name>
  <cost>1.9995E2</cost>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <name>
    <name>table</name>
  </name>
  <cost>1.9995E2</cost>
</Product>
```

21 USE-TYPE

This encoding instruction optimizes the E-XER encoding of a `CHOICE` type. It requires a type identification attribute to be encoded in the enclosing element to identify the alternative that has been chosen (unless this is the first alternative) and the removal of the start-tag and end-tag around the encoding of the alternatives.

There are certain restrictions in applying this encoding instruction:

- It can only be applied to a `CHOICE` type without an `UNTAGGED` encoding instruction.
- None of the alternatives of the `CHOICE` type can have an `UNTAGGED` encoding instruction.
- None of the alternatives of the `CHOICE` type can itself be a `CHOICE` type with a `USE-TYPE` encoding instruction.

Example 1:

This example shows if the selected alternative is not the first alternative of `CHOICE`, type identification attribute is added in the attribute list.

```
Shirt ::= SEQUENCE {
   color IA5String,
   make  IA5String,
   size  INTEGER
}

Trousers ::= SEQUENCE {
   available [ATTRIBUTE] BOOLEAN,
   color IA5String,
   make  IA5String
}

Shoes ::= SEQUENCE {
   available [ATTRIBUTE] BOOLEAN,
   color IA5String,
   size  INTEGER
}

Product ::= [USE-TYPE] CHOICE {
   shirt Shirt,
   trousers Trousers,
   shoes Shoes
}

value Product ::= shoes:{available FALSE, color "red", size 9}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" available="false"
xsi:type="shoes">
  <color>red</color>
  <size>9</size>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <shoes>
    <available><false/></available>
    <color>red</color>
    <size>9</size>
  </shoes>
</Product>
```

Example 2:

This example shows that if the selected alternative is the first alternative of the CHOICE, the type identification attribute may be omitted by the encoder.

```
Shirt ::= SEQUENCE {
   color IA5String,
   make  IA5String,
   size  INTEGER
}

Trousers ::= SEQUENCE {
   available [ATTRIBUTE] BOOLEAN,
   color IA5String,
   make  IA5String
}

Shoes ::= SEQUENCE {
   available [ATTRIBUTE] BOOLEAN,
   color IA5String,
   size  INTEGER
}

Product ::= [USE-TYPE] CHOICE {
   shirt Shirt,
   trousers Trousers,
   shoes Shoes
}

value Product ::= shirt:{color "red", make "ABC Company", size 9}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <color>red</color>
  <make>ABC Company</make>
  <size>9</size>
```

```
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <shirt>
    <color>red</color>
    <make>ABC Company</make>
    <size>9</size>
  </shirt>
</Product>
```

22 USE-UNION

This encoding instruction is used to produce a special encoding of a `CHOICE` type, where no tags are generated around the chosen alternative. This encoding instruction can only be applied if the encoding of the abstract values of each alternative is sufficiently distinct from the encoding of abstract values of the textually-preceding alternatives, so that a decoder can determine the abstract value represented by the analysis of the encoding. If this condition is not satisfied, this encoding instruction can still be applied if the CHOICE type is encoded as the content of an XML element. In this case, a type identification attribute can be used to determine the alternative being encoded.

If an `ATTRIBUTE` or `UNTAGGED` encoding instruction is not applied to the `CHOICE` type with a `USE-UNION` encoding instruction, then this encoding instruction can result in the insertion of a type identification attribute in the enclosing element to identify the alternative that has been encoded. If the `CHOICE` type has a final `ATTRIBUTE` or `UNTAGGED` encoding instruction, or is the component of a `SEQUENCE OF` or `SET OF` type with a `LIST` encoding instruction, the insertion of the type identification attribute is not possible. This encoding instruction causes the removal of the start-tag and end-tag around the encoding of the alternative.

This encoding instruction can only be assigned to `CHOICE` type. All the alternatives of the `CHOICE` type must be character-encodable types, but cannot be another `CHOICE` with a `USE-UNION` encoding instruction

Example 1:

```
ProductId ::= [USE-UNION] CHOICE {
    c1 INTEGER (1..100),
    c2 INTEGER (1..1000),
    c3 INTEGER (1001..2000)
}

Product ::= SEQUENCE {
    id ProductId,
    price REAL,
    color IA5String
}

value Product ::= { id c2:100, price 25.34, color "green"}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
```

```
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com" xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance">
  <id xsi:type="c2">100</id>
  <price>2.534E1</price>
  <color>green</color>
</exm:Product>
```

E-XER encoding:

```
<Product>
  <id>
    <c2>100</c2>
  </id>
  <price>2.534E1</price>
  <color>green</color>
</Product>
```

Example 2:

```
ProductId ::= [USE-UNION] CHOICE {
    c1 INTEGER (1..100),
    c2 INTEGER (201..1000),
    c3 INTEGER (1001..2000)
}

ProductList ::= [LIST] SEQUENCE OF ProductId

value ProductList ::= { c1:20, c2:234, c3:1004 }

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProductList xmlns="http://www.example.com"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 20 234 1004
</ProductList>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProductList>
  <c1>20</c1>
  <c2>234</c2>
  <c3>1004</c3>
</ProductList>
```

Example 3:

```
ProductId ::= [USE-UNION] CHOICE {
    c1 INTEGER (1..100),
    c2 INTEGER (101..1000),
    c3 INTEGER (1001..2000)
}

Product ::= SEQUENCE {
    id [ATTRIBUTE] ProductId,
    price REAL,
    color IA5String
}

value Product ::= { id c2:200, price 25.34, color "green"}

ENCODING-CONTROL XER
  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
  NAMESPACE ALL AS "http://www.example.com" PREFIX "exm"
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<exm:Product xmlns:exm="http://www.example.com" xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance" id="200">
  <price>2.534E1</price>
  <color>green</color>
</exm:Product>
```

BASIC-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
  <id>
    <c2>200</c2>
  </id>
  <price>2.534E1</price>
  <color>green</color>
</Product>
```

23 WHITESPACE

This encoding instruction specifies the whitespace normalization rules, and allows decoders to accept additional options in the encoding of the space (#x20) character and in the use of leading and trailing whitespaces for character string encodings. This encoding instruction can only be applied to string types restricted not to contain tab (#x9), line feed (#xA), and carriage return (#xD) characters.

Example 1:

If this encoding instruction is used with the REPLACE keyword, the encoder can replace each space in the abstract value with a tab, line feed or carriage return character. The decoder will replace these characters with spaces before doing validation.

```
C ::= [WHITESPACE REPLACE] IA5String (FROM("A.."Z" | "a.."z" | " "))

value C ::= "First Line Second Line"

ENCODING-CONTROL XER

  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
```

The following is a legal E-XER encoding of "value". At the time of decoding, the content of C will be normalized to "First Line Second Line" which is a legal value of type C.

```
<?xml version="1.0" encoding="UTF-8"?>
<C xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">First
Line
Second
Line</C>
```

Example 2:

If the WHITESPACE COLLAPSE encoding instruction is used, the encoder can replace each space (#x20) character by any number of space, tab, line feed, and carriage return characters. In addition, one or more such characters can be added to the beginning and/or to the end of the encoded value.

The decoder first replaces each occurrence of tab (#x9), line feed (#xA), and carriage return (#xD) with a space (#x20). After this replacement, all groups of consecutive spaces are collapsed into a single space. In addition, leading and trailing spaces are deleted.

```
C ::= [WHITESPACE COLLAPSE] IA5String (FROM("A.."Z" | "a.."z" | " "))

value C ::= {"First Line", " Second Line"}

ENCODING-CONTROL XER

  GLOBAL-DEFAULTS MODIFIED-ENCODINGS
  GLOBAL-DEFAULTS CONTROL-NAMESPACE
        "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
```

The following is a valid encoding of "value". At the time of decoding, the content of C will be normalized to "First Line Second Line", which is a legal value of type C.

```
<?xml version="1.0" encoding="UTF-8"?>
<C xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  First Line
  Second Line
</C>
```

Appendix A

A.1: The following table shows the allowed combinations of encoding instructions in the presence of `GLOBAL-DEFAULTS MODIFIED-ENCODINGS`.

Encoding instruction	Permitted other encoding instructions
ANY-ATTRIBUTES	ELEMENT, NAME, NAMESPACE
ANY-ELEMENT	ELEMENT, NAME, NAMESPACE
ATTRIBUTE	BASE64, DECIMAL, ELEMENT, LIST, NAME, NAMESPACE, TEXT, USE-NUMBER, USE-QNAME, USE-UNION, WHITESPACE
BASE64	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
DECIMAL	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
DEFAULT-FOR-EMPTY	BASE64, DECIMAL, ELEMENT, EMBED-VALUES, LIST, NAME, NAMESPACE, PI-OR-COMMENT, TEXT, USE-NIL, USE-NUMBER, USE-ORDER, USE-QNAME, USE-UNION, WHITESPACE
ELEMEN	Equivalent to NOT UNTAGGED
EMBED-VALUES	DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, USE-NIL, USE-ORDER
LIST	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
NAME	No restrictions
NAMESPACE	No restrictions
PI-OR-COMMENT	BASE64, DECIMAL, DEFAULT-FOR-EMPTY, ELEMENT, EMBED-VALUES, LIST, NAME, NAMESPACE, TEXT, USE-NIL, USE-NUMBER, USE-ORDER, USE-QNAME, USE-TYPE, USE-UNION, WHITESPACE
TEXT	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
UNTAGGED	BASE64, DECIMAL, LIST, NAME, NAMESPACE, TEXT, USE-NUMBER, USE-QNAME, USE-UNION, WHITESPACE
USE-NIL	DEFAULT-FOR-EMPTY, ELEMENT, EMBED-VALUES, NAME, NAMESPACE, PI-OR-COMMENT, USE-ORDER
USE-NUMBER	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
USE-ORDER	DEFAULT-FOR-EMPTY, ELEMENT, EMBED-VALUES, NAME, NAMESPACE, PI-OR-COMMENT, USE-NIL.
USE-QNAME	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
USE-TYPE	ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT
USE-UNION	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED
WHITESPACE	ATTRIBUTE, DEFAULT-FOR-EMPTY, ELEMENT, NAME, NAMESPACE, PI-OR-COMMENT, UNTAGGED

A.2: The following table shows the allowed combinations of encoding instructions in the absence of `GLOBAL-DEFAULTS MODIFIED-ENCODINGS`.

Encoding instruction	Permitted other encoding instructions
ANY-ATTRIBUTES	Not permitted
ANY-ELEMENT	Not permitted
ATTRIBUTE	BASE64, LIST, NAME, TEXT, USE-NUMBER, WHITESPACE
BASE64	ATTRIBUTE, NAME, PI-OR-COMMENT
DECIMAL	Not permitted
DEFAULT-FOR-EMPTY	Not permitted
ELEMENT	Not permitted
EMBED-VALUES	Not permitted
LIST	ATTRIBUTE, NAME, PI-OR-COMMENT
NAME	ATTRIBUTE, BASE64, LIST, PI-OR-COMMENT, TEXT, USE-NUMBER, WHITESPACE
NAMESPACE	Not permitted
PI-OR-COMMENT	BASE64, LIST, NAME, TEXT, USE-NUMBER, WHITESPACE
TEXT	ATTRIBUTE, NAME, PI-OR-COMMENT
UNTAGGED	Not permitted
USE-NIL	Not permitted
USE-NUMBER	ATTRIBUTE, NAME, PI-OR-COMMENT
USE-ORDER	Not permitted
USE-QNAME	Not permitted
USE-TYPE	Not permitted
USE-UNION	Not permitted
WHITESPACE	ATTRIBUTE, NAME, PI-OR-COMMENT