Network Working Group
Request for Comments: 4910
Category: Experimental

S. Legg
eB2Bcom
D. Prager

July 2007

Robust XML Encoding Rules (RXER) for Abstract Syntax Notation One (ASN.1)

Status of This Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document defines a set of Abstract Syntax Notation One (ASN.1) encoding rules, called the Robust XML Encoding Rules or RXER, that produce an Extensible Markup Language (XML) representation for values of any given ASN.1 data type. Rules for producing a canonical RXER encoding are also defined.

Table of Contents

1. Introduction

This document defines a set of Abstract Syntax Notation One (ASN.1) [X.680] encoding rules, called the Robust XML Encoding Rules or RXER, that produce an Extensible Markup Language (XML) [XML10][XML11] representation of ASN.1 values of any given ASN.1 type.

An ASN.1 value is regarded as analogous to the content and attributes of an XML element, or in some cases, just an XML attribute value. The RXER encoding of an ASN.1 value is the well-formed and valid content and attributes of an element, or an attribute value, in an XML document [XML10][XML11] conforming to XML namespaces [XMLNS10][XMLNS11]. Simple ASN.1 data types such as PrintableString, INTEGER, and BOOLEAN define character data content or attribute values, while the ASN.1 combining types (i.e., SET, SEQUENCE, SET OF, SEQUENCE OF, and CHOICE) define element content and attributes. The attribute and child element names are generally provided by the identifiers of the components in combining type definitions, i.e., elements and attributes correspond to the NamedType notation.

RXER leaves some formatting details to the discretion of the encoder, so there is not a single unique RXER encoding for an ASN.1 value. However, this document also defines a restriction of RXER, called the Canonical Robust XML Encoding Rules (CRXER), which does produce a single unique encoding for an ASN.1 value. Obviously, the CRXER encoding of a value is also a valid RXER encoding of that value. The restrictions on RXER to produce the CRXER encoding are interspersed with the description of the rules for RXER.

Note that "ASN.1 value" does not mean a Basic Encoding Rules (BER) [X.690] encoding. The ASN.1 value is an abstract concept that is independent of any particular encoding. BER is just one possible way to encode an ASN.1 value. This document defines an alternative way to encode an ASN.1 value.

A separate document [RXEREI] defines encoding instructions [X.680-1] that may be used in an ASN.1 specification to modify how values are encoded in RXER, for example, to encode a component of a combining ASN.1 type as an attribute rather than as a child element. A pre-existing ASN.1 specification will not have RXER encoding instructions, so any mention of encoding instructions in this document can be ignored when dealing with such specifications. Encoding instructions for other encoding rules have no effect on RXER encodings.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in BCP 14, RFC 2119 [BCP14]. The key word "OPTIONAL" is exclusively used with its ASN.1 meaning.

A reference to an ASN.1 production [X.680] (e.g., Type, NamedType) is a reference to the text in an ASN.1 specification corresponding to that production.

The specification of RXER makes use of definitions from the XML Information Set (Infoset) [INFOSET]. In particular, information item property names follow the Infoset convention of being shown in square brackets, e.g., [local name]. Literal values of Infoset properties are enclosed in double quotes; however, the double quotes are not part of the property values. In the sections that follow, "information item" will be abbreviated to "item", e.g., "element information item" is abbreviated to "element item". The term "element" or "attribute" (without the "item") is referring to an element or attribute in an XML document, rather than an information item.

Literal character strings to be used in an RXER encoding appear within double quotes; however, the double quotes are not part of the literal value and do not appear in the encoding.

This document uses the namespace prefix [XMLNS10][XMLNS11] "asnx:" to stand for the namespace name "urn:ietf:params:xml:ns:asnx", uses the namespace prefix "xs:" to stand for the namespace name "http://www.w3.org/2001/XMLSchema", and uses the namespace prefix "xsi:" to stand for the namespace name "http://www.w3.org/2001/XMLSchema-instance". However, in practice, any valid namespace prefixes are permitted in non-canonical RXER encodings (namespace prefixes are deterministically generated for CRXER).

The encoding instructions [X.680-1] referenced by name in this specification are encoding instructions for RXER [RXEREI].

Throughout this document, references to the Markup, AnyURI, NCName, Name, and QName ASN.1 types are references to the types described in Section 4 and consolidated in the AdditionalBasicDefinitions module in Appendix A. Any provisions associated with the reference do not apply to types defined in other ASN.1 modules that happen to have these same names.

Code points for characters [UCS][UNICODE] are expressed using the Unicode convention U+n, where n is four to six hexadecimal digits, e.g., the space character is U+0020.

3. Definitions

Definition (white space character): A white space character is a space (U+0020), tab (U+0009), carriage return (U+000D), or line feed (U+000A) character.

Definition (white space): White space is a sequence of one or more white space characters.

Definition (line break): A line break is any sequence of characters that is normalized to a line feed by XML End-of-Line Handling [XML10][XML11].

Definition (serialized white space): Serialized white space is a sequence of one or more white space characters and/or line breaks.

Definition (declaring the default namespace): A namespace declaration attribute item is declaring the default namespace if the [prefix] of the attribute item has no value, the [local name] of the attribute item is "xmlns" and the [normalized value] is not empty.

Definition (undeclaring the default namespace): A namespace declaration attribute item is undeclaring the default namespace if the [prefix] of the attribute item has no value, the [local name] of the attribute item is "xmlns" and the [normalized value] is empty (i.e., xmlns="").

Definition (canonical namespace prefix): A canonical namespace prefix is an NCName [XMLNS10] beginning with the letter 'n' (U+006E) followed by a non-negative number string. A non-negative number string is either the digit character '0' (U+0030), or a non-zero decimal digit character (U+0031-U+0039) followed by zero, one, or more of the decimal digit characters '0' to '9' (U+0030-U+0039).

For convenience, a CHOICE type where the ChoiceType is subject to a UNION encoding instruction will be referred to as a UNION type, and a SEQUENCE OF type where the SequenceOfType is subject to a LIST encoding instruction will be referred to as a LIST type.

4. Additional Basic Types

This section defines an ASN.1 type for representing markup in abstract values, as well as basic types that are useful in encoding instructions [RXEREI] and other related specifications [ASN.X].

The ASN.1 definitions in this section are consolidated in the AdditionalBasicDefinitions ASN.1 module in Appendix A.

4.1. The Markup Type

A value of the Markup ASN.1 type holds the [prefix], [attributes], [namespace attributes], and [children] of an element item, i.e., the content and attributes of an element.

RXER has special provisions for encoding values of the Markup type (see Section 6.10). For other encoding rules, a value of the Markup type is encoded according to the following ASN.1 type definition (with AUTOMATIC TAGS):

```
Markup ::= CHOICE {
    text    SEQUENCE {
        prolog      UTF8String (SIZE(1..MAX)) OPTIONAL,
        prefix      NCName OPTIONAL,
        attributes  UTF8String (SIZE(1..MAX)) OPTIONAL,
        content     UTF8String (SIZE(1..MAX)) OPTIONAL
    }
}
```

The text alternative of the Markup CHOICE type provides for the [prefix], [attributes], [namespace attributes], and [children] of an element item to be represented as serialized XML using the UTF-8 character encoding [UTF-8].

Aside: The CHOICE allows for one or more alternative compact representations of the content and attributes of an element to be supported in a future specification.

With respect to some element item whose content and attributes are represented by a value of the text alternative of the Markup type:

(1) the prolog component of the value contains text that, after line break normalization, conforms to the XML prolog production [XML10][XML11],

(2) the prefix component is absent if the [prefix] of the element item has no value; otherwise, the prefix component contains the [prefix] of the element item,

(3) the attributes component of the value contains an XML serialization of the [attributes] and [namespace attributes] of the element item, if any, with each attribute separated from the next by serialized white space, and

(4) the content component is absent if the [children] property of the element item is empty; otherwise, the content component of the value contains an XML serialization of the [children] of the element item.

All the components of a value of the Markup type MUST use the same version of XML, either version 1.0 [XML10] or version 1.1 [XML11]. If XML version 1.1 is used, then the prolog component MUST be present and MUST have an XMLDecl for version 1.1. If the prolog component is absent, then XML version 1.0 is assumed.

If the prefix component is present, then there MUST be a namespace declaration attribute in the attributes component that defines that namespace prefix (since an element whose content and attributes are described by a value of Markup is required to be self-contained; see Section 4.1.1).

Note that the prefix component is critically related to the NamedType that has Markup as its type. If a Markup value is extracted from one enclosing abstract value and embedded in another enclosing abstract value (i.e., becomes associated with a different NamedType), then the prefix may no longer be appropriate, in which case it will need to be revised. It may also be necessary to add another namespace

declaration attribute to the attributes component so as to declare a new namespace prefix.

Leading and/or trailing serialized white space is permitted in the attributes component. A value of the attributes component consisting only of serialized white space (i.e., no actual attributes) is permitted.

The attributes and content components MAY contain entity references [XML10][XML11]. If any entity references are used (other than references to the predefined entities), then the prolog component MUST be present and MUST contain entity declarations for those entities in the internal or external subset of the document type definition.

Example

Given the following ASN.1 module:

```
MyModule DEFINITIONS
AUTOMATIC TAGS ::= BEGIN

Message ::= SEQUENCE {
    messageType   INTEGER,
    messageValue  Markup
}

ENCODING-CONTROL RXER

    TARGET-NAMESPACE "http://example.com/ns/MyModule"

    COMPONENT message Message
        -- a top-level NamedType

END
```

consider the following XML document:

```
<?xml version='1.0'?>
<!DOCTYPE message [
    <!ENTITY TRUE 'true'>
]>
<message>
 <messageType>1</messageType>
 <messageValue xmlns:ns="http://www.example.com/ABD"
               ns:foo="1" bar="0">
  <this>&TRUE;</this>
  <that/>
```

```
 </messageValue>
</message>
```

A Markup value corresponding to the content and attributes of the <messageValue> element is, in ASN.1 value notation [X.680] (where "lf" represents the line feed character):

```
text:{
    prolog     { "<?xml version='1.0'?>", lf,
                 "<!DOCTYPE message [", lf,
                 "    <!ENTITY TRUE 'true'>", lf,
                 "]>", lf },
    attributes { " xmlns:ns=""http://www.example.com/ABD""",
                 lf,
                 "               ns:foo=""1"" bar=""0""" },
    content    { lf,
                 "  <this>&TRUE;</this>", lf,
                 "  <that/>", lf, " " }
}
```

The following Markup value is an equivalent representation of the content and attributes of the <messageValue> element:

```
text:{
    attributes {
                 "bar=""0"" ns:foo=""1"" ",
                 "xmlns:ns=""http://www.example.com/ABD""" },
    content    { lf,
                 "  <this>true</this>", lf,
                 "  <that/>", lf, " " }
}
```

By itself, the Markup ASN.1 type imposes no data type restriction on the markup contained by its values and is therefore analogous to the XML Schema anyType [XSD1].

There is no ASN.1 basic notation that can directly impose the constraint that the markup represented by a value of the Markup type must conform to the markup allowed by a specific type definition. However, certain encoding instructions (i.e., the reference encoding instructions [RXEREI]) have been defined to have this effect.

4.1.1. Self-Containment

An element, its attributes and its content, including descendent elements, may contain qualified names [XMLNS10][XMLNS11] as the names of elements and attributes, in the values of attributes, and as character data content of elements. The binding between namespace

prefix and namespace name for these qualified names is potentially determined by the namespace declaration attributes of ancestor elements (which in the Infoset representation are inherited as namespace items in the [in-scope namespaces]).

In the absence of complete knowledge of the data type of an element item whose content and attributes are described by a value of the Markup type, it is not possible to determine with absolute certainty which of the namespace items inherited from the [in-scope namespaces] of the [parent] element item are significant in interpreting the Markup value. The safe and easy option would be to assume that all the namespace items from the [in-scope namespaces] of the [parent] element item are significant and need to be retained within the Markup value. When the Markup value is re-encoded, any of the retained namespace items that do not appear in the [in-scope namespaces] of the enclosing element item in the new encoding could be made to appear by outputting corresponding namespace declaration attribute items in the [namespace attributes] of the enclosing element item.

From the perspective of the receiver of the new encoding, this enlarges the set of attribute items in the [namespace attributes] represented by the Markup value.

In addition, there is no guarantee that the sender of the new encoding has recreated the original namespace declaration attributes on the ancestor elements, so the [in-scope namespaces] of the enclosing element item is likely to have new namespace declarations that the receiver will retain and pass on in the [namespace attributes] when it in turn re-encodes the Markup value.

This unbounded growth in the set of attribute items in the [namespace attributes] defeats any attempt to produce a canonical encoding.

The principle of self-containment is introduced to avoid this problem. An element item (the subject element item) is self-contained if the constraints of Namespaces in XML 1.0 [XMLNS10] are satisfied (i.e., that prefixes are properly declared) and none of the following bindings are determined by a namespace declaration attribute item in the [namespace attributes] of an ancestor element item of the subject element item:

(1) the binding between the [prefix] and [namespace name] of the subject element item,

(2) the binding between the [prefix] and [namespace name] of any descendant element item of the subject element item,

(3) the binding between the [prefix] and [namespace name] of any attribute item in the [attributes] of the subject element item or the [attributes] of any descendant element item of the subject element item,

(4) the binding between the namespace prefix and namespace name of any qualified name in the [normalized value] of any attribute item in the [attributes] of the subject element item or the [attributes] of any descendant element item of the subject element item, or

(5) the binding between the namespace prefix and namespace name of any qualified name represented by a series of character items (ignoring processing instruction and comment items) in the [children] of the subject element item or the [children] of any descendant element item of the subject element item.

Aside: If an element is self-contained, then separating the element from its parent does not change the semantic interpretation of its name and any names in its content and attributes.

A supposedly self-contained element in a received RXER encoding that is in fact not self-contained SHALL be treated as an ASN.1 constraint violation.

Aside: ASN.1 does not require an encoding with a constraint violation to be immediately rejected; however, the constraint violation must be reported at some point, possibly in a separate validation step.

Implementors should note that an RXER decoder will be able to detect some, but not all, violations of self-containment. For example, it can detect element and attribute names that depend on namespace declarations appearing in the ancestors of a supposedly self-contained element. Similarly, where type information is available, it can detect qualified names in character data that depend on the namespace declarations of ancestor elements. However, type information is not always available, so some qualified names will escape constraint checking. Thus, the onus is on the creator of the original encoding to ensure that element items required to be self-contained really are completely self-contained.

An element item whose content and attributes are described by a value of the Markup type MUST be self-contained.

Aside: The procedures in Section 6 take account of the requirements for self-containment so that an RXER encoder following these procedures will not create violations of self-containment.

4.1.2. Normalization for Canonical Encoding Rules

Implementations are given some latitude in how the content and attributes of an element are represented as an abstract value of the Markup type, in part because an Infoset can have different equivalent serializations. For example, the order of attributes and the amount and kind of white space characters between attributes are irrelevant to the Infoset representation. The content can also include one or more elements corresponding to an ASN.1 top-level NamedType or having a data type that is an ASN.1 type. It is only necessary to preserve the abstract value for such elements, and a particular abstract value can have different Infoset representations.

These two characteristics mean that when an RXER encoded value of the Markup type is decoded, the components of the recovered Markup value may not be exactly the same, character for character, as the original value that was encoded, though the recovered value will be semantically equivalent.

However, canonical ASN.1 encoding rules such as the Distinguished Encoding Rules (DER) and the Canonical Encoding Rules (CER) [X.690], which encode Markup values according to the ASN.1 definition of the Markup type, depend on character-for-character preservation of string values. This requirement can be accommodated if values of the Markup type are normalized when they are encoded according to a set of canonical encoding rules.

Aside: The RXER encoding and decoding of a Markup value might change the character string components of the value from the perspective of BER, but there will be a single, repeatable encoding for DER.

A value of the Markup type will appear as the content and attributes of an element in an RXER encoding. When the value is encoded using a set of ASN.1 canonical encoding rules other than CRXER, the components of the text alternative of the value MUST be normalized as follows, by reference to the element as it would appear in a CRXER encoding:

(1) The value of the prolog component SHALL be the XMLDecl <?xml version="1.1"?> with no other leading or trailing characters.

(2) If the element's name is unprefixed in the CRXER encoding, then the prefix component SHALL be absent; otherwise, the value of the prefix component SHALL be the prefix of the element's name in the CRXER encoding.

(3) Take the character string representing the element's attributes, including namespace declarations, in the CRXER encoding. If the first attribute is a namespace declaration that undeclares the default namespace (i.e., xmlns=""), then remove it. Remove any leading space characters. If the resulting character string is empty, then the attributes component SHALL be absent; otherwise, the value of the attributes component SHALL be the resulting character string.

Aside: Note that the attributes of an element can change if an RXER encoding is re-encoded in CRXER.

(4) If the element has no characters between the start-tag and end-tag [XML11] in the CRXER encoding, then the content component SHALL be absent; otherwise, the value of the content component SHALL be identical to the character string in the CRXER encoding bounded by the element's start-tag and end-tag.

Aside: A consequence of invoking the CRXER encoding is that any nested element corresponding to an ASN.1 top-level NamedType, or indeed the element itself, will be normalized according to its ASN.1 value rather than its Infoset representation. Likewise for an element whose data type is an ASN.1 type. Section 6.4 describes how these situations can arise.

Aside: It is only through values of the Markup type that processing instructions and comments can appear in CRXER encodings.

If an application uses DER, but has no knowledge of RXER, then it will not know to normalize values of the Markup type. If RXER is deployed into an environment containing such applications, then Markup values SHOULD be normalized, even when encoding using non-canonical encoding rules.

4.2. The AnyURI Type

A value of the AnyURI ASN.1 type is a character string conforming to the format of a Uniform Resource Identifier (URI) [URI].

```
AnyURI ::= UTF8String (CONSTRAINED BY
            { -- conforms to the format of a URI -- })
```

4.3. The NCName Type

A value of the NCName ASN.1 type is a character string conforming to the NCName production of Namespaces in XML 1.0 [XMLNS10].

```
NCName ::= UTF8String (CONSTRAINED BY
                { -- conforms to the NCName production of
                  -- Namespaces in XML 1.0 -- })
```

Aside: The NCName production for Namespaces in XML 1.1 [XMLNS11] allows a wider range of characters than the NCName production for Namespaces in XML 1.0. The NCName type for ASN.1 is currently restricted to the characters allowed by Namespaces in XML 1.0, though this may change in a future specification of RXER.

4.4. The Name Type

A value of the Name ASN.1 type is a character string conforming to the Name production of XML version 1.0 [XML10].

```
Name ::= UTF8String (CONSTRAINED BY
                { -- conforms to the Name production of XML -- })
```

4.5. The QName Type

A value of the QName ASN.1 type describes an expanded name [XMLNS10], which appears as a qualified name [XMLNS10] in an RXER encoding.

RXER has special provisions for encoding values of the QName type (see Section 6.7.11). For other encoding rules, a value of the Qname type is encoded according to the following ASN.1 type definition (with AUTOMATIC TAGS):

```
QName ::= SEQUENCE {
    namespace-name  AnyURI OPTIONAL,
    local-name      NCName
}
```

The namespace-name component holds the namespace name of the expanded name. If the namespace name of the expanded name has no value, then the namespace-name component is absent.

Aside: A namespace name can be associated with ASN.1 types and top-level NamedType instances by using the TARGET-NAMESPACE encoding instruction.

The local-name component holds the local name of the expanded name.

5. Expanded Names for ASN.1 Types

A TypeAssignment in ASN.1 associates a typereference with a Type. For RXER and Abstract Syntax Notation X (ASN.X) [ASN.X], a TypeAssignment is also regarded as associating an expanded name [XMLNS10] with the Type. The local name of the expanded name is the typereference on the left-hand side of the TypeAssignment. If the target namespace [RXEREI] of the ASN.1 module in which the TypeAssignment is defined is not absent, then the namespace name of the expanded name is that target namespace; otherwise, the namespace name of the expanded name has no value.

A Type that is a BuiltinType or ReferencedType that is one of the productions in Table 1 is regarded as a reference to a built-in ASN.1 type. These built-in types also have expanded names. In each case, the local name of the expanded name is as indicated in Table 1, and the namespace name of the expanded name is "urn:ietf:params:xml:ns:asnx".

Table 1: Local Names for Built-in Types

ASN.1 Production	Local Name
BitStringType	
without a NamedBitList	BIT-STRING
BooleanType	BOOLEAN
CharacterStringType	
RestrictedCharacterStringType	
BMPString	BMPString
GeneralString	GeneralString
GraphicString	GraphicString
IA5String	IA5String
ISO646String	ISO646String
NumericString	NumericString
PrintableString	PrintableString
TeletexString	TeletexString
T61String	T61String
UniversalString	UniversalString
UTF8String	UTF8String
VideotexString	VideotexString
VisibleString	VisibleString
UnrestrictedCharacterStringType	CHARACTER-STRING
EmbeddedPDVType	EMBEDDED-PDV
ExternalType	EXTERNAL
IntegerType	
without a NamedNumberList	INTEGER
NullType	NULL
ObjectIdentifierType	OBJECT-IDENTIFIER
OctetStringType	OCTET-STRING
RealType	REAL
RelativeOIDType	RELATIVE-OID
UsefulType	
GeneralizedTime	GeneralizedTime
UTCTime	UTCTime
ObjectDescriptor	ObjectDescriptor

When the expanded name for an ASN.1 type is used in an RXER encoding, it appears as a qualified name [XMLNS10][XMLNS11]. The namespace prefix for the qualified name is determined according to Section 6.7.11.1.

If a compatible XML Schema translation of an ASN.1 specification is provided (see Section 6.4), then that schema SHOULD associate the same expanded name with the XML Schema translation of an ASN.1 type.

Definition (namespace-qualified reference): An ASN.1 Type is a namespace-qualified reference if one of the following applies:

(1) the Type is a typereference (not a DummyReference) or an ExternalTypeReference in a DefinedType in a ReferencedType, the ASN.1 module in which the referenced type is defined has a TARGET-NAMESPACE encoding instruction, the referenced type is not directly or indirectly an open type [X.681], and the referenced type is not directly or indirectly the Markup type (Section 4.1), or

(2) the Type is a BuiltinType or ReferencedType that is one of the productions in Table 1.

The type definition referenced by a namespace-qualified reference will have an expanded name with a value for the namespace name.

6. Encoding Rules

With respect to RXER, ASN.1 abstract values are uniformly regarded as analogous to the content and attributes of an element, or just an attribute value, not complete elements or attributes in their own right. Elements and attributes in an RXER encoding are defined by ASN.1 NamedType notation. Since elements are the fundamental discrete structures of an XML document, the notion of a NamedType having a value that can be encoded is useful for descriptive purposes (particularly for describing the RXER encoding of values of the ASN.1 combining types). There is no conceptual basis in X.680 [X.680] for talking about the value of a NamedType, or its encoding, so the terminology is introduced here.

Definition (value of a NamedType): An abstract value of the Type in a NamedType is also a value of that NamedType. The RXER encoding of the value of a NamedType is the RXER encoding of the abstract value of the Type encapsulated according to the definition of that NamedType.

This document does not refer to a value of a NamedType as being an abstract value so as to remain consistent with X.680. An abstract value is exclusively a value of an ASN.1 type.

A complete ASN.1 encoding is traditionally the encoding of an abstract value, but it is more natural to think of an XML document as being the RXER encoding of a value of a NamedType (because an XML document has a single root element that contains all the other elements and attributes). The ASN.1 basic notation does not allow a NamedType to appear on its own, outside of an enclosing combining type. That is, the basic notation does not have a concept analogous to a global element or attribute definition. However, an ASN.1 specification may use an RXER encoding control section [RXEREI] to define global elements and attributes using the NamedType notation. A NamedType that is not contained in an ASN.1 type definition is called a top-level NamedType [RXEREI]. Thus, an RXER encoding would typically be described as the encoding of a value of a top-level NamedType.

Section 6.2 describes how a value of a NamedType is encoded. Section 6.3 defines an alternative method for encoding the document element of an XML document when a top-level NamedType is not specified. Section 6.4 describes how the encodings of ASN.1 values can be embedded in an XML document where the other parts of the document are validated by an XML Schema.

The RXER encoding of an abstract value, or the encoding of a value of a NamedType, is described as a translation into a synthetic Infoset, which is then serialized as XML. This separation has been chosen for descriptive convenience and is not intended to impose any particular architecture on RXER implementations. An RXER encoder is free to encode an ASN.1 value directly to XML provided the result is equivalent to following the two stage procedure described in this document.

The process of translating an abstract value into an Infoset is described as producing either:

(1) a string of characters that either becomes part of the [normalized value] of an attribute item or becomes character items among the [children] of an enclosing element item, or

(2) a collection of zero or more attribute items contributing to the [attributes] of an enclosing element item, plus a series of zero or more character, element, processing instruction (PI), or comment items contributing to the [children] of the enclosing element item.

NamedType notation in the ASN.1 specification controls whether the translation of an abstract value is encapsulated in an element item or in an attribute item.

Sections 6.5 to 6.10 describe the translation of abstract values into an Infoset for each of the ASN.1 type notations.

Section 6.11 describes post-processing of namespace prefixes for CRXER encodings.

Section 6.12 specifies how the Infoset translation is serialized as XML.

This specification assumes that the COMPONENTS OF transformation specified in X.680, Clause 24.4 [X.680] has already been applied to all relevant types.

Examples of RXER encodings in the following sections use a <value> start-tag and </value> end-tag to hold attributes and delimit the content. These start-tags and end-tags are for illustration only and are not part of the encoding of an abstract value. In normal use, the name of the enclosing element is provided by the context of the type of the abstract value, e.g., a NamedType in an enclosing SEQUENCE type.

An RXER decoder is a conforming XML processor [XML10][XML11].

6.1. Identifiers

An identifier, as defined in ASN.1 notation (Clause 11.3 of X.680 [X.680]), is a character string that begins with a Latin lowercase letter (U+0061-U+007A) and is followed by zero, one or more Latin letters (U+0041-U+005A, U+0061-U+007A), decimal digits (U+0030-U+0039), and hyphens (U+002D). A hyphen is not permitted to be the last character, and a hyphen is not permitted to be followed by another hyphen. The case of letters in an identifier is always significant.

ASN.1 identifiers are used for the [local name] of attribute and element items, and may also appear in the character data content of elements or the values of attributes. RXER encoding instructions can be used to substitute an NCName [XMLNS10] for an identifier.

6.2. Component Encodings

The translation of the value of a NamedType is the translation of the abstract value of the Type of the NamedType encapsulated according to the definition of that NamedType. This section specifies the form of this encapsulation.

6.2.1. Referenced Components

A value of a NamedType that is subject to a COMPONENT-REF encoding instruction is translated as a value of the top-level NamedType referenced by the encoding instruction.

6.2.2. Element Components

A value of a NamedType that is not subject to an ATTRIBUTE, ATTRIBUTE-REF, GROUP, or SIMPLE-CONTENT encoding instruction is translated as an element item, either as a child element item added to the [children] of the enclosing element item or as the document element item added to the [children] and [document element] of the document item. If the element item is a child element item, then the [parent] is the enclosing element item; otherwise, the [parent] is the document item.

The [local name] of the element item is the local name of the expanded name of the NamedType (see [RXEREI]).

> Aside: If there are no NAME, ATTRIBUTE-REF, COMPONENT-REF, ELEMENT-REF, or REF-AS-ELEMENT encoding instructions, then the local name of the expanded name of a NamedType is the same as the identifier of the NamedType.

If the namespace name of the expanded name has no value, then the [namespace name] of the element item has no value (i.e., the element's name is not namespace qualified); otherwise, the [namespace name] is the namespace name of the expanded name.

If the type of the NamedType is directly or indirectly the Markup type, then the [in-scope namespaces] and [namespace attributes] of the element item are constructed as specified in Section 6.10; otherwise, the [in-scope namespaces] and [namespace attributes] of the element item are constructed as specified in Section 6.2.2.1.

If the [namespace name] of the element item has no value, then the [prefix] of the element item has no value; else if the type of the NamedType is not directly or indirectly the Markup type, then the

[prefix] of the element item is determined as specified in Section 6.2.2.2; otherwise, the [prefix] is determined by the Markup value as specified in Section 6.10.

The element item becomes the enclosing element item for the translation of the value of the Type of the NamedType.

For a non-canonical RXER encoding, if the type of the NamedType is not directly or indirectly the Markup type, then PI and comment items MAY be added to the [children] of the element item (before or after any other items). The element item becomes the [parent] for each PI and comment item. These particular PI and comment items in a received RXER encoding MAY be discarded by an application.

> Aside: There is no provision for representing comments and PIs in ASN.1 abstract values of types other than the Markup type. These items will be lost if the abstract value is re-encoded using a different set of encoding rules.

For a non-canonical RXER encoding, an attribute item with the [local name] "type" and the [namespace name] "http://www.w3.org/2001/XMLSchema-instance" (i.e., xsi:type [XSD1]) SHOULD be added to the [attributes] of the element item if the corresponding NamedType is subject to a TYPE-AS-VERSION encoding instruction and MAY be added to the [attributes] of the element item if the Type of the corresponding NamedType is a namespace-qualified reference (see Section 5). The [prefix] of this attribute item is determined as specified in Section 6.2.3.1. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced type, with the namespace prefix determined as specified in Section 6.7.11.1. The element item is the [owner element] for the attribute item.

> Aside: Where a compatible XML Schema translation of the ASN.1 specification has been provided, the xsi:type attribute indicates to an XML Schema validator which type definition it should use for validating the RXER encoding.

> Aside: An xsi:type attribute is generally not permitted in a CRXER encoding. Section 6.4 describes some circumstances where it is required in a CRXER encoding. An xsi:type attribute might also appear in a CRXER encoding if it is contained in a value of the Markup type.

For a non-canonical RXER encoding, if the type of the NamedType is not directly or indirectly the Markup type, then attribute items with the [local name] "schemaLocation" or "noNamespaceSchemaLocation" and the [namespace name] "http://www.w3.org/2001/XMLSchema-instance"

[XSD1] MAY be added to the [attributes] of the element item. The [prefix] for each of these attribute items is determined as specified in Section 6.2.3.1. The [normalized value] of these attribute items MUST reference a compatible XML Schema translation of the ASN.1 specification. The element item is the [owner element] for the attribute items.

6.2.2.1. Namespace Properties for Elements

This section describes how the [in-scope namespaces] and [namespace attributes] of an element item are constructed when the content and attributes of the element item are not described by a value of the Markup type (otherwise, see Section 6.10).

The [in-scope namespaces] property of the element item initially contains only the mandatory namespace item for the "xml" prefix [INFOSET].

For a CRXER encoding, if the element item is not the [document element] of the document item and the [in-scope namespaces] property of the element item's [parent] contains a namespace item for the default namespace, then a namespace declaration attribute item that undeclares the default namespace (see Section 3) SHALL be added to the element item's [namespace attributes].

Definition (default namespace restricted): With respect to an element item, the default namespace is restricted if:

(1) the [namespace name] of the element item has no value (i.e., the element's name is not namespace qualified), or

(2) the element item is the enclosing element item for a value of the UNION type where the member attribute will be required (see Section 6.7.14), or

(3) the element item is the enclosing element item for a value of the QName type where the namespace-name component is absent (see Section 6.7.11). This includes the case where the translation of the QName value is contained in the [normalized value] of an attribute item in the [attributes] of the element item.

For a non-canonical RXER encoding, if the element item is not the [document element] of the document item and the [in-scope namespaces] property of the element item's [parent] contains a namespace item for the default namespace, then either:

(1) that item is copied to the [in-scope namespaces] of the element item, or

(2) a namespace declaration attribute item that declares the default namespace is added to the element item's [namespace attributes] (the namespace name is the encoder's choice), and an equivalent namespace item is added to the [in-scope namespaces] of the element item, or

(3) a namespace declaration attribute item that undeclares the default namespace is added to the element item's [namespace attributes].

Options (1) and (2) SHALL NOT be used if the default namespace is restricted with respect to the element item.

For a CRXER encoding, if the element item is not the [document element] of the document item and the element item is not required to be self-contained, then all the namespace items in the [in-scope namespaces] of the [parent], excluding the namespace item for the "xml" prefix and any namespace item for the default namespace, are copied to the [in-scope namespaces] of the element item.

For a non-canonical RXER encoding, if the element item is not the [document element] of the document item and the element item is not required to be self-contained, then any subset (including none or all) of the namespace items in the [in-scope namespaces] of the [parent], excluding certain items, is copied to the [in-scope namespaces] of the element item. The excluded items that MUST NOT be copied are: the namespace item for the "xml" prefix, any namespace item for the default namespace, and any namespace item that matches the [prefix], but not the [namespace name], of a namespace item retained for the re-encoding of an unknown attribute item (see Section 6.8.8) or an unknown alternative of a UNION (see Section 6.7.14).

> Aside: The descriptive approach used by this document only allows a namespace prefix to be used by a new namespace item if it is not currently used by another namespace item in the [in-scope namespaces]. By not inheriting a namespace item, the prefix of that namespace is again available for reuse without fear of breaking an existing dependency on the prefix.

Element items that are required to be self-contained inherit none of the namespace items in the [in-scope namespaces] of the [parent].

Any namespace item that is retained for the re-encoding of an unknown attribute item (Section 6.8.8) or an unknown alternative of a UNION (Section 6.7.14) and which is not in the [in-scope namespaces] of the element item MUST be added to the [in-scope namespaces]. An

equivalent namespace declaration attribute item MUST be added to the [namespace attributes] of the element item.

Definition (unused namespace prefix): A namespace prefix is unused if it does not match the [prefix] of any namespace item in the [in-scope namespaces] of the element item.

For a non-canonical RXER encoding, if the type of the NamedType is not directly or indirectly the Markup type, then additional namespace declaration attribute items for currently unused namespace prefixes MAY be added to the [namespace attributes] of the element item. An equivalent namespace item MUST be added to the [in-scope namespaces] of the element item for each additional namespace declaration attribute item.

For a non-canonical RXER encoding, if the type of the NamedType is not directly or indirectly the Markup type, and the [in-scope namespaces] property of the element item does not contain a namespace item for the default namespace, and the default namespace is not restricted with respect to the element item, then a namespace declaration attribute item for the default namespace MAY be added to the [namespace attributes] of the element item, in which case an equivalent namespace item MUST be added to the [in-scope namespaces] of the element item.

Whenever a namespace declaration attribute item is added to an element item's [namespace attributes], the [owner element] of the attribute item is set to the element item.

6.2.2.2. Namespace Prefixes for Element Names

This section describes how the [prefix] of an element item is determined when the element item has a value for its [namespace name] and the content and attributes of the element item are not described by a value of the Markup type (otherwise, see Section 6.10).

For a CRXER encoding, if the [namespace name] of the element item has a value, then the [prefix] of the element item is any unused non-canonical namespace prefix unless the [in-scope namespaces] property of the element item contains a namespace item with the same [namespace name] as the element item. In that case, the [prefix] of that namespace item SHALL be used as the [prefix] of the element item.

> Aside: These prefixes will be rewritten to canonical namespace prefixes during the final step in producing the Infoset translation (see Section 6.11). Canonical namespace prefixes are not used here in the first instance because canonicalization

depends on knowing the final [namespace attributes] produced by encoding the abstract value of the type of the NamedType. If an implementation looks ahead to determine this final set prior to translating the abstract value, then it can assign the appropriate canonical namespace prefix in this step and skip the rewriting step.

For a non-canonical RXER encoding, if the [namespace name] has a value, then the [prefix] of the element item is any unused namespace prefix unless the [in-scope namespaces] property of the element item contains a namespace item with the same [namespace name] as the element item. In that case, the [prefix] of that namespace item MAY be used as the [prefix] of the element item. Note that the [prefix] of a namespace item for the default namespace has no value.

If the [prefix] of the element item is an unused namespace prefix, then a namespace declaration attribute item associating the namespace prefix with the namespace name MUST be added to the [namespace attributes] of the element item, and a corresponding namespace item MUST be added to the [in-scope namespaces] of the element item.

Aside: The [local name] of the namespace declaration attribute item is the same as the [prefix] of the element item, the [namespace name] of the attribute item is "http://www.w3.org/2000/xmlns/", and the [normalized value] of the attribute item is the same as the [namespace name] of the element item. The namespace item has the same [prefix] and [namespace name] as the element item.

6.2.3. Attribute Components

A value of a NamedType subject to an ATTRIBUTE or ATTRIBUTE-REF encoding instruction is translated as an attribute item added to the [attributes] of the enclosing element item (which becomes the [owner element] of the attribute item).

The [local name] of the attribute item is the local name of the expanded name of the NamedType (see [RXEREI]).

If the namespace name of the expanded name has no value, then the [namespace name] of the attribute item has no value; otherwise, the [namespace name] is the namespace name of the expanded name.

If the [namespace name] has a value, then the [prefix] of the attribute item is determined as specified in Section 6.2.3.1; otherwise, the [prefix] of the attribute item has no value.

The [normalized value] of the attribute item is the translation of the value of the Type of the NamedType.

For completeness, the [specified] property is set to true, the [attribute type] has no value, and the value of the [references] property is set to unknown.

6.2.3.1. Namespace Prefixes for Attribute Names

This section applies when an attribute item with a value for its [namespace name] is added to the [attributes] of an element item.

For a CRXER encoding, the [prefix] of the attribute item is any unused non-canonical namespace prefix unless the [in-scope namespaces] property of the [owner element] contains a namespace item with a value for the [prefix] (i.e., is not a namespace item for the default namespace) and the same [namespace name] as the attribute item. In that case, the [prefix] of that namespace item SHALL be used as the [prefix] of the attribute item.

For a non-canonical RXER encoding, the [prefix] of the attribute item is any unused namespace prefix unless the [in-scope namespaces] property of the [owner element] contains a namespace item with a value for the [prefix] and the same [namespace name] as the attribute item. In that case, the [prefix] of that namespace item MAY be used as the [prefix] of the attribute item.

If the [prefix] of the attribute item is an unused namespace prefix, then a namespace declaration attribute item associating the namespace prefix with the namespace name MUST be added to the [namespace attributes] of the [owner element], and a corresponding namespace item MUST be added to the [in-scope namespaces] of the [owner element].

6.2.4. Unencapsulated Components

A value of a NamedType subject to a GROUP or SIMPLE-CONTENT encoding instruction is translated as the value of the Type of the NamedType, i.e., without encapsulation in an element item or attribute item. Consequently, the enclosing element item for the translation of the value of the NamedType is also the enclosing element item for the translation of the value of the Type of the NamedType.

6.2.5. Examples

Consider this type definition:

```
CHOICE {
    one    [0] BOOLEAN,
    two    [1] [RXER:ATTRIBUTE] INTEGER,
    three  [2] [RXER:NAME AS "THREE"] OBJECT IDENTIFIER,
    four   [3] [RXER:ATTRIBUTE-REF {
                   namespace-name "http://www.example.com",
                   local-name     "foo" }] UTF8String,
    five   [4] [RXER:ELEMENT-REF {
                   namespace-name "http://www.example.com",
                   local-name     "bar" }] Markup,
    six    [5] [RXER:GROUP] SEQUENCE {
        seven  [0] [RXER:ATTRIBUTE] INTEGER,
        eight  [1] INTEGER
    }
}
```

The content and attributes of each of the following <value> elements are the RXER encoding of a value of the above type:

```
<value>
 <one>true</one>
</value>

<value two="100"/>

<value>
 <THREE>2.5.4.3</THREE>
</value>

<value xmlns:ex="http://www.example.com"
       ex:foo="a string"/>

<value>
 <ex:bar xmlns:ex="http://www.example.com">another string</ex:bar>
</value>

<value seven="200">
 <eight>300</eight>
</value>
```

6.3. Standalone Encodings

A typical RXER encoding is the encoding of a value of a nominated top-level NamedType. An abstract value MAY be encoded as an XML document without nominating an explicit top-level NamedType by invoking a Standalone RXER Encoding or Standalone CRXER Encoding.

In a Standalone RXER Encoding or Standalone CRXER Encoding, the abstract value is encoded as the value of a notional NamedType where the identifier of the NamedType is "value" and the Type of the NamedType is the type of the abstract value. The NamedType is assumed to be subject to no encoding instructions.

> Aside: Thus, the element item corresponding to the document element will have the [local name] "value" and no value for the [namespace name] and [prefix].

If RXER is chosen as the transfer syntax in an EMBEDDED PDV value, then the data-value OCTET STRING SHALL contain a Standalone RXER encoding.

If CRXER is chosen as the transfer syntax in an EMBEDDED PDV value, then the data-value OCTET STRING SHALL contain a Standalone CRXER encoding.

If RXER is chosen as the transfer syntax in an EXTERNAL value, then the octet-aligned OCTET STRING or arbitrary BIT STRING SHALL contain a Standalone RXER encoding.

If CRXER is chosen as the transfer syntax in an EXTERNAL value, then the octet-aligned OCTET STRING or arbitrary BIT STRING SHALL contain a Standalone CRXER encoding.

6.4. Embedded ASN.1 Values

The reference encoding instructions [RXEREI] allow XML Schema definitions to be referenced from an ASN.1 specification. It is also possible to reference an ASN.1 type or top-level NamedType from an XML Schema definition or from an information item validated by an XML Schema wildcard. The manner in which an XML Schema definition references an ASN.1 type or top-level NamedType has an effect on the CRXER encoding of a value of the type or top-level NamedType.

This section also applies to XML Schema definitions that validate information items that are contained in a value of the Markup type.

Aside: So the document element of an XML document might be described by an XML Schema definition that at some point references an ASN.1 definition that uses a reference encoding instruction to reference another XML Schema definition that then references another ASN.1 definition, and so on.

In each of the following cases, an element or attribute item is only permitted to be, or to encapsulate, an RXER Infoset translation of an ASN.1 value if an XML Schema element declaration or ASN.1 NamedType is known for the [parent] element item ([owner element] in the case of an attribute declaration), for the [parent] of the [parent] element item, and so on, to the document element of the XML document. This condition is not satisfied by a NamedType where the Type is directly or indirectly the Markup type and the NamedType is not subject to a reference encoding instruction.

Aside: An element declaration becomes known for an element item through assessment [XSD1]. A NamedType becomes known for an element item through decoding.

Aside: If an XML Schema element declaration or ASN.1 NamedType is not known for an element item, then the type of the element item and the type of every nested element item are treated as unknown. Although an xsi:type attribute definitively identifies the type of an element item even if an element declaration for the element item is not known, this attribute is generally optional in an RXER encoding and so cannot be relied upon when seen in isolation from an element declaration. Although only top-level NamedType instances can have namespace-qualified names in the current RXER specification, a future version may allow nested NamedType instances to also have namespace-qualified names, in which case it will not necessarily be possible to distinguish a nested NamedType from a top-level NamedType without knowledge of the type of the [parent] element item.

An ASN.1 type with an expanded name (Section 5) MAY be referenced by the type attribute of an XML Schema element declaration. The reference takes the form of a qualified name for the expanded name. An element item validated by such an element declaration encapsulates the Infoset translation of an abstract value of the ASN.1 type. The [namespace name] and [local name] of the element item are determined by the XML Schema element declaration. The remaining properties are determined according to RXER. The element item MUST be self-contained for a CRXER encoding.

Aside: The element item is not required to be self-contained for a non-canonical RXER encoding.

A top-level NamedType MAY be referenced by the ref attribute of an XML Schema element declaration if the NamedType is not subject to an ATTRIBUTE encoding instruction. The reference takes the form of a qualified name for the expanded name of the top-level NamedType [RXEREI]. An element item validated by such an element declaration is the Infoset translation of a value of the referenced top-level NamedType. All the properties of the element item are determined according to RXER. The element item MUST be self-contained for a CRXER encoding.

A top-level NamedType MAY be referenced by the ref attribute of an XML Schema attribute declaration if the NamedType is subject to an ATTRIBUTE encoding instruction and the definition of the type of the NamedType does not depend on the QName type in any way. An attribute item validated by such an attribute declaration is the Infoset translation of a value of the referenced top-level NamedType, except that whatever valid [prefix] is initially chosen for the attribute item MUST be preserved in any re-encoding. The remaining properties of the attribute item are determined according to RXER.

> Aside: The exclusion of the QName type means that the attribute value is not dependent upon any namespace declarations of its parent element item.

An element item that is validated by an XML Schema element declaration that has the ur-type (i.e., anyType) as its type definition MAY encapsulate the Infoset translation of a value of an ASN.1 type with an expanded name. The [namespace name] and [local name] of the element item are determined by the XML Schema element declaration. The remaining properties of the element item are determined according to RXER. The [attributes] of the element item SHALL contain an attribute item with the [local name] "type" and the [namespace name] "http://www.w3.org/2001/XMLSchema-instance" (i.e., an xsi:type attribute). The [prefix] of this attribute item is determined as specified in Section 6.2.3.1. The [normalized value] of this attribute item is a qualified name for the expanded name of the ASN.1 type, with the namespace prefix determined as specified in Section 6.7.11.1. The element item MUST be self-contained for a CRXER encoding.

An element item that is validated by an XML Schema wildcard (i.e., <xs:any/>) MAY be the Infoset translation of a value of a top-level NamedType that is not subject to an ATTRIBUTE encoding instruction and comes from an ASN.1 module with a target namespace [RXEREI] that satisfies the namespace constraint of the wildcard. All the properties of the element item are determined according to RXER. The element item MUST be self-contained for a CRXER encoding.

An attribute item that is validated by an XML Schema wildcard (i.e., <xs:anyAttribute/>) MAY be the Infoset translation of a value of a top-level NamedType if the NamedType is subject to an ATTRIBUTE encoding instruction, comes from an ASN.1 module with a target namespace that satisfies the namespace constraint of the wildcard, and has a type that does not depend on the QName type in any way. Whatever valid [prefix] is initially chosen for the attribute item MUST be preserved in any re-encoding. The remaining properties of the attribute item are determined according to RXER.

No other mechanisms for referencing an ASN.1 type or top-level NamedType from a different XML schema language are supported in this version of RXER. In particular, this excludes an ASN.1 type being used as the base type in an XML Schema derivation by extension or restriction, as a member type for an XML Schema union type, as an item type for an XML Schema list type, or as the type in an XML Schema attribute declaration.

A fully conformant RXER implementation will understand both ASN.1 and XML Schema and will recognize the transitions between information items controlled by ASN.1 definitions and those controlled by XML Schema definitions. However, a purely XML Schema validator used to assess the validity of an RXER encoding will perceive any reference to an ASN.1 type or top-level NamedType as an unresolved reference. In order to enable such assessment, it is desirable to provide an XML Schema translation of the ASN.1 definitions being referenced from an XML Schema. Although XML Schema and ASN.1 are broadly similar, they each have unique features that cannot be adequately expressed in the other language, so a semantically equivalent translation is not possible in the general case. Fortunately, to simply achieve successful assessment it is sufficient for the XML Schema translation of an ASN.1 specification to be compatible with that ASN.1 specification. That is, the XML Schema translation MUST be constructed such that every correct RXER encoding is assessed as valid. Although not ideal, it is acceptable for the XML Schema to assess some incorrect RXER encodings as also being valid (a conformant RXER decoder will, of course, reject such an encoding).

The simplest compatible XML Schema translation of an ASN.1 module is one in which every type is equivalent to the XML Schema ur-type. For example, given an ASN.1 type with the reference name MyType, a sufficient compatible XML Schema type definition is:

```
<xs:complexType name="MyType" mixed="true">
 <xs:sequence>
  <xs:any processContents="lax"
           minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:anyAttribute processContents="lax"/>
</xs:complexType>

    OR

<xs:complexType name="MyType">
 <xs:complexContent>
  <xs:extension base="xs:anyType"/>
 </xs:complexContent>
</xs:complexType>
```

Aside: Because of the possible presence of an asnx:context attribute (Section 6.8.8.1), it is easiest to assume that all ASN.1 types translate into XML Schema complex types.

Given an ASN.1 top-level NamedType that is not subject to an ATTRIBUTE encoding instruction and has the reference name myElement, a sufficient compatible XML Schema element declaration is:

```
<xs:element name="myElement"/>
```

Given an ASN.1 top-level NamedType that is subject to an ATTRIBUTE encoding instruction and has the reference name myAttribute, a sufficient compatible XML Schema attribute declaration is:

```
<xs:attribute name="myAttribute"/>
```

An application specification that mixes ASN.1 and XML Schema is free to provide a stricter translation of its ASN.1 definitions; however, a more thorough treatment for translating an ASN.1 module into an XML Schema is out of scope for this document.

6.5. Type Referencing Notations

A value of a type with a defined type name is translated according to the type definition on the right-hand side of the type assignment for the type name.

A value of a type denoted by the use of a parameterized type with actual parameters is translated according to the parameterized type with the DummyReferences [X.683] substituted with the actual parameters.

A value of a constrained type is translated as a value of the type without the constraint. See X.680 [X.680] and X.682 [X.682] for the details of ASN.1 constraint notation.

A prefixed type [X.680-1] associates an encoding instruction with a type. A value of a prefixed type is translated as a value of the type without the prefix.

> Aside: This does not mean that RXER encoding instructions are ignored. It is simply easier to describe their effects in relation to specific built-in types, rather than as the translation of a value of a prefixed type.

A tagged type is a special case of a prefixed type. A value of a tagged type is translated as a value of the type without the tag. ASN.1 tags do not appear in the XML encodings defined by this document.

A value of a fixed type denoted by an ObjectClassFieldType is translated according to that fixed type (see Section 6.9 for the case of an ObjectClassFieldType denoting an open type).

A value of a selection type is translated according to the type referenced by the selection type. Note that component encoding instructions are not inherited by the type referenced by a selection type [RXEREI].

A value of a type described by TypeFromObject notation [X.681] is translated according to the denoted type.

A value of a type described by ValueSetFromObjects notation [X.681] is translated according to the governing type.

6.6. TypeWithConstraint, SEQUENCE OF Type, and SET OF Type

For the purposes of this document, a TypeWithConstraint is treated as if it were the parent type [X.680] (either a SEQUENCE OF or SET OF type).

For example,

```
SEQUENCE SIZE(1..MAX) OF SomeType

   is treated like

SEQUENCE OF SomeType
```

Additionally, a "SEQUENCE OF Type" (including the case where it is the parent type for a TypeWithConstraint) is treated as if it were a "SEQUENCE OF NamedType", where the identifier of the NamedType is assumed to be "item". Similarly, a "SET OF Type" (including the case where it is the parent type for a TypeWithConstraint) is treated as if it were a "SET OF NamedType", where the identifier of the NamedType is assumed to be "item".

For example,

SEQUENCE SIZE(1..MAX) OF SomeType

is ultimately treated like

SEQUENCE OF item SomeType

6.7. Character Data Translations

For the majority of ASN.1 built-in types, encodings of values of those types never have element content. The encoding of a value of an ASN.1 combining type (except a UNION or LIST type) typically has element content.

For those types that do not produce element content, the translation of an abstract value is described as a character string of ISO 10646 characters [UCS]. This character data translation will be destined to become either part of the [normalized value] of an attribute item, or a series of character items in the [children] of an element item (which becomes the [parent] for the character items). The case that applies is determined in accordance with Section 6.2.

For a non-canonical RXER encoding, if the type of the abstract value is not directly or indirectly a restricted character string type, the NULL type, or a UNION type, then leading and/or trailing white space characters MAY be added to the character data translation.

Aside: White space characters are significant in the encoding of a value of a restricted character string type, and a restricted character string type can be a member type of a UNION type. The encoding of a NULL value produces no character data.

Aside: Optional white space characters are not permitted in a CRXER encoding.

For a non-canonical RXER encoding, if the type of the abstract value is directly or indirectly the AnyURI, NCName, or Name type, then leading and trailing white space characters MAY be added to the character data translation.

Aside: These types are indirectly a restricted character string type (UTF8String); however, their definitions exclude white space characters, so any white space characters appearing in an encoding are not part of the abstract value and can be safely ignored. This exception does not apply to other subtypes of a restricted character string type that happen to exclude white space characters.

6.7.1. Restricted Character String Types

The character data translation of a value of a restricted character string type is the sequence of characters in the string.

Depending on the ASN.1 string type, and an application's internal representation of that string type, a character may need to be translated to or from the equivalent ISO 10646 character code [UCS]. The NumericString, PrintableString, IA5String, VisibleString (ISO646String), BMPString, UniversalString, and UTF8String character encodings use the same character codes as ISO 10646. For the remaining string types (GeneralString, GraphicString, TeletexString, T61String, and VideotexString), see X.680 [X.680].

The null character (U+0000) is not a legal character for XML. It is omitted from the character data translation of a string value.

Certain other control characters are legal for XML version 1.1, but not for version 1.0. If any string value contains these characters, then the RXER encoding must use XML version 1.1 (see Section 6.12).

All white space characters in the RXER encoding of a value of a restricted character string type (excluding the AnyURI, NCName, and Name subtypes) are significant, i.e., part of the abstract value.

Examples

The content of each of the following <value> elements is the RXER encoding of an IA5String value:

```
<value> Don't run with scissors! </value>

<value>Markup (e.g., &lt;value&gt;) has to be escaped.</value>

<value>Markup (e.g., <![CDATA[<value>]]>)
has to be escaped. </value>
```

6.7.2. BIT STRING

The character data translation of a value of the BIT STRING type is either a binary digit string, a hexadecimal digit string, or a list of bit names.

A binary digit string is a sequence of zero, one, or more of the binary digit characters '0' and '1' (i.e., U+0030 and U+0031). Each bit in the BIT STRING value is encoded as a binary digit in order from the first bit to the last bit.

For a non-canonical RXER encoding, if the BIT STRING type has a NamedBitList, then trailing zero bits MAY be omitted from a binary digit string.

A hexadecimal digit string is permitted if and only if the number of bits in the BIT STRING value is zero or a multiple of eight and the character data translation is destined for the [children] of an element item.

A hexadecimal digit string is a sequence of zero, one, or more pairs of the hexadecimal digit characters '0'-'9', 'A'-'F', and 'a'-'f' (i.e., U+0030-U+0039, U+0041-U+0046 and U+0061-U+0066). Each group of eight bits in the BIT STRING value is encoded as a pair of hexadecimal digits where the first bit is the most significant. An odd number of hexadecimal digits is not permitted. The characters 'a'-'f' (i.e., U+0061-U+0066) SHALL NOT be used in the CRXER encoding of a BIT STRING value. If a hexadecimal digit string is used, then the enclosing element's [attributes] MUST contain an attribute item with the [local name] "format", the [namespace name] "urn:ietf:params:xml:ns:asnx", and the [normalized value] "hex" (i.e., asnx:format="hex"). The [prefix] of the attribute item is determined as specified in Section 6.2.3.1.

> Aside: The hexadecimal digit string is intended to conform to the lexical representation of the XML Schema [XSD2] hexBinary data type.

For a non-canonical RXER encoding, if the preconditions for using a hexadecimal digit string are satisfied, then a hexadecimal digit string MAY be used.

A list of bit names is permitted if and only if the BIT STRING type has a NamedBitList and each '1' bit in the BIT STRING value has a corresponding identifier in the NamedBitList.

> Aside: ASN.1 does not require that an identifier be assigned for every bit.

A list of bit names is a sequence of names for the '1' bits in the BIT STRING value, in any order, each separated from the next by at least one white space character. If the BitStringType is not subject to a VALUES encoding instruction, then each '1' bit in the BIT STRING value is represented by its corresponding identifier from the NamedBitList. If the BitStringType is subject to a VALUES encoding instruction, then each '1' bit in the BIT STRING value is represented by the replacement name [RXEREI] for its corresponding identifier.

For a CRXER encoding, if the BIT STRING type has a NamedBitList, then a binary digit string MUST be used, and trailing zero bits MUST be omitted from the binary digit string; else if the number of bits in the BIT STRING value is greater than or equal to 64, and the preconditions for using a hexadecimal digit string are satisfied, then a hexadecimal digit string MUST be used; otherwise, a binary digit string MUST be used.

Aside: Because the asnx:format attribute adds an overhead to a hexadecimal encoding (including a namespace declaration for the "asnx" prefix), a bit string of less than 64 bits is more compactly encoded as a binary digit string.

Examples

Consider this type definition:

```
BIT STRING { black(0), red(1), orange(2), yellow(3),
    green(4), blue(5), indigo(6), violet(7) }
```

The content and attributes of each of the following <value> elements are an RXER encoding of the same abstract value:

```
<value>  green violet  orange</value>

<value> 001<!--Orange-->01001 </value>

<value xmlns:asnx="urn:ietf:params:xml:ns:asnx"
       asnx:format="hex">
 29
</value>

<value>00101001</value>
```

The final case contains the CRXER encoding of the abstract value.

6.7.3. BOOLEAN

For a non-canonical RXER encoding, the character data translation of the BOOLEAN value TRUE is the string "true" or "1", at the encoder's discretion. For a CRXER encoding, the character data translation of the BOOLEAN value TRUE is the string "true".

For a non-canonical RXER encoding, the character data translation of the BOOLEAN value FALSE is the string "false" or "0", at the encoder's discretion. For a CRXER encoding, the character data translation of the BOOLEAN value FALSE is the string "false".

Aside: The RXER encoding of BOOLEAN values is intended to conform to the lexical representation of the XML Schema [XSD2] boolean data type.

Examples

The content of each of the following <value> elements is the RXER encoding of a BOOLEAN value:

```
<value>1</value>

<value>
    false
</value>

<value> fal<!-- a pesky comment -->se </value>
```

6.7.4. ENUMERATED

The character data translation of a value of an ENUMERATED type where the EnumeratedType is not subject to a VALUES encoding instruction is the identifier corresponding to the actual value.

Examples

Consider this type definition:

```
ENUMERATED { sunday, monday, tuesday,
    wednesday, thursday, friday, saturday }
```

The content of both of the following <value> elements is the RXER encoding of a value of the above type:

```
<value>monday</value>

<value>
    thursday
</value>
```

The character data translation of a value of an ENUMERATED type where the EnumeratedType is subject to a VALUES encoding instruction is the replacement name [RXEREI] for the identifier corresponding to the actual value.

Examples

Consider this type definition:

```
[RXER:VALUES ALL CAPITALIZED,
        sunday AS "SUNDAY", saturday AS "SATURDAY"]
    ENUMERATED { sunday, monday, tuesday,
        wednesday, thursday, friday, saturday }
```

The content of each of the following <value> elements is the RXER encoding of a value of the above type:

```
<value>SUNDAY</value>

<value>
    Monday
</value>

<value> Tuesday </value>
```

6.7.5. GeneralizedTime

The character data translation of a value of the GeneralizedTime type is a date, the letter 'T' (U+0054), a time of day, optional fractional seconds, and an optional time zone.

The date is two decimal digits representing the century, followed by two decimal digits representing the year, a hyphen ('-', U+002D), two decimal digits representing the month, a hyphen ('-', U+002D), and two decimal digits representing the day.

The time of day is two decimal digits representing the hour, followed by a colon (':', U+003A), two decimal digits representing the minutes, a colon (':', U+003A), and two decimal digits representing the seconds.

Note that the hours value "24" is disallowed [X.680].

A GeneralizedTime value with fractional hours or minutes is first converted to the equivalent time with whole minutes and seconds and, if necessary, fractional seconds.

The minutes are encoded as "00" if the GeneralizedTime value omits minutes. The seconds are encoded as "00" if the GeneralizedTime value omits seconds.

The fractional seconds part is a full stop ('.', U+002E) followed by zero, one, or more decimal digits (U+0030-U+0039). For a CRXER encoding, trailing zero digits (U+0030) in the fractional seconds SHALL be omitted, and the full stop SHALL be omitted if there are no following digits.

The time zone, if present, is either the letter 'Z' (U+005A) to indicate Coordinated Universal Time, a plus sign ('+', U+002B) followed by a time zone differential, or a minus sign ('-', U+002D) followed by a time zone differential.

A time zone differential indicates the difference between local time (the time specified by the preceding date and time of day) and Coordinated Universal Time. Coordinated Universal Time can be calculated from the local time by subtracting the differential.

For a CRXER encoding, a GeneralizedTime value with a time zone differential SHALL be encoded as the equivalent Coordinated Universal Time, i.e., the time zone will be "Z".

A local time GeneralizedTime value is not converted to Coordinated Universal Time for a CRXER encoding. Other canonical ASN.1 encoding rules specify that local times must be encoded as Coordinated Universal Time but do not specify a method to convert a local time to a Coordinated Universal Time. Consequently, canonicalization of local time values is unreliable and applications SHOULD NOT use local time.

A time zone differential is encoded as two decimal digits representing hours, a colon (':', U+003A), and two decimal digits representing minutes. The minutes are encoded as "00" if the GeneralizedTime value omits minutes from the time zone differential.

> Aside: The RXER encoding of GeneralizedTime values is intended to conform to the lexical representation of the XML Schema [XSD2] dateTime data type.

Examples

The content of each of the following <value> elements is the RXER encoding of a GeneralizedTime value:

```
<value>2004-06-15T12:00:00Z</value>

<value> 2004-06-15T02:00:00+10:00 </value>

<value>
    2004-06-15T12:00:00.5
</value>
```

6.7.6. INTEGER

For a CRXER encoding, the character data translation of a value of an IntegerType is a canonical number string representing the integer value.

A canonical number string is either the digit character '0' (U+0030), or an optional minus sign ('-', U+002D) followed by a non-zero decimal digit character (U+0031-U+0039) followed by zero, one, or more of the decimal digit characters '0' to '9' (U+0030-U+0039).

For a non-canonical RXER encoding, the character data translation of a value of the IntegerType without a NamedNumberList is a number string representing the integer value.

A number string is a sequence of one or more of the decimal digit characters '0' to '9' (U+0030-U+0039), with an optional leading sign, either '+' (U+002B) or '-' (U+002D). Leading zero digits are permitted in a number string for a non-canonical RXER encoding.

Aside: The RXER encoding of values of the IntegerType without a NamedNumberList is intended to conform to the lexical representation of the XML Schema [XSD2] integer data type.

For a non-canonical RXER encoding, if the IntegerType has a NamedNumberList, and the NamedNumberList defines an identifier for the actual value, and the IntegerType is not subject to a VALUES encoding instruction, then the character data translation of the value is either a number string or the identifier.

Examples

Consider this type definition:

```
INTEGER { zero(0), one(1) }
```

The content of each of the following <value> elements is the RXER encoding of a value of the above type:

```
<value>0</value>

<value> zero </value>

<value> 2 <!-- This number doesn't have a name. --> </value>

<value>00167</value>
```

For a non-canonical RXER encoding, if the IntegerType is subject to a VALUES encoding instruction (it necessarily must have a NamedNumberList) and the NamedNumberList defines an identifier for the actual value, then the character data translation of the value is either a number string or the replacement name [RXEREI] for the identifier.

Examples

Consider this type definition:

```
[RXER:VALUES ALL UPPERCASED] INTEGER { zero(0), one(1) }
```

The content of both of the following <value> elements is the RXER encoding of a value of the above type:

```
<value>0</value>

<value> ZERO </value>
```

6.7.7. NULL

The character data translation of a value of the NULL type is an empty character string.

Examples

```
<value/>

<value><!-- Comments don't matter. --></value>

<value></value>
```

The final case is the CRXER encoding.

6.7.8. ObjectDescriptor

A value of the ObjectDescriptor type is translated according to the GraphicString type.

6.7.9. OBJECT IDENTIFIER and RELATIVE-OID

The character data translation of a value of the OBJECT IDENTIFIER or RELATIVE-OID type is a full stop ('.', U+002E) separated list of the object identifier components of the value.

Each object identifier component is translated as a non-negative number string. A non-negative number string is either the digit character '0' (U+0030), or a non-zero decimal digit character (U+0031-U+0039) followed by zero, one, or more of the decimal digit characters '0' to '9' (U+0030-U+0039).

Examples

The content of each of the following <value> elements is the RXER encoding of an OBJECT IDENTIFIER value:

```
<value>2.5.6.0</value>

<value>
    2.5.4.10
</value>

<value> 2.5.4.3 <!-- commonName --> </value>
```

6.7.10. OCTET STRING

The character data translation of a value of the OCTET STRING type is the hexadecimal digit string representation of the octets.

The octets are encoded in order from the first octet to the last octet. Each octet is encoded as a pair of the hexadecimal digit characters '0'-'9', 'A'-'F', and 'a'-'f' (i.e., U+0030-U+0039, U+0041-U+0046, and U+0061-U+0066) where the first digit in the pair corresponds to the four most significant bits of the octet. An odd number of hexadecimal digits is not permitted. The characters 'a'-'f' (i.e., U+0061-U+0066) SHALL NOT be used in the CRXER encoding of an OCTET STRING value.

Aside: The RXER encoding of OCTET STRING values is intended to conform to the lexical representation of the XML Schema [XSD2] hexBinary data type.

Examples

The content of each of the following <value> elements is the RXER encoding of an OCTET STRING value:

```
<value>27F69A0300</value>

<value>
    efA03bFF
</value>
```

6.7.11. QName

The character data translation of a value of the QName type (Section 4.5) is a qualified name conforming to the QName production of Namespaces in XML 1.0 [XMLNS10].

The local part (i.e., LocalPart) of the qualified name SHALL be the value of the local-name component of the QName value.

If the namespace-name component of the QName value is absent, then the namespace prefix (i.e., Prefix) of the qualified name SHALL be absent; otherwise, the namespace prefix is determined as specified in Section 6.7.11.1 using the value of the namespace-name component of the QName value as the namespace name.

6.7.11.1. Namespace Prefixes for Qualified Names

This section describes how the namespace prefix of a qualified name is determined given the namespace name to which the namespace prefix must map.

For a CRXER encoding, the namespace prefix of the qualified name is any unused non-canonical namespace prefix unless the [in-scope namespaces] property of the enclosing element item contains a namespace item with a [namespace name] that matches the namespace name. In that case, the [prefix] of that namespace item SHALL be used as the namespace prefix of the qualified name.

Aside: If the qualified name appears in the [normalized value] of an attribute item, then the enclosing element item is the [owner element] for that attribute item.

For a non-canonical RXER encoding, the namespace prefix of the qualified name is any unused namespace prefix unless the [in-scope namespaces] property of the enclosing element item contains a namespace item with the same [namespace name] as the element item. In that case, the [prefix] of that namespace item MAY be used as the

namespace prefix of the qualified name. Note that the [prefix] of a namespace item for the default namespace has no value.

If the namespace prefix of the qualified name is an unused namespace prefix, then a namespace declaration attribute item associating the namespace prefix with the namespace name MUST be added to the [namespace attributes] of the enclosing element item, and a corresponding namespace item MUST be added to the [in-scope namespaces] of the enclosing element item.

6.7.12. REAL

The character data translation of a value of the REAL type is the character string "0" if the value is positive zero, the character string "-0" if the value is negative zero, the character string "INF" if the value is positive infinity, the character string "-INF" if the value is negative infinity, the character string "NaN" if the value is not a number, or a real number otherwise.

A real number is the mantissa followed by either the character 'E' (U+0045) or 'e' (U+0065) and the exponent. The character 'e' SHALL NOT be used for a CRXER encoding. If the exponent is zero, then the 'E' or 'e' and exponent MAY be omitted for a non-canonical RXER encoding.

The mantissa is a decimal number with an optional leading sign, either '+' (U+002B) or '-' (U+002D). A decimal number is a sequence of one or more of the decimal digit characters '0' to '9' (U+0030-U+0039) optionally partitioned by a single full stop character ('.', U+002E) representing the decimal point. Multiple leading zero digits are permitted for a non-canonical RXER encoding.

The exponent is encoded as a number string (see Section 6.7.6).

> Aside: The RXER encoding of REAL values is intended to be compatible with the lexical representation of the XML Schema [XSD2] double data type, but allows real values outside the set permitted by double.

For a CRXER encoding:

(1) The real number MUST be normalized so that the mantissa has a single non-zero digit immediately to the left of the decimal point.

(2) Leading zero digits SHALL NOT be used.

(3) A leading plus sign SHALL NOT be used in the mantissa or the exponent.

(4) The fractional part of the mantissa (i.e., that part following the decimal point) MUST have at least one digit (which may be '0') and MUST NOT have any trailing zeroes after the first digit.

(5) The exponent SHALL be present and SHALL be a canonical number string (see Section 6.7.6).

Examples

The content of each of the following <value> elements is the RXER encoding of a REAL value:

```
<value>3.14159<!-- pi --></value>

<value> 1.0e6 </value>

<value> INF </value>

<value>
    -01e-06
</value>
```

6.7.13. UTCTime

The character data translation of a value of the UTCTime type is a date, the letter 'T' (U+0054), a time of day, and a time zone.

The date is two decimal digits representing the year (no century), a hyphen ('-', U+002D), two decimal digits representing the month, a hyphen ('-', U+002D), and two decimal digits representing the day.

The time of day is two decimal digits representing the hour, followed by a colon (':', U+003A), two decimal digits representing the minutes, a colon (':', U+003A), and two decimal digits representing the seconds.

Note that the hours value "24" is disallowed [X.680].

The seconds are encoded as "00" if the UTCTime value omits seconds.

The time zone is either the letter 'Z' (U+005A) to indicate Coordinated Universal Time, a plus sign ('+', U+002B) followed by a time zone differential, or a minus sign ('-', U+002D) followed by a time zone differential.

A time zone differential indicates the difference between local time (the time specified by the preceding date and time of day) and Coordinated Universal Time. Coordinated Universal Time can be calculated from the local time by subtracting the differential.

For a CRXER encoding, a UTCTime value with a time zone differential SHALL be encoded as the equivalent Coordinated Universal Time, i.e., the time zone will be "Z".

A time zone differential is encoded as two decimal digits representing hours, a colon (':', U+003A), and two decimal digits representing minutes.

6.7.14. CHOICE as UNION

The chosen alternative of a value of a UNION type corresponds to some NamedType in the UNION type definition (a ChoiceType).

The character data translation of a value of a UNION type is the character data translation of the value of the type of the chosen alternative, i.e., without any kind of encapsulation.

Leading and trailing white space characters are not permitted to be added to the character data translation of a value of a UNION type (see Section 6.7); however, this does not preclude such white space being added to the character data translation of the value of the chosen alternative.

The character data translation of a value of a UNION type is necessarily destined for the [children] of an enclosing element item.

> Aside: This is because the ATTRIBUTE encoding instruction cannot be applied to a NamedType with a type that is a UNION type.

The chosen alternative can be identified by a member attribute item, i.e., an attribute item with the [local name] "member" and [namespace name] "urn:ietf:params:xml:ns:asnx", added to the [attributes] of the enclosing element item. The [prefix] of this attribute item is determined as specified in Section 6.2.3.1. The [normalized value] of the attribute item is a qualified name for the expanded name of the NamedType (see [RXEREI]) corresponding to the chosen alternative.

> Aside: It is not possible to associate a namespace name with a NamedType in a UNION type using the current specification for RXER encoding instructions. Consequently, the [normalized value] of the member attribute item will always contain a qualified name without a namespace prefix.

For a CRXER encoding, the member attribute item MUST be used, and the [normalized value] of the attribute item MUST be the CRXER translation of the QName value equal to the expanded name.

In the absence of a member attribute item, an RXER decoder MUST determine the chosen alternative by considering the alternatives of the choice in the order prescribed below and accepting the first alternative for which the encoding is valid.

If the UNION encoding instruction has a PrecedenceList, then the alternatives of the ChoiceType referenced by the PrecedenceList are considered in the order identified by that PrecedenceList, then the remaining alternatives are considered in the order of their definition in the ChoiceType. If the UNION encoding instruction does not have a PrecedenceList, then all the alternatives of the ChoiceType are considered in the order of their definition in the ChoiceType.

A non-canonical RXER encoder MUST use the member attribute item if an RXER decoder would determine the chosen alternative to be something other than the actual chosen alternative of the CHOICE value being translated; otherwise, the member attribute item MAY be used.

Examples

Consider this type definition:

```
[RXER:UNION PRECEDENCE serialNumber] CHOICE {
    name          [0] IA5String,
    serialNumber  [1] INTEGER
}
```

In the absence of a member attribute, an RXER decoder would first consider whether the received encoding was a valid serialNumber (an INTEGER) before considering whether it was a valid name (an IA5String).

The content and attributes of each of the following <value> elements are the RXER encoding of a value of the above type:

```
<value>Bob</value>

<value xmlns:asnx="urn:ietf:params:xml:ns:asnx"
       asnx:member="name">Alice</value>

<value>
 <!-- Don't have a name for this one! --> 344
</value>
```

```
<value xmlns:asnx="urn:ietf:params:xml:ns:asnx"
       asnx:member="name"><!-- A strange name. -->100</value>
```

The member attribute is required in the final case to prevent the value being interpreted as a serialNumber.

If the UNION (i.e., CHOICE) type is extensible [X.680], then an application MUST accept and be prepared to re-encode (using the same encoding rules) any unknown extension in received encoded values of the type. An unknown extension in a value of a UNION type (an unknown alternative) takes the form of an unknown name in the [normalized value] of the member attribute and/or character data in the [children] of the enclosing element item that do not conform to any of the known alternatives.

To enable re-encoding of an unknown alternative, it is necessary to retain the [normalized value] of the member attribute, if present, and the [children] property of the enclosing element item.

The character data for an unknown alternative may contain qualified names that depend on the [in-scope namespaces] of the enclosing element item for their interpretation. Therefore, semantically faithful re-encoding of an unknown alternative may require reproduction of at least some part of the [in-scope namespaces] of the enclosing element item. The problem is deciding which of the namespace items are actually needed. In the absence of type information, it is not possible to discern whether anything that syntactically resembles a qualified name in the character data of the enclosing element item actually is a qualified name. The simplest approach is to retain all the namespace items from the [in-scope namespaces] of the enclosing element item and output them as namespace declaration attribute items in the [namespace attributes] of the enclosing element item when re-encoding the unknown alternative. At best, an application can omit the namespace items that do not define the namespace prefix of any potential qualified name.

An application MUST retain the namespace items in the [in-scope namespaces] of the enclosing element item that define the namespace prefixes of all the potential qualified names in the [children] of the enclosing element item. Other namespace items in the [in-scope namespaces] of the enclosing element item MAY be retained. The effect of these retained namespace items on the [namespace attributes] and [in-scope namespaces] of the enclosing element item when re-encoding is considered in Section 6.2.2.1.

Aside: The context attribute (Section 6.8.8) is not added to the [attributes] of the enclosing element item when re-encoding an unknown alternative since the type of a NamedType in a UNION type cannot be the Markup type.

6.7.15. SEQUENCE OF as LIST

The character data translation of a value of a LIST type (a SEQUENCE OF NamedType) is the concatenation of the character data translations of the component values, i.e., the abstract values of the type of the NamedType, each separated from the next by at least one white space character. For a CRXER encoding, separating white space MUST be exactly one space character (U+0020).

Example

Consider this type definition:

```
[LIST] SEQUENCE OF timeStamp GeneralizedTime
```

The content of the following <value> element is the RXER encoding of a value of the above type:

```
<value>
    2004-06-15T12:14:56Z
    2004-06-15T12:18:13Z
    2004-06-15T01:00:25Z
</value>
```

6.8. Combining Types

The encoding of a value of an ASN.1 combining type (except a UNION or LIST type) typically has element content.

The Infoset translation of a value of a specific ASN.1 combining type (excluding a UNION or LIST type) contains zero or more attribute items to be added to the [attributes] of the enclosing element item and zero or more element items to be added to the [children] of the enclosing element item. These translations are described in Sections 6.8.1 to 6.8.7.

For a non-canonical RXER encoding, white space character items MAY be added to the [children] of the enclosing element item (before or after any other items).

For a CRXER encoding, a character item with the [character code] U+000A (a line feed) MUST be inserted immediately before each element item in the [children] of the enclosing element item. No other white space character items are permitted to be added to the [children] of the enclosing element item.

Aside: Without the single line feed character before each child element, a typical CRXER encoding would be a single, very long line.

6.8.1. CHARACTER STRING

A value of the unrestricted CHARACTER STRING type is translated according to the corresponding SEQUENCE type defined in Clause 40.5 of X.680 [X.680].

6.8.2. CHOICE

The chosen alternative of a value of a CHOICE type corresponds to, and is a value of (see Section 6), some NamedType in the CHOICE type definition.

The translation of a value of a CHOICE type other than the Markup type or a UNION type (see Section 6.7.14) is the translation of the value of the NamedType corresponding to the actual chosen alternative.

Examples

Consider this type definition:

```
CHOICE {
    name          [0] IA5String,
    serialNumber  [1] INTEGER
}
```

The content of each of the following <value> elements is the RXER encoding of a value of the above type:

```
<value><name>Bob</name></value>

<value>
 <name>Alice</name>
</value>
```

```
<value>
 <!-- Don't have a name for this one! -->
 <serialNumber>
  344
 </serialNumber>
</value>

<value>
 <!-- A strange name. -->
 <name>100</name>
</value>
```

If the CHOICE type is extensible [X.680], then an application MUST accept, and be prepared to re-encode (in RXER), any attribute item or child element item with a name that is not recognized (see Section 6.8.8).

6.8.3. EMBEDDED PDV

A value of the EMBEDDED PDV type is translated according to the corresponding SEQUENCE type defined in Clause 33.5 of X.680 [X.680].

6.8.4. EXTERNAL

A value of the EXTERNAL type is translated according to the corresponding SEQUENCE type defined in Clause 8.18.1 of X.690 [X.690].

6.8.5. INSTANCE OF

A value of the INSTANCE OF type is translated according to the corresponding SEQUENCE type defined in Annex C of X.681 [X.681].

6.8.6. SEQUENCE and SET

Each component value of a value of a SEQUENCE or SET type corresponds to, and is a value of (see Section 6), some NamedType in the SEQUENCE or SET type definition.

A value of a SEQUENCE or SET type, other than the QName type (Section 4.5), is translated by translating in turn each component value actually present in the SEQUENCE or SET value and adding the resulting attribute items and/or element items to the [attributes] and/or [children] of the enclosing element item. Attribute items may be added to the [attributes] of the enclosing element item in any order. Element items resulting from the translation of component

values MUST be appended to the [children] of the enclosing element item in the order of the component values' corresponding NamedType definitions in the SEQUENCE or SET type definition.

Aside: In the case of the SET type, this is a deliberate departure from BER [X.690], where the components of a SET can be encoded in any order.

If a DEFAULT value is defined for a NamedType and the value of the NamedType is the same as the DEFAULT value, then the translation of the value of the NamedType SHALL be omitted for a CRXER encoding and MAY be omitted for a non-canonical RXER encoding.

Examples

Consider this type definition:

```
SEQUENCE {
    name        [0] IA5String OPTIONAL,
    partNumber  [1] INTEGER,
    quantity    [2] INTEGER DEFAULT 0
}
```

The content of each of the following <value> elements is the RXER encoding of a value of the above type:

```
<value>
 <partNumber>23</partNumber>
 <!-- The quantity defaults to zero. -->
</value>

<value>
 <name>chisel</name>
 <partNumber> 37 </partNumber>
 <quantity> 0 </quantity>
</value>

<value>
 <!-- The name component is optional. -->
 <partNumber>1543</partNumber>
 <quantity>29</quantity>
</value>
```

If the SEQUENCE or SET type is extensible [X.680], then an application MUST accept, and be prepared to re-encode (in RXER), any attribute item or child element item with a name that is not recognized (see Section 6.8.8).

6.8.7. SEQUENCE OF and SET OF

Each component value of a value of a type that is a SET OF NamedType or a SEQUENCE OF NamedType corresponds to, and is a value of (see Section 6), the NamedType in the type definition.

A value of a type that is a SET OF NamedType, or a SEQUENCE OF NamedType other than a LIST type (see Section 6.7.15), is translated by adding the translation of each value of the NamedType to the [children] of the enclosing element item.

> Aside: An ATTRIBUTE encoding instruction cannot appear in the component type for a SEQUENCE OF or SET OF type, so there are no attribute items to add to the [attributes] of the enclosing element item.

If the type is a SEQUENCE OF NamedType, then the values of the NamedType are translated in the order in which they appear in the value of the type.

For a non-canonical RXER encoding, if the type is a SET OF NamedType, then the values of the NamedType may be translated in any order.

For a CRXER encoding, if the type is a SET OF NamedType, then the values of the NamedType MUST be translated in ascending order where the order is determined by comparing the octets of their CRXER encodings (which will be UTF-8 encoded character strings; see Section 6.12.2). A shorter encoding is ordered before a longer encoding that is identical up to the length of the shorter encoding.

Examples

Consider this type definition:

```
SEQUENCE OF timeStamp GeneralizedTime
```

The content of the following <value> element is the RXER encoding of a value of the above type:

```
<value>
    <timeStamp>2004-06-15T12:14:56Z</timeStamp>
    <timeStamp>2004-06-15T12:18:13Z</timeStamp>
    <timeStamp>
        2004-06-15T01:00:25Z
    </timeStamp>
</value>
```

Consider this type definition (also see Section 6.6):

```
SEQUENCE OF INTEGER
```

The content of the following <value> element is the RXER encoding of a value of the above type:

```
<value>
 <item>12</item>
 <item>
  9
 </item>
 <item> 7 <!-- A prime number. --></item>
</value>
```

6.8.8. Extensible Combining Types

An application must accept and be prepared to re-encode (using the same encoding rules) any unknown extension appearing in the encoding of a value of an extensible CHOICE, SEQUENCE, or SET type. An unknown extension in a value of an extensible combining type (except UNION types) takes the form of unknown element and/or attribute items. Section 6.8.8.1 describes the processing of unknown element items and Section 6.8.8.2 describes the processing of unknown attribute items.

An application cannot produce a canonical encoding if an abstract value contains unknown extensions. However, the method for re-encoding unknown extensions does not prevent a receiving application with knowledge of the extension from producing the correct canonical encoding.

6.8.8.1. Unknown Elements in Extensions

To enable re-encoding of an unknown element item it is necessary to retain the [prefix], [local name], [attributes], [namespace attributes], and [children] properties of the element item.

Definition (inherited namespace item): An inherited namespace item is a namespace item in the [in-scope namespaces] of an element item for which there is no corresponding namespace declaration attribute item in the [namespace attributes] of the element item.

The content and attributes of an unknown element item may contain qualified names whose interpretation depends on inherited namespace items. Semantically faithful re-encoding of the unknown item may require reproduction of at least some of the inherited namespace

items. The problem is deciding which of the inherited namespace items are actually needed. Qualified names as the names of elements and attributes are easily recognized, but in the absence of type information it is not possible to discern whether anything that syntactically resembles a qualified name in the value of an attribute or the character data of an element actually is a qualified name.

The simplest approach is to retain all the inherited namespace items and output corresponding namespace declaration attribute items in the [namespace attributes] of the unknown element item when re-encoding the element item. At best, an application can omit the inherited namespace items that do not define the namespace prefix of any definite or potential qualified name, though this requires examining the content and attributes of the unknown extension.

Regardless of how clever an implementation tries to be, adding any namespace declaration attribute items to an unknown element item is harmful to canonicalization if the ASN.1 type for the element item turns out to be the Markup type. To counter this problem, a special attribute is used to identify the namespace declaration attribute items added to an unknown element item so that they can be removed later, if it proves necessary.

If the outermost element item in an unknown extension does not have an attribute item with the [local name] "context" and [namespace name] "urn:ietf:params:xml:ns:asnx" in its [attributes], then namespace declaration attribute items corresponding to the inherited namespace items that define the namespace prefixes of all the definite and potential qualified names in the content and attributes of the element item MUST be added to the retained [namespace attributes]. Other inherited namespace items MAY be added to the retained [namespace attributes].

If there are one or more of these added namespace declaration attribute items, then an attribute item with the [local name] "context" and [namespace name] "urn:ietf:params:xml:ns:asnx" MUST be added to the retained [attributes].

The [prefix] of the context attribute item is any namespace prefix that does not match the [local name] of any namespace declaration attribute item in the [namespace attributes] unless the [namespace attributes] property contains a namespace declaration attribute item with a non-empty [prefix] and a [normalized value] of "urn:ietf:params:xml:ns:asnx". In that case, the [local name] of that namespace declaration attribute item MAY be used as the [prefix] of the context attribute item.

If the [prefix] of the context attribute item does not match the [local name] of any namespace declaration attribute item, then an attribute item with the [prefix] "xmlns", [namespace name] "urn:ietf:params:xml:ns:asnx", and [local name] equal to the [prefix] of the context attribute item MUST be added to the retained [namespace attributes] of the element item.

The [normalized value] of the context attribute is the white-space-separated unordered list of the [local names] of the added namespace declaration attribute items (i.e., a list of the namespace prefixes), including any namespace declaration attribute item added to define the [prefix] of the context attribute. Note that the [local name] for a namespace declaration attribute item declaring the default namespace is "xmlns".

> Aside: A receiver that knows about the extension will use the context attribute to strip out the added namespace declaration attributes if the type of the associated NamedType is the Markup type (Section 6.10), and will discard the context attribute otherwise. A receiver that does not know about the extension will re-encode the extension as is.

Adding the required namespace declaration attribute items to an element item effectively makes the element item self-contained. A received encoding has an encoding error if it contains an element item that is not self-contained but has a context attribute item in its [attributes].

An RXER encoder MUST NOT add the context attribute item to an element item corresponding to a NamedType that is known to it.

An RXER decoder MUST accept the context attribute item on an element item corresponding to a NamedType that does not appear to be an extension.

> Aside: It is not uncommon for extension markers to be neglected in specifications traditionally using only BER, since extension markers do not alter BER encodings. Consequently, it is not immediately obvious in later versions of the specification which instances of NamedType belong to extensions of the original base specification.

Example

> Suppose there are three applications, A, B, and C. Suppose that Application A uses the first edition of an ASN.1 specification containing the following type definition:

```
MyType ::= SEQUENCE {
        field1  INTEGER,  -- present in first edition
        ...
}
```

Suppose that Application B uses the second edition of the ASN.1 specification:

```
MyType ::= SEQUENCE {
        field1  INTEGER,  -- present in first edition
        ...,
        field2  QName     -- added in second edition
}
```

Suppose that Application C uses the third edition of the ASN.1 specification:

```
MyType ::= SEQUENCE {
        field1  INTEGER,  -- present in first edition
        ...,
        field2  QName,    -- added in second edition
        field3  Markup    -- added in third edition
}
```

Application C produces the following RXER encoding and sends it to Application B:

```
<value xmlns:p2="http://example.com/ns2">
 <field1> 100 </field1>
 <field2> p2:foobar </field2>
 <field3 xmlns:p1="http://example.com/ns1"> p1:foobar </field3>
</value>
```

Application B doesn’t know about <field3>, so it adds the asnx:context attribute to <field3> when it re-encodes the abstract value to send to Application A:

```
<value xmlns:p1="http://example.com/ns2">
 <!-- Application B knows the white space in field1 and
      field2 is optional and discards it. -->
 <field1>100</field1>
 <field2>p1:foobar</field2>
 <!-- Application B doesn't know about field3
      so it leaves the character data alone. -->
 <field3 asnx:context="asnx p2"
         xmlns:asnx="urn:ietf:params:xml:ns:asnx"
         xmlns:p1="http://example.com/ns1"
         xmlns:p2="http://example.com/ns2"> p1:foobar </field3>
</value>
```

Application A doesn't know about <field2> and <field3>, so it adds the asnx:context attribute to <field2> and leaves <field3> alone when it re-encodes the abstract value:

```
<value>
 <!-- Application A knows about field1 and chooses
      to add some white space. -->
 <field1> 100 </field1>
 <!-- Application A doesn't know about field2 or field3
      so it leaves the character data alone. -->
 <field2 asnx:context="asnx p1"
         xmlns:asnx="urn:ietf:params:xml:ns:asnx"
         xmlns:p1="http://example.com/ns2">p1:foobar</field2>
 <field3 asnx:context="asnx p2"
         xmlns:asnx="urn:ietf:params:xml:ns:asnx"
         xmlns:p1="http://example.com/ns1"
         xmlns:p2="http://example.com/ns2"> p1:foobar </field3>
</value>
```

If Application C receives this final encoding, it has sufficient information to discard the asnx:context, xmlns:asnx, and xmlns:p2 attributes from the received Markup value of <field3> to recover the original value. Application C knows about <field2>, so it uses the namespace declaration for p1 when decoding the QName value and ignores the other declarations.

6.8.8.2. Unknown Attributes in Extensions

To enable re-encoding of an unknown attribute item it is necessary to retain at least the [local name], [namespace name], and [normalized value] properties of the attribute item.

The [normalized value] of an unknown attribute item may contain qualified names whose interpretation depends on the [in-scope namespaces] of the [owner element]. Semantically faithful

re-encoding of the unknown attribute item may require reproduction of at least some part of the [in-scope namespaces]. In the absence of type information, it is not possible to discern whether anything that syntactically resembles a qualified name in the [normalized value] of an unknown attribute item actually is a qualified name.

The simplest approach is to retain all the namespace items of the [in-scope namespaces] and output corresponding namespace declaration attribute items in the [namespace attributes] of the [owner element] when re-encoding the extension. At best, an application can omit the namespace items that do not define the namespace prefix of any potential qualified name in the [normalized value].

An application MUST retain the namespace items in the [in-scope namespaces] of the [owner element] that define the namespace prefixes of all the potential qualified names in the [normalized value] of the unknown attribute item. Other namespace items in the [in-scope namespaces] of the [owner element] MAY be retained.

> Aside: If the enclosing element item has more than one unknown attribute item, then it is sufficient to save the union of the retained namespace items with the element item, rather than saving the retained namespace items with each unknown attribute item.

When the unknown attribute item is re-encoded, the retained namespace items affect the [namespace attributes] and [in-scope namespaces] of the enclosing element item as specified in Section 6.2.2.1, and the [prefix] of the attribute item is determined as specified in Section 6.2.3.1.

> Aside: The context attribute is not added to the [attributes] of the [owner element] when re-encoding an unknown attribute item because the type of a NamedType subject to an ATTRIBUTE or ATTRIBUTE-REF encoding instruction cannot be the Markup type.

6.9. Open Type

A value of an open type denoted by an ObjectClassFieldType [X.681] is translated according to the specific Type of the value.

If the specific Type of the value is directly or indirectly the Markup type, then the enclosing element item MUST be self-contained.

For a non-canonical RXER encoding, if the translation of the value does not generate an attribute item with the [local name] "type" and the [namespace name] "http://www.w3.org/2001/XMLSchema-instance" (i.e., xsi:type) and the specific Type of the value is a

namespace-qualified reference (Section 5), then an attribute item with the [local name] "type" and the [namespace name] "http://www.w3.org/2001/XMLSchema-instance" (i.e., xsi:type) MAY be added to the [attributes] of the enclosing element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced type, with the namespace prefix determined as specified in Section 6.7.11.1.

Aside: The xsi:type attribute is added by RXER encoders for the benefit of XML Schema validators. This attribute tells an XML Schema validator which type definition in a compatible XML Schema translation of the ASN.1 specification it should use for validating the content and attributes of the enclosing element. For an RXER decoder, the actual type in an open type value is generally determined by an associated component relation constraint [X.682], so the xsi:type attribute can be ignored.

Example

The content and attributes of the following <value> element are the RXER encoding of an open type value containing a BOOLEAN value:

```
<value xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:asnx="urn:ietf:params:xml:ns:asnx"
       xsi:type="asnx:BOOLEAN"> true </value>
```

If the ObjectClassFieldType denoting an open type is not constrained by a TableConstraint, or is constrained by a TableConstraint where the constraining object set is extensible, then an application MUST accept and be prepared to re-encode (using the same encoding rules) any value of the open type where the specific Type of the value is unknown. In such cases, the enclosing element item is treated like an unknown element item in the value of an extensible combining ASN.1 type (see Section 6.8.8.1).

6.10. Markup

Conceptually, a value of the Markup type holds the [prefix], [attributes], [namespace attributes], and [children] of an element item. The Infoset translation of a value of the Markup type initially simply sets the [prefix], [attributes], [namespace attributes], and [children] of the enclosing element item to the corresponding properties represented by the Markup value.

Recall that the enclosing element item for the translation of a Markup value is required to be self-contained (Section 4.1.1).

If the enclosing element item is not the [document element] of the document item, and the [in-scope namespaces] property of the enclosing element item's [parent] contains a namespace item for the default namespace, and the [namespace attributes] property represented by the Markup value does not contain a namespace item declaring or undeclaring the default namespace, then a namespace declaration attribute item that undeclares the default namespace SHALL be added to the enclosing element item's [namespace attributes].

It is not necessary to populate the [in-scope namespaces] of the enclosing element item for encoding purposes (though it may be warranted for other purposes).

An element item nested in the [children] is potentially the Infoset translation of a value of a top-level NamedType (as allowed by Section 6.4), and the entire Markup value can represent the content and attributes of an element item that is the translation of a value of a top-level NamedType.

> Aside: The latter case arises when an ELEMENT-REF encoding instruction references a top-level NamedType.

The content and attributes of an element item nested in the [children] of a Markup value are potentially the Infoset translation of an abstract value of an ASN.1 type (as allowed by Section 6.4), and the entire Markup value can represent the translation of a single abstract value.

> Aside: The latter case arises when a TYPE-REF encoding instruction references an ASN.1 type.

For a non-canonical RXER encoding, any element item, at any level of nesting (including the enclosing element item itself), that corresponds to the value of a top-level NamedType MAY be replaced with any valid translation of that value.

For a non-canonical RXER encoding, any element item, at any level of nesting (including the enclosing element item itself), with content and attributes that correspond to an abstract value of an ASN.1 type MAY have that content and those attributes replaced with any valid translation of that abstract value. If the content and attributes are replaced, then the [prefix], [in-scope namespaces], and [namespace attributes] of the element item are constructed as specified in Sections 6.2.2.1 and 6.2.2.2. The enclosing element item for the Markup value is still required to be self-contained.

Aside: Insofar as a Markup value represents ASN.1 abstract values, it is sufficient for the RXER encoding of the Markup value to preserve the abstract values rather than preserve the exact Infoset representation.

For a CRXER encoding, any element item, at any level of nesting (including the enclosing element item itself), that corresponds to a value of a top-level NamedType MUST be replaced with the CRXER translation of that value.

For a CRXER encoding, any element item, at any level of nesting (including the enclosing element item itself), with content and attributes that correspond to an abstract value of an ASN.1 type MUST have that content and those attributes replaced with the CRXER translation of that abstract value. The [prefix], [in-scope namespaces], and [namespace attributes] of the element item are constructed as specified in Sections 6.2.2.1 and 6.2.2.2.

If the [attributes] property of the enclosing element item from a received RXER encoding contains an attribute item with the [local name] "context" and [namespace name] "urn:ietf:params:xml:ns:asnx" (i.e., asnx:context), then this attribute item MUST be omitted from the [attributes] represented by the Markup value, and each namespace declaration attribute item with a [local name] matching an NCName in the [normalized value] of the attribute item MUST be omitted from the [namespace attributes] represented by the Markup value.

6.11. Namespace Prefixes for CRXER

The final step in translating the value of a top-level NamedType for a CRXER encoding, or an abstract value for a Standalone CRXER Encoding, is the replacement of the arbitrarily chosen namespace prefixes with algorithmically determined canonical namespace prefixes. This procedure for prefix replacement applies to each element item where the [namespace attributes] have been constructed according to Section 6.2.2.1. This includes any element item corresponding to a value of a top-level NamedType, or with content and attributes that correspond to an abstract value of an ASN.1 type, that is nested in a value of the Markup type.

For each element item where prefix replacement applies, the following sequence of steps is repeated until there are no more eligible attribute items to select in step (1):

(1) Select the attribute item with the least [normalized value] from amongst the attribute items of the [namespace attributes] that have a [local name] that is not a canonical namespace prefix (i.e., select from the namespace declaration attribute items that have not already been processed). A [normalized value] is less than another [normalized value] if the former appears before the latter in an ordering of the values determined by comparing the ISO 10646 code points [UCS] of their characters, from first to last. A shorter string of characters is ordered before a longer string of characters that is identical up to the length of the shorter string.

Aside: Note that when a namespace declaration (other than for the default namespace) is represented as an attribute item in the [namespace attributes], the attribute's [prefix] is "xmlns", its [local name] is the namespace prefix, and its [normalized value] is the namespace name.

(2) A canonical namespace prefix is unused if it is not currently the [prefix] of any namespace item in the [in-scope namespaces] of the element item. Replace the [local name] of the selected attribute item with the unused canonical namespace prefix that has the non-negative number string with the least integer value (e.g., n2 is less than n10).

(3) The selected attribute item has a corresponding namespace item in the [in-scope namespaces] of the element. Replace the [prefix] of this corresponding namespace item with the canonical namespace prefix determined in step (2).

(4) The element item and its [attributes] property, and descendent element items and their [attributes] properties, may depend on the selected attribute item to determine the binding between their [prefix] and [namespace name]. Replace the [prefix] of any such dependent element items and attribute items with the canonical namespace prefix determined in step (2).

Note that a namespace prefix can be redeclared (reused). Replacement of the prefix does not apply to an element item wherein the prefix is redeclared, or to the descendants of such an element item.

(5) The character data translations for values of the QName ASN.1 type may depend on the selected attribute item to determine the binding between their namespace prefix and namespace name. Replace the namespace prefix of any such dependent character data translation with the canonical namespace prefix determined in step (2).

Note that a character data translation can appear in the [normalized value] of an attribute item, or as a sequence of character items in the [children] of an element item.

6.12. Serialization

The final RXER encoding is produced by serializing the Infoset translation as an XML document. An implementation MUST serialize the Infoset translation as an XML document in such a way that the Infoset of the resulting XML document matches the Infoset translation, after ignoring the following properties:

(1) all properties of the document item except the [document element],

(2) the [base URI] of any item,

(3) the [element content whitespace] of character items,

(4) the [notation] of processing instruction items,

(5) the [in-scope namespaces] of element items.

Aside: The [in-scope namespaces] of a parent element item are only selectively inherited by its child element items in the Infoset translations of ASN.1 values. This means that the Infoset reconstructed by parsing the XML document serialization of the original Infoset will generally have more namespace items in its [in-scope namespaces], but these extra namespace items will not be significant.

Aside: A consequence of case (1) is that comments and PIs before and after the document element are permitted.

In general, there is more than one possible serialization for any given Infoset translation. Section 6.12.1 highlights some important considerations in producing a correct serialization and discusses some of the serialization options.

Section 6.12.2 applies to CRXER encodings and limits the serialization options so that each distinct Infoset has only one possible serialization.

6.12.1. Non-Canonical Serialization

This section discusses aspects of Infoset serialization for non-canonical RXER encodings, but is not an exhaustive list of the options for non-canonical serialization.

If one or more character items have a [character code] in the range U+0001 to U+0008, U+000B to U+000C, or U+000E to U+001F, or one or more characters in any attribute's [normalized value] are in the range U+0001 to U+0008, U+000B to U+000C, or U+000E to U+001F, then the Infoset translation MUST be serialized as an XML version 1.1 document; otherwise, the Infoset translation is serialized as either an XML version 1.0 or version 1.1 document.

A non-canonical RXER encoding may use any of the allowed character encoding schemes for XML. RXER encoders and decoders MUST support the UTF-8 character encoding.

An element item may be serialized as an empty-element tag if it has no items in its [children].

Attributes of an element can appear in any order since the [attributes] and [namespace attributes] of an element item are unordered.

Ampersand ('&', U+0026) and open angle bracket ('<', U+003C) characters in the [normalized value] of an attribute item must be escaped appropriately [XML10][XML11] (with a character reference or a predefined entity reference). Double quote (U+0022) and single quote (U+0027) characters in an attribute item's [normalized value] may also need to be escaped. Character items with the [character code] U+0026 (ampersand, '&') or U+003C (open angle bracket, '<') must be escaped appropriately (with a character reference, a predefined entity reference or a CDATA section).

Line break normalization by XML processors allows some freedom in how a character item for a line feed character (U+000A) is serialized:

(1) If XML version 1.0 is selected, then a character item with the [character code] U+000A (line feed) is serialized as either a line feed character (U+000A), a carriage return character (U+000D) followed by a line feed character (U+000A), or just a carriage return character (U+000D) provided the next item is not a character item that is serialized as a line feed character (U+000A).

(2) If XML version 1.1 is selected, then a character item with the [character code] U+000A (line feed) is serialized as either a line feed character (U+000A), a next line character (U+0085), a line separator character (U+2028), a carriage return character (U+000D) followed by a line feed character (U+000A), a carriage return character (U+000D) followed by a next line character

(U+0085), or just a carriage return character (U+000D) provided the next item is not a character item that is serialized as a line feed (U+000A) or next line (U+0085) character.

Aside: All these sequences will be normalized to a line feed character (U+000A) during decoding.

A character item with the [character code] U+000D (carriage return), U+0085 (next line), or U+2028 (line separator) must be serialized as a character reference to protect the character from line break normalization during decoding.

The attribute value normalization performed by XML processors allows some freedom in how a space character (U+0020) is serialized:

(1) If XML version 1.0 is selected, then a space character (U+0020) in an attribute item's [normalized value] is serialized as either a space character (U+0020), a tab character (U+0009), a carriage return character (U+000D), a line feed character (U+000A), a carriage return character (U+000D) followed by a line feed character (U+000A), or just a carriage return character (U+000D) provided the next character in the [normalized value] is not serialized as a line feed character (U+000A).

(2) If XML version 1.1 is selected, then a space character (U+0020) in an attribute item's [normalized value] is serialized as either a space character (U+0020), a tab character (U+0009), a carriage return character (U+000D), a line feed character (U+000A), a next line character (U+0085), a line separator character (U+2028), a carriage return character (U+000D) followed by a line feed character (U+000A), a carriage return character (U+000D) followed by a next line character (U+0085), or just a carriage return character (U+000D) provided the next character in the [normalized value] is not serialized as a line feed (U+000A) or next line (U+0085) character.

Aside: All these sequences will be normalized to a space character (U+0020) during decoding, through a combination of line break normalization and attribute value normalization.

Each tab (U+0009), line feed (U+000A), or carriage return (U+000D) character in an attribute item's [normalized value] must be serialized as a character reference to protect the character from attribute value normalization during decoding. In addition, if XML version 1.1 is selected, then each next line (U+0085) or line separator (U+2028) character must be serialized as a character reference.

Parsed entity references may be used (unless the environment in which the RXER encoding is used disallows entity references). If entity references to other than the predefined entities are used, then the XML document containing the RXER encoding must necessarily contain a document type declaration, and the internal or external subset of the document type definition must contain entity declarations for those entities.

6.12.2. Canonical Serialization

This section discusses Infoset serialization for CRXER encodings. The serialization of an Infoset for a CRXER encoding is restricted so that each distinct Infoset has only one possible serialization as an XML document.

> Aside: These restrictions have been chosen so as to be consistent with Canonical XML [CXML], where possible.

The document SHALL be encoded in UTF-8 without a leading Byte Order Mark [UCS].

The XMLDecl of the document SHALL be <?xml version="1.1"?>.

A document type declaration (doctypedecl) SHALL NOT be used.

> Aside: This has the effect of excluding entity references, except those for the predefined entities (e.g., &).

A single line feed character (U+000A) MUST be inserted immediately before the document element.

No other white space characters are permitted before or after the document element.

There SHALL NOT be any PIs or comments before or after the document element.

An element item MUST NOT be serialized as an empty-element tag.

> Aside: If an element item has no items in its [children], then it is serialized as a start-tag followed by an end-tag.

There SHALL NOT be any white space characters immediately before the closing '>' of an element's start-tag and end-tag. The white space preceding each attribute SHALL be exactly one space character (U+0020). There SHALL NOT be any white space characters immediately before or after the equals sign (U+003D) in an attribute.

The delimiter for attribute values SHALL be the double quote character (U+0022).

Namespace declaration attributes MUST appear before any other attributes of an element. A namespace declaration for the default namespace, if present, MUST appear as the first attribute. The remaining namespace declaration attributes MUST appear in lexicographic order based on [local name].

> Aside: In particular, this means that xmlns:n10 comes before xmlns:n2.

The attributes that are not namespace declarations MUST be lexicographically ordered on [namespace name] as the primary key and [local name] as the secondary key.

CDATA sections SHALL NOT be used.

Each ampersand character ('&', U+0026) in an attribute item's [normalized value] MUST be serialized as the entity reference &. Each open angle bracket character ('<', U+003C) in an attribute item's [normalized value] MUST be serialized as the entity reference <. Each double quote character (U+0022) in an attribute item's [normalized value] MUST be serialized as the entity reference ". Each character in the range U+0001 to U+001F or U+007F to U+009F in an attribute item's [normalized value] MUST be serialized as a character reference. No other character in a [normalized value] is permitted to be serialized as an entity reference or character reference.

Each character item with the [character code] U+0026 (the ampersand character) MUST be serialized as the entity reference &. Each character item with the [character code] U+003C (the open angle bracket character) MUST be serialized as the entity reference <. Each character item with the [character code] U+003E (the closing angle bracket character) MUST be serialized as the entity reference >. Each character item with a [character code] in the range U+0001 to U+0008, U+000B to U+001F, or U+007F to U+009F MUST be serialized as a character reference. No other character item is permitted to be serialized as an entity reference or character reference.

Character references, where they are permitted, SHALL use uppercase hexadecimal with no leading zeroes. For example, the carriage return character is represented as
.

A space character (U+0020) in an attribute item's [normalized value] MUST be serialized as a single U+0020 character.

A character item with the [character code] U+000A MUST be serialized as a single U+000A character.

The white space separating the [target] and [content] in the serialization of a processing instruction item SHALL be exactly one space character (U+0020).

Aside: A processing instruction or comment can only appear in a CRXER encoding if it is embedded in a Markup value.

6.12.3. Unicode Normalization in XML Version 1.1

XML Version 1.1 recommends, but does not absolutely require, that text be normalized according to Unicode Normalization Form C [UNICODE]. ASN.1 has no similar requirement on abstract values of string types, and ASN.1 canonical encoding rules depend on the code points of characters being preserved.

To accommodate both requirements, applications SHOULD normalize abstract values of ASN.1 character string types according to Unicode Normalization Form C at the time the values are created, but MUST NOT normalize a previously decoded abstract value of an ASN.1 character string type prior to re-encoding it. An application may, of course, normalize a decoded abstract value for other purposes, such as display to a user.

6.13. Syntax-Based Canonicalization

ASN.1 encoding rules are designed to preserve abstract values, but not to preserve every detail of each transfer syntax that is used. In the case of RXER, this means that the Infoset representation of an abstract value is not necessarily preserved when the abstract value is decoded and re-encoded (regardless of the encoding rules used). However, syntax-based canonicalization for XML documents (e.g., Canonical XML [CXML]) depends on the Infoset of an XML document being preserved. The Infoset representation of an XML document containing the RXER encoding of an ASN.1 abstract value potentially changes if that value is decoded and re-encoded, disrupting the Canonical XML representation. Extra normalization is required if RXER is to be usefully deployed in environments where syntax-based canonicalization is used.

Prior to applying syntax-based canonicalization to an XML document, any element items in the Infoset representation of the document that correspond to the value of an ASN.1 top-level NamedType or have content and attributes that correspond to an ASN.1 abstract value MUST be replaced by the translation of the value according to CRXER.

If an application uses Canonical XML but has no knowledge of RXER, then it will not know to normalize RXER encodings. If RXER is deployed into an environment containing such applications, then the Infoset translation for CRXER SHOULD be used for all RXER encodings.

7. Transfer Syntax Identifiers

7.1. RXER Transfer Syntax

The following OBJECT IDENTIFIER has been assigned by xmled.org to identify the Robust XML Encoding Rules, under an arc assigned to xmled.org by the Internet Assigned Numbers Authority (IANA):

```
{ iso(1) identified-organization(3) dod(6)
  internet(1) private(4) enterprise(1)
  xmled(21472) asnx(1) encoding(1) rxer(0) }
```

This OBJECT IDENTIFIER would be used, for example, to describe the transfer syntax for an RXER encoded data-value in an EMBEDDED PDV value.

7.2. CRXER Transfer Syntax

The following OBJECT IDENTIFIER has been assigned by xmled.org to identify the Canonical Robust XML Encoding Rules, under an arc assigned to xmled.org by the IANA:

```
{ iso(1) identified-organization(3) dod(6)
  internet(1) private(4) enterprise(1)
  xmled(21472) asnx(1) encoding(1) crxer(1) }
```

This OBJECT IDENTIFIER would be used, for example, to describe the transfer syntax for a CRXER encoded data-value in an EMBEDDED PDV value.

8. Relationship to XER

The Robust XML Encoding Rules (RXER) and the XML Encoding Rules (XER) [X.693] are separate, distinctly different and incompatible ASN.1 encoding rules for producing XML markup from ASN.1 abstract values. RXER is therefore unrelated to the XML value notation of X.680 [X.680].

This section describes some of the major differences between RXER and XER.

There are essentially two varieties of XER: BASIC-XER (with a canonical form called CANONICAL-XER) and EXTENDED-XER. The significant difference between the two varieties is that XER encoding instructions are used by EXTENDED-XER, but are ignored by BASIC-XER (and therefore by CANONICAL-XER). There isn't a canonical variant of EXTENDED-XER. Characteristics that are common to BASIC-XER and EXTENDED-XER will simply be noted as being characteristics of XER.

Elements and attributes are the fundamental discrete structures of an XML document. Not surprisingly, schema languages for XML typically have the means to describe, name, and reference global (i.e., top-level) elements and attributes. Global type definitions are seen more as a convenience for defining the contents of elements and attributes. Traditional ASN.1 has the means to define global types (and other global constructs that support the definition of types) but nothing akin to a global element or attribute definition. The fundamental difference between RXER and XER is in how this omission is addressed.

With XER, type definitions are also regarded as being element definitions by default, or as attribute definitions in the presence of an XER ATTRIBUTE encoding instruction. In some circumstances an anonymous Type is required to define an element, which leads to element names like <BOOLEAN> and <SEQUENCE>. NamedType notation also defines local elements, and there are some curious cases in EXTENDED-XER where NamedType notation can define a global type. So under XER, types can be defined by either Type or NamedType notation, and elements and attributes can also be defined by either Type or NamedType notation.

With RXER, types are only defined by Type notation and elements and attributes are only defined by NamedType notation. Global element and attribute definitions are made possible by top-level NamedType notation in an RXER encoding control section.

RXER, with its clean separation of Type notation for types and NamedType notation for elements and attributes, is a better basis than XER for translating an ASN.1 specification into an XML representation (i.e., ASN.X [ASN.X]) or a compatible XML Schema, where type, element, and attribute definitions are also distinctly separate constructs.

There is usually a requirement on applications specified in ASN.1 to maintain backward compatibility with the encodings generated by previous versions. The encodings in question are typically BER. Even with the backward-compatibility constraint there is still considerable leeway for specification writers to rewrite the earlier specification. For example, they could rename types, factor out an

in-line type definition as a defined type (or the reverse), or replace a type definition with an equivalent parameterized reference. These changes produce no change to BER, DER, CER [X.690], Packed Encoding Rules (PER) [X.691], or Generic String Encoding Rules (GSER) [GSER] encodings (so specification writers have felt free to make such changes to improve their specification), but can change the names of elements in the XER encoding because XER uses types as element definitions. The RXER encoding is immune to this problem, thus RXER encodings are more stable than XER encodings over successive revisions of an ASN.1 specification (which explains the first 'R' in RXER). This has an obvious benefit for interoperability.

RXER has special provisions for encoding values of the QName and Markup types. QName is used to hold qualified names and Markup can be used to hold arbitrary untyped markup. XER doesn't recognize any special types like these, but it is possible to get the same effects as RXER's QName and Markup types by using XER encoding instructions. Since CANONICAL-XER ignores encoding instructions, this means that under XER an application can either support qualified names and untyped markup, or support canonical XML encodings, but not both. In contrast, CRXER has canonicalization rules for qualified names and for Markup. Furthermore, EXTENDED-XER does not address the issues of normalization of untyped data for other ASN.1 canonical encoding rules (e.g., for DER; see Section 4.1.2) or normalization of XML encodings for syntax-based canonicalization (e.g., for Canonical XML; see Section 6.13).

Both EXTENDED-XER and RXER use encoding instructions to define attributes, union types, and list types, among other things. Since CANONICAL-XER ignores encoding instructions, this means that under XER an application must choose between making use of attributes, union types, list types, etc., or supporting canonical XML encodings. In contrast, the canonicalization rules for CRXER encompass all the encoding instructions for RXER.

9. Security Considerations

RXER does not necessarily enable the exact BER octet encoding of values of the TeletexString, VideotexString, GraphicString, or GeneralString types to be reconstructed, so a transformation from DER to RXER and back to DER may not reproduce the original DER encoding. This is a result of inadequate normalization of values of these string types in DER. A character in a TeletexString value (for example) that corresponds to a specific ISO 10646 character can be encoded for BER in a variety of ways that are indistinguishable in an

RXER re-encoding of the TeletexString value. DER does not mandate one of these possible character encodings in preference to all others.

Because of the above, RXER MUST NOT be used to re-encode, whether for storage or transmission, ASN.1 abstract values whose original DER or CER encoding must be recoverable, and whose type definitions involve the TeletexString, VideotexString, GraphicString, or GeneralString type. Such recovery is needed for the verification of digital signatures. In such cases, protocols ought to use DER or a DER-reversible encoding. In other cases where ASN.1 canonical encoding rules are used, values of the Markup type must be normalized as described in Section 4.1.2.

A transformation from CRXER to BER and back to CRXER does reproduce the original CRXER encoding, therefore it is safe to use BER, DER, or CER to re-encode ASN.1 abstract values whose original CRXER encoding must be recoverable.

Digital signatures may also be calculated on the Canonical XML representation of an XML document. If RXER encodings appear in such documents, then applications must normalize the encodings as described in Section 6.13.

The null character (U+0000) cannot be represented in XML and hence cannot be transmitted in an RXER encoding. Null characters in abstract values of ASN.1 string types will be dropped if the values are RXER encoded; therefore, RXER MUST NOT be used by applications that attach significance to the null character.

When interpreting security-sensitive fields, and in particular fields used to grant or deny access, implementations MUST ensure that any comparisons are done on the underlying abstract value, regardless of the particular encoding used. Comparisons of Markup values MUST operate as though the values have been normalized as specified in Section 4.1.2.

10. Acknowledgements

The technology described in this document is the product of a research project begun jointly by Adacel Technologies Limited and Deakin University, and subsequently refined and completed by eB2Bcom.

11. IANA Considerations

The IANA has registered a new XML namespace in accordance with RFC 3688 [XMLREG].

URI: urn:ietf:params:xml:ns:asnx

Registrant Contact: Steven Legg <steven.legg@eb2bcom.com>

XML: None

12. References

12.1. Normative References

[BCP14] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.

[XMLREG] Mealling, M., "The IETF XML Registry", RFC 3688, January 2004.

[URI] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RXEREI] Legg, S., "Encoding Instructions for the Robust XML Encoding Rules (RXER)", RFC 4911, July 2007.

[ASN.X] Legg, S., "Abstract Syntax Notation X (ASN.X)", RFC 4912, July 2007.

[X.680] ITU-T Recommendation X.680 (07/02) | ISO/IEC 8824-1, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[X.680-1] ITU-T Recommendation X.680 (2002) Amendment 1 (10/03) | ISO/IEC 8824-1:2002/Amd 1:2004, Support for EXTENDED-XER.

[X.681] ITU-T Recommendation X.681 (07/02) | ISO/IEC 8824-2, Information technology - Abstract Syntax Notation One (ASN.1): Information object specification.

[X.682] ITU-T Recommendation X.682 (07/02) | ISO/IEC 8824-3, Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification.

[X.683] ITU-T Recommendation X.683 (07/02) | ISO/IEC 8824-4, Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.

[X.690] ITU-T Recommendation X.690 (07/02) | ISO/IEC 8825-1, Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[UCS] ISO/IEC 10646-1:2000, Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 4.0", Boston, MA, Addison-Wesley Developers Press, 2003. ISBN 0-321-18578-1.

[XML10] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, http://www.w3.org/TR/2006/REC-xml-20060816, August 2006.

[XML11] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", W3C Recommendation, http://www.w3.org/TR/2006/REC-xml11-20060816, August 2006.

[XMLNS10] Bray, T., Hollander, D., Layman, A., and R. Tobin, "Namespaces in XML 1.0 (Second Edition)", W3C Recommendation, http://www.w3.org/TR/2006/REC-xml-names-20060816, August 2006.

[XMLNS11] Bray, T., Hollander, D., Layman, A. and R. Tobin, "Namespaces in XML 1.1 (Second Edition)", W3C Recommendation, http://www.w3.org/TR/2006/REC-xml-names11-20060816, August 2006.

[INFOSET] Cowan, J. and R. Tobin, "XML Information Set (Second Edition)", W3C Recommendation, http://www.w3.org/TR/2004/REC-xml-infoset-20040204, February 2004.

[XSD1] Thompson, H., Beech, D., Maloney, M. and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", W3C Recommendation, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/, October 2004.

12.2. Informative References

[GSER] Legg, S., "Generic String Encoding Rules (GSER) for ASN.1 Types", RFC 3641, October 2003.

[X.691] ITU-T Recommendation X.691 (07/02) | ISO/IEC 8825-4:2002, Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).

[X.693] ITU-T Recommendation X.693 (12/01) | ISO/IEC 8825-4:2002, Information technology - ASN.1 encoding rules: XML encoding rules (XER).

[XSD2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/, October 2004.

[CXML] Boyer, J., "Canonical XML Version 1.0", W3C Recommendation, http://www.w3.org/TR/2001/REC-xml-c14n-20010315, March 2001.

Appendix A. Additional Basic Definitions Module

This appendix is normative.

```
AdditionalBasicDefinitions
    { iso(1) identified-organization(3) dod(6)
      internet(1) private(4) enterprise(1)
      xmled(21472) asnx(1) module(0) basic(0) }

-- Copyright (C) The IETF Trust (2007).  This version of
-- this ASN.1 module is part of RFC 4910; see the RFC itself
-- for full legal notices.
--
-- Regarding this ASN.1 module or any portion of it, the authors
-- make no guarantees and are not responsible for any damage
-- resulting from its use.  The authors grant irrevocable permission
-- to anyone to use, modify, and distribute it in any way that does
-- not diminish the rights of anyone else to use, modify, and
-- distribute it, provided that redistributed derivative works do
-- not contain misleading author or version information.
-- Derivative works need not be licensed under similar terms.

DEFINITIONS
RXER INSTRUCTIONS
AUTOMATIC TAGS
EXTENSIBILITY IMPLIED ::= BEGIN

Markup ::= CHOICE {
    text    SEQUENCE {
        prolog      UTF8String (SIZE(1..MAX)) OPTIONAL,
        prefix      NCName OPTIONAL,
        attributes  UTF8String (SIZE(1..MAX)) OPTIONAL,
        content     UTF8String (SIZE(1..MAX)) OPTIONAL
    }
}

AnyURI ::= UTF8String (CONSTRAINED BY
               { -- conforms to the format of a URI -- })

NCName ::= UTF8String (CONSTRAINED BY
               { -- conforms to the NCName production of
                 -- Namespaces in XML 1.0 -- })

Name ::= UTF8String (CONSTRAINED BY
               { -- conforms to the Name production of XML -- })
```

```
   QName ::= SEQUENCE {
       namespace-name  AnyURI OPTIONAL,
       local-name      NCName
   }

   ENCODING-CONTROL RXER

       TARGET-NAMESPACE "urn:ietf:params:xml:ns:asnx" PREFIX "asnx"

       COMPONENT context [ATTRIBUTE] [LIST] SEQUENCE OF prefix NCName

   END
```

Authors' Addresses

Dr. Steven Legg
eB2Bcom
Suite 3, Woodhouse Corporate Centre
935 Station Street
Box Hill North, Victoria 3129
AUSTRALIA

Phone: +61 3 9896 7830
Fax: +61 3 9896 7801
EMail: steven.legg@eb2bcom.com

Dr. Daniel Prager

EMail: dap@austhink.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.