

Abstract Syntax Notation X (ASN.X)

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

Copyright (C) The IETF Trust (2007).

Abstract

Abstract Syntax Notation X (ASN.X) is a semantically equivalent Extensible Markup Language (XML) representation for Abstract Syntax Notation One (ASN.1) specifications. ASN.X completely avoids the numerous ambiguities inherent in the ASN.1 language; therefore, specifications written in ASN.X are much easier to parse and manage than original ASN.1 specifications. ASN.X, together with the Robust XML Encoding Rules (RXER), constitutes a schema language for XML documents that offers, through other ASN.1 encoding rules, alternative compact binary encodings for XML instance documents.

Table of Contents

1. Introduction	4
2. Conventions	5
3. General Considerations	6
3.1. Annotations	7
4. ModuleDefinition Translation	8
5. Translation of Assignments	11
5.1. Referencing Named Constructs	11
5.2. Importing Namespaces	12
5.3. TypeAssignment Translation	14
5.4. ValueAssignment and XMLValueAssignment Translation	14
5.5. ValueSetTypeAssignment Translation	15
5.6. ObjectClassAssignment Translation	15
5.7. ObjectAssignment Translation	16
5.8. ObjectSetAssignment Translation	16
5.9. ParameterizedAssignment Translation	17
6. Translation of Types	17
6.1. Identifier Replacement	17
6.2. DefinedType Translation	18
6.3. Translation of Built-in Types	20
6.4. BitStringType Translation	21
6.5. IntegerType Translation	22
6.6. EnumeratedType Translation	24
6.7. PrefixedType Translation	25
6.7.1. Short Form TaggedType Translation	28
6.7.2. Long Form TaggedType Translation	29
6.8. SelectionType Translation	30
6.9. InstanceOfType Translation	31
6.10. ObjectClassFieldType Translation	31
6.11. TypeFromObject and ValueSetFromObjects Translation	32
6.12. Translation of Combining Types	32
6.12.1. NamedType Translation	32
6.12.2. SequenceType Translation	36
6.12.3. SetType Translation	38
6.12.4. ChoiceType Translation	39
6.12.5. Translation of UNION Types	40
6.12.6. SequenceOfType Translation	41
6.12.7. Translation of LIST Types	42
6.12.8. SetOfType Translation	42
6.12.9. Effect of Insertion Encoding Instructions	43
6.13. Translation of Constrained Types	43
6.13.1. Constraint Translation	46
6.13.2. UserDefinedConstraint Translation	46
6.13.3. TableConstraint Translation	47
6.13.4. ContentsConstraint Translation	49
6.13.5. ExceptionSpec Translation	50

7. Translation of Values	51
7.1. Translation of Literal Values	53
7.2. Translation of Notational Values	54
7.2.1. DefinedValue Translation	56
7.2.2. BuiltinValue Translation	57
7.2.3. ValueFromObject Translation	60
7.2.4. ObjectClassFieldValue Translation	60
8. Translation of Value Sets	61
8.1. ElementSetSpecs Translation	62
8.2. ElementSetSpec Translation	62
8.3. SubtypeElements Translation	63
8.3.1. ValueRange Translation	64
8.3.2. InnerTypeConstraints Translation	65
9. Translation of Object Classes	66
9.1. DefinedObjectClass Translation	66
9.2. ObjectClassDefn Translation	68
9.2.1. TypeFieldSpec Translation	68
9.2.2. FixedTypeValueFieldSpec Translation	69
9.2.3. FixedTypeValueSetFieldSpec Translation	70
9.2.4. VariableTypeValueFieldSpec Translation	71
9.2.5. VariableTypeValueSetFieldSpec Translation	73
9.2.6. FieldName Translation	74
9.2.7. ObjectFieldSpec Translation	75
9.2.8. ObjectSetFieldSpec Translation	76
10. Translation of Objects	77
10.1. DefinedObject Translation	77
10.2. ObjectDefn Translation	78
10.3. ObjectFromObject Translation	80
11. Translation of Object Sets	80
11.1. DefinedObjectSet Translation	81
11.2. ObjectSetElements Translation	82
11.2.1. ObjectSetFromObjects Translation	83
12. Translation of Information From Objects	83
13. Translation of Parameterized Definitions	83
14. EncodingControlSections Translation	93
15. Security Considerations	94
16. Acknowledgements	94
17. References	95
17.1. Normative References	95
17.2. Informative References	97
Appendix A. ASN.1 for ASN.X	95
Appendix B. ASN.X for ASN.X	115

1. Introduction

A full parser for the Abstract Syntax Notation One (ASN.1) language [X.680][X.680-1][X.681][X.682][X.683] is difficult to implement due to numerous ambiguities in the notation. For example, certain notations for a ValueSet, Object, ObjectSet, DummyReference, or SimpleTableConstraint. An ObjectClassAssignment, ObjectAssignment, or ObjectSetAssignment resembles respectively a TypeAssignment, ValueAssignment, or ValueSetTypeAssignment. A FixedTypeValueFieldSpec or FixedTypeValueSetFieldSpec resembles respectively an ObjectFieldSpec or ObjectSetFieldSpec, and an ObjectClassFieldType resembles InformationFromObjects notation. In general, such ambiguities can only be resolved once the entire specification has been parsed. There are other notations that are not mutually ambiguous but still require several lexical tokens to be scanned before they can be distinguished from each other. The difficulty of parsing ASN.1 is an impediment to its wider adoption.

This document defines a semantically equivalent Extensible Markup Language (XML) [XML10][XML11] representation for ASN.1 specifications called Abstract Syntax Notation X (ASN.X). An ASN.X module is a well-formed and valid XML document conforming to XML namespaces [XMLNS10][XMLNS11]. ASN.X completely avoids the inherent ambiguities of the ASN.1 language; therefore, specifications written in ASN.X are much easier to parse and manage than original ASN.1 specifications. For example, any conformant XML processor forms the basis of an ASN.1 toolkit.

ASN.X, together with the Robust XML Encoding Rules (RXER) [RXER], constitutes a schema language for XML documents that offers, through other ASN.1 encoding rules, alternative compact binary encodings for XML instance documents conforming to an ASN.X specification. ASN.X definitions can also incorporate type, element, and attribute definitions from XML Schema [XSD1] documents, RELAX NG [RNG] documents, or Document Type Definitions (DTDs) [XML10][XML11].

ASN.X is defined in terms of rules for translating from an ASN.1 specification. This does not preclude an ASN.X module being written directly without a pre-existing ASN.1 module; however, such an ASN.X module is considered valid if and only if there exists, in principle, an ASN.1 module that when translated would yield the ASN.X module.

The format for ASN.X has also been designed so that the content of an ASN.X module conforms to the RXER encoding of an abstract value of an ASN.1 type, the ModuleDefinition type, presented in Appendix A. This means that it is possible to decode an ASN.X module using an RXER decoder and then re-encode the abstract value (for storage or

transmission) using any of the other encoding rules for ASN.1. Thus, the "X" in ASN.X can be regarded as standing for either XML or RXER, or more generally, for any set of ASN.1 encoding rules.

The ASN.X translation of the ASN.1 module in Appendix A is presented in Appendix B.

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.

Throughout this document "type" shall be taken to mean an ASN.1 type, and "value" shall be taken to mean an ASN.1 abstract value.

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 description of the translation of an ASN.1 module into an ASN.X module 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". Element items will be referred to by their [local name] in angle brackets, e.g., "the <type> element item" means the element item with the [local name] "type". Attribute items will be referred to by their [local name], e.g., "the type attribute item" means the attribute item with the [local name] "type".

This document uses the namespace prefix "asn:" to stand for the namespace name "urn:ietf:params:xml:ns:asn", though in practice any valid namespace prefix is permitted in ASN.X.

Encoding instructions [X.680-1] referenced by name in this specification are encoding instructions for RXER [RXEREI]. The associated provisions do not apply to encoding instructions for other encoding rules that happen to have the same name.

Code points for characters [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. General Considerations

ASN.X is defined in terms of rules for translating an ASN.1 module into a synthetic Infoset. This synthetic Infoset is then serialized into a well-formed and valid XML document (the ASN.X module) in the same manner that the synthetic Infoset for a non-canonical RXER encoding is serialized into an XML document (see Section 6.12 of the specification for RXER [RXER]).

Aside: The serialization permits CDATA sections, character references, and parsed entity references. However, note that an ASN.X module may be transferred as data in a protocol and that some protocols disallow entity references.

Apart from the [document element] of the document item for an ASN.X module, the translation of some ASN.1 construct belongs to the [children] or [attributes] of an enclosing element item.

Where the translation of the construct is an element item, it is appended to the [children] of the enclosing element item. Elements MUST be appended to the [children] of the enclosing element item in the order described. Translators MAY add white space character items (i.e., U+0020, U+0009, U+000D and U+000A) to the [children] of any element item (to improve the layout) except element items with the [local name] "literalValue", "fieldName", or "restrictBy".

Aside: White space in the [children] of <fieldName> and <restrictBy> element items is explicitly covered under their respective descriptions.

Where the translation of the construct is an attribute item, it is added to the [attributes] of the enclosing element item. The order of attribute items is not significant. Translators MAY add leading and trailing white space characters to the [normalized value] of any attribute item except an attribute item with the [local name] "literalValue".

Aside: An attribute or element item with the [local name] "literalValue" holds an RXER Infoset translation of an abstract value, and white space characters may be significant in that abstract value. In most cases, RXER itself permits optional leading and trailing white space characters in the Infoset translation.

Translators MAY add comment and processing instruction (PI) items to the [children] of any element item except an element item with the [local name] "literalValue".

Aside: In most cases, RXER itself permits comment and PI items in the [children] of the element items with the [local name] "literalValue".

Aside: Note that an ASN.X module may be transferred as data in a protocol and that some protocols disallow processing instructions.

The [in-scope namespaces] and [namespace attributes] for <literalValue> and <restrictBy> element items are determined according to Section 6.10 of the specification for RXER [RXER]. The [in-scope namespaces] and [namespace attributes] for other element items in the translation are determined according to Section 6.2.2.1 of the specification for RXER.

The [namespace name] of any element item or attribute item generated by the translation from an ASN.1 specification has no value unless specified otherwise. In those cases where the [namespace name] of an element item has a value, the [prefix] of the element item is determined according to Section 6.2.2.2 of the specification for RXER. In those cases where the [namespace name] of an attribute item has a value, the [prefix] of the attribute item is determined according to Section 6.2.3.1 of the specification for RXER.

Aside: Non-canonical RXER allows all valid namespace prefixes and all valid placements for their corresponding namespace declaration attributes.

Whenever an element item is added to the [children] of an enclosing element item, the enclosing element item becomes the [parent] of the element item.

Whenever an attribute item is added to the [attributes] of an element item, the element item becomes the [owner element] of the attribute item. For each attribute item, the [specified] property is set to true, the [attribute type] has no value, and the value of the [references] property is set to unknown.

3.1. Annotations

In a number of places, as indicated in subsequent sections, the translator is permitted to add an element item with the [local name] "annotation". The [children] and [attributes] of the <annotation> element item are at the discretion of the translator.

Typical uses of the <annotation> element item would be to hold comments from the ASN.1 specification that are normative in nature, e.g., a comment in a user-defined constraint, or to hold directives for an ASN.1 compiler.

Free text or XML comments in an <annotation> element will be preserved in a Canonical RXER (CRXER) encoding [RXER] (because the corresponding ASN.1 type for the <annotation> element item is the Markup type [RXER]), while XML comments outside <annotation> elements will not be preserved.

Vendors using the <annotation> element items to hold ASN.1 compiler directives (as attributes or child elements of the <annotation> element) SHOULD use element or attribute names that are qualified with a namespace name specific to the vendor.

4. ModuleDefinition Translation

The translation of a ModuleDefinition [X.680] (an ASN.1 module) is an element item with the [local name] "module" and the [namespace name] "urn:ietf:params:xml:ns:asn1" (i.e., an <asn1:module> element item). The element item is typically the [document element] of a document item.

An attribute item with the [local name] "format" and [normalized value] "1.0" MAY be added to the [attributes] of the <asn1:module> element item.

An ASN.1 module has a schema identity URI if it contains a SCHEMA-IDENTITY encoding instruction, in which case the schema identity URI is the character string specified by the AnyURIValue of the SCHEMA-IDENTITY encoding instruction.

If the ASN.1 module being translated has a schema identity URI, then an attribute item with the [local name] "schemaIdentity" SHALL be added to the [attributes] of the <asn1:module> element item. The [normalized value] of this attribute item is the schema identity URI of the module.

If the target namespace [RXEREI] for the ASN.1 module is not absent, then an attribute item with the [local name] "targetNamespace" SHALL be added to the [attributes] of the <asn1:module> element item. The [normalized value] of this attribute item is the target namespace of the module.

Aside: An ASN.1 module has a target namespace if it contains a TARGET-NAMESPACE encoding instruction.

If the ASN.1 module contains a TARGET-NAMESPACE encoding instruction that specifies a Prefix, then an attribute item with the [local name] "targetPrefix" SHALL be added to the [attributes] of the

<asn:module> element item. The [normalized value] of this attribute item is the character string specified by the NCNameValue in the Prefix.

In examples in the remainder of this document, the namespace prefix "tns:" is used to stand for the target namespace of the module being translated.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <asn:module> element item. The [normalized value] of this attribute item is the modulereference in the ModuleIdentifier in the ModuleDefinition.

If the DefinitiveIdentifier in the ModuleIdentifier in the ModuleDefinition is not empty, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <asn:module> element item. The [normalized value] of this attribute item is the RXER character data translation [RXER] of the DefinitiveIdentifier.

If the TagDefault in the ModuleDefinition is empty, then an attribute item with the [local name] "tagDefault" and [normalized value] "explicit" SHALL be added to the [attributes] of the <asn:module> element item.

If the TagDefault in the ModuleDefinition is not empty and the first keyword in the TagDefault is not "AUTOMATIC", then an attribute item with the [local name] "tagDefault" SHALL be added to the [attributes] of the <asn:module> element item. The [normalized value] of this attribute item is the first keyword in the TagDefault with all letters downcased, i.e., "explicit" or "implicit".

If the TagDefault in the ModuleDefinition is not empty and the first keyword in the TagDefault is "AUTOMATIC", then an attribute item with the [local name] "tagDefault" and [normalized value] "automatic" MAY be added to the [attributes] of the <asn:module> element item.

If the ExtensionDefault in the ModuleDefinition is not empty, then an attribute item with the [local name] "extensibilityImplied" and [normalized value] "true" or "1" SHALL be added to the [attributes] of the <asn:module> element item.

If the ExtensionDefault in the ModuleDefinition is empty, then an attribute item with the [local name] "extensibilityImplied" and [normalized value] "false" or "0" MAY be added to the [attributes] of the <asn:module> element item.

An element item with the [local name] "annotation" MAY be added to the [children] of the <asn:module> element item.

The translation of each Assignment in the AssignmentList in the ModuleBody in the ModuleDefinition of the module being translated SHALL be appended to the [children] of the <asn:module> element item.

If the EncodingControlSections instance in the ModuleDefinition contains an EncodingControlSection for RXER, then the translation of each NamedType in a TopLevelComponent [RXEREI] nested in the EncodingInstructionAssignmentList SHALL be added to the [children] of the <asn:module> element item. The relative order of the top-level components [RXEREI] SHOULD be preserved in the translation; however, the translations of the top-level components MAY be interspersed with the translations of the assignments in the AssignmentList.

The translation of the EncodingControlSections instance in the ModuleDefinition of the module being translated SHALL be appended to the [children] of the <asn:module> element item.

Example

```
MyModule DEFINITIONS
  IMPLICIT TAGS
  EXTENSIBILITY IMPLIED ::=
  BEGIN

  MyType ::= INTEGER

  ENCODING-CONTROL RXER

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

    COMPONENT myElement INTEGER

  END

<asn:module xmlns:asn="urn:ietf:params:xml:ns:asn"
  name="MyModule"
  schemaIdentity="http://example.com/id/MyModule"
  targetNamespace="http://example.com/ns/MyModule"
  tagDefault="implicit"
  extensibilityImplied="true">

  <namedType name="MyType" type="asn:INTEGER"/>
```

```
<element name="myElement" type="asn:INTEGER"/>

</asn:module>
```

5. Translation of Assignments

5.1. Referencing Named Constructs

An Assignment in ASN.1 associates a reference name with a Type, Value, ValueSet, ObjectClass, Object, or ObjectSet. For ASN.X, an Assignment is also regarded as associating an expanded name [XMLNS10][XMLNS11] with the Type, Value, ValueSet, ObjectClass, Object, or ObjectSet. ASN.X uses these expanded names, rendered as qualified names [XMLNS10][XMLNS11], in place of the references in an ASN.1 specification.

In every case, the local name of the expanded name is the `typereference`, `valuereference`, `objectclassreference`, `objectreference`, or `objectsetreference` in the Assignment (i.e., the [normalized value] of the name attribute item in the translation of the Assignment, ignoring white space characters). If the target namespace of the ASN.1 module in which the Assignment 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. When the expanded name is rendered as a qualified name, the namespace prefix is determined according to Section 6.7.11.1 of the specification for RXER [RXER].

If an ASN.1 specification contains two or more modules where the target namespace is absent, then there exists the possibility that the expanded names defined by the ASN.X translations of those modules are not distinct. The expanded names are not distinct if:

- (1) two or more type or value set assignments define the same `typereference`, or
- (2) two or more value assignments define the same `valuereference`, or
- (3) two or more object class assignments define the same `objectclassreference`, or
- (4) two or more object assignments define the same `objectreference`, or
- (5) two or more object set assignments define the same `objectsetreference`, or

- (6) two or more top-level element components [RXEREI] have the same local name, or
- (7) two or more top-level attribute components [RXEREI] have the same local name.

If the expanded names are not distinct, then an unambiguous translation into ASN.X does not exist unless each of the modules has a SCHEMA-IDENTITY encoding instruction. Consequently, if two or more modules where the target namespace is absent are being translated into ASN.X and the reference names defined in those modules will not be distinct, then as a local action prior to the translation, a SCHEMA-IDENTITY encoding instruction MUST be added to each of the modules that defines one or more of the indistinct expanded names and that does not already have a SCHEMA-IDENTITY encoding instruction. The character string (a URI) specified by the AnyURIValue of each added SCHEMA-IDENTITY encoding instruction is freely chosen by the translator, subject to the condition that these character strings are distinct [RXEREI].

Aside: Although this means that different translators might produce ASN.X modules that are syntactically different for any given ASN.1 module, those ASN.X modules will be semantically equivalent to each other and to the original ASN.1 module.

TARGET-NAMESPACE and SCHEMA-IDENTITY encoding instructions are RECOMMENDED for every ASN.1 module.

5.2. Importing Namespaces

An Assignment is referenced from an ASN.X module if its associated expanded name appears as a qualified name in the [normalized value] of an attribute item with the [local name] "type", "value", "class", "object", or "objectSet". These references are categorized as direct references. An Assignment or top-level component is also referenced from an ASN.X module if its expanded name appears as a qualified name in the [normalized value] of an attribute item with the [local name] "ref". This reference is only categorized as direct if the ref attribute is not the result of the translation of a DefinedType subject to a TYPE-REF encoding instruction or a NamedType subject to an ATTRIBUTE-REF or ELEMENT-REF encoding instruction.

Aside: In the case of an indirect reference, an attribute item with the [local name] "embedded" and [normalized value] "true" or "1" will also be present.

Definition (external module): An external module is any module other than the module being translated and the AdditionalBasicDefinitions module [RXER].

Aside: The AdditionalBasicDefinitions module is always assumed to be imported, as are all the built-in types and object classes of ASN.1.

An element item with the [local name] "import" SHALL be added to the [children] of the <asn:module> element item for each external module containing Assignments or top-level components that are directly referenced from the ASN.X module. An <import> element item MAY be added to the [children] of the <asn:module> element item for any other external module.

An attribute item with the [local name] "name" SHOULD be added to the [attributes] of the <import> element item. The [normalized value] of this attribute item is the modulereference in the ModuleIdentifier in the ModuleDefinition of the external module.

If the DefinitiveIdentifier in the ModuleIdentifier in the ModuleDefinition of the external module is not empty, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <import> element item. The [normalized value] of this attribute item is the RXER character data translation of the DefinitiveIdentifier.

If the external module has a schema identity URI, then an attribute item with the [local name] "schemaIdentity" SHALL be added to the [attributes] of the <import> element item. The [normalized value] of this attribute item is the schema identity URI of the external module.

If the target namespace of the external module is not absent, then an attribute item with the [local name] "namespace" SHALL be added to the [attributes] of the <import> element item. The [normalized value] of this attribute item is the target namespace of the external module.

An attribute item with the [local name] "schemaLocation" MAY be added to the [attributes] of the <import> element item. The [normalized value] of this attribute item is a URI [URI] indicating the physical location of the ASN.X translation of the external module.

The <import> element items MUST follow an <annotation> element item (if present) and MUST precede any other element items in the [children] of the <asn:module> element item.

Note that because of the way parameterized references are expanded in ASN.X (see Section 13), the modules in the Imports in the ModuleBody in the ModuleDefinition may not correspond exactly to the <import> element items.

5.3. TypeAssignment Translation

The translation of a TypeAssignment is an element item with the [local name] "namedType". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedType> element item. The [normalized value] of this attribute item is the typereference on the left-hand side of the assignment.

An element item with the [local name] "annotation" MAY be added to the [children] of the <namedType> element item. The translation of the Type on the right-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedType> element item.

Example

```
MyType ::= INTEGER
```

```
<namedType name="MyType" type="asn:INTEGER"/>
```

5.4. ValueAssignment and XMLValueAssignment Translation

The translation of a ValueAssignment is an element item with the [local name] "namedValue". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedValue> element item. The [normalized value] of this attribute item is the valuereference on the left-hand side of the assignment.

An element item with the [local name] "annotation" MAY be added to the [children] of the <namedValue> element item. The translation of the Type on the left-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedValue> element item. The translation of the Value on the right-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedValue> element item.

Example

```
myValue INTEGER ::= 10
```

```
<namedValue name="myValue" type="asn:INTEGER" literalValue="10"/>
```

An XMLValueAssignment is converted into the equivalent ValueAssignment and then translated as a ValueAssignment. Note that the ASN.X representation for a Value is unrelated to XMLTypedValue.

5.5. ValueSetTypeAssignment Translation

The translation of a ValueSetTypeAssignment is an element item with the [local name] "namedValueSet". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedValueSet> element item. The [normalized value] of this attribute item is the typereference on the left-hand side of the assignment.

An element item with the [local name] "annotation" MAY be added to the [children] of the <namedValueSet> element item. The translation of the Type on the left-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedValueSet> element item. The translation of the ValueSet on the right-hand side of the assignment SHALL be added to the [children] of the <namedValueSet> element item.

Example

```
MyValueSet INTEGER ::= { 10 }

<namedValueSet name="MyValueSet" type="asn:INTEGER">
  <valueSet>
    <literalValue>10</literalValue>
  </valueSet>
</namedValueSet>
```

5.6. ObjectClassAssignment Translation

The translation of an ObjectClassAssignment is an element item with the [local name] "namedClass". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedClass> element item. The [normalized value] of this attribute item is the objectclassreference on the left-hand side of the assignment.

An element item with the [local name] "annotation" MAY be added to the [children] of the <namedClass> element item. The translation of the ObjectClass on the right-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedClass> element item.

Example

```
MY-CLASS ::= TYPE-IDENTIFIER
```

```
<namedClass name="MY-CLASS" class="asn:TYPE-IDENTIFIER"/>
```

5.7. ObjectAssignment Translation

The translation of an ObjectAssignment is an element item with the [local name] "namedObject". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedObject> element item. The [normalized value] of this attribute item is the objectreference on the left-hand side of the assignment.

An element item with the [local name] "annotation" MAY be added to the [children] of the <namedObject> element item. The translation of the DefinedObjectClass on the left-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedObject> element item. The translation of the Object on the right-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedObject> element item.

Example

```
myObject TYPE-IDENTIFIER ::=
```

```
{ NULL IDENTIFIED BY { 1 3 14 3 2 26 } }
```

```
<namedObject name="myObject" class="asn:TYPE-IDENTIFIER">
```

```
<object>
```

```
<field name="id" literalValue="1.3.14.3.2.26"/>
```

```
<field name="Type" type="asn:NULL"/>
```

```
</object>
```

```
</namedObject>
```

5.8. ObjectSetAssignment Translation

The translation of an ObjectSetAssignment is an element item with the [local name] "namedObjectSet". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedObjectSet> element item. The [normalized value] of this attribute item is the objectsetreference on the left-hand side of the assignment.

An element item with the [local name] "annotation" MAY be added to the [children] of the <namedObjectSet> element item. The translation of the DefinedObjectClass on the left-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedObjectSet> element item. The translation of the ObjectSet on

the right-hand side of the assignment SHALL be added to the [children] or [attributes] of the <namedObjectSet> element item.

Example

```
MyObjectSet TYPE-IDENTIFIER ::= { myObject }

<namedObjectSet name="MyObjectSet" class="asn:TYPE-IDENTIFIER">
  <objectSet>
    <object ref="tns:myObject"/>
  </objectSet>
</namedObjectSet>
```

5.9. ParameterizedAssignment Translation

The translation of an ASN.1 specification into ASN.X replaces any reference to a parameterized definition [X.683] with the definition expanded in-line. Consequently, there is no direct translation for a ParameterizedAssignment, though its definition may come into play in the translation of references to the parameterized definition (see Section 13).

6. Translation of Types

The rules for translating the different varieties of Type are detailed in this section.

Note that the notation of ASN.1 is ambiguous where a Type is both prefixed [X.680-1] (e.g., tagged) and constrained. For example, the notation "[0] INTEGER (0..10)" could be interpreted as either a tagged ConstrainedType or a constrained TaggedType. For the purposes of the translation into ASN.X, the constraint is assumed to have higher precedence than the prefix, so the above notation would be taken to be a tagged ConstrainedType.

6.1. Identifier Replacement

Various RXER encoding instructions can be used to override an identifier in an ASN.1 specification with an NCName [XMLNS10]. The NCName is given preeminence in the ASN.X representation, and the identifier is not explicitly given if it is algorithmically related to the NCName. The cases where an NCName overrides an identifier are covered individually in other parts of this specification and make use of the following definition.

Definition (reduction): The reduction of an NCName is the string of characters resulting from the following operations performed in order on the NCName:

- (1) replace each full stop ('.', U+002E) and low line ('_', U+005F) character with a hyphen character ('-', U+002D),
- (2) remove every character except Latin letters (U+0041-U+005A, U+0061-U+007A), decimal digits (U+0030-U+0039), and hyphens (U+002D),
- (3) remove leading and trailing hyphen characters,
- (4) replace sequences of two or more hyphen characters with a single hyphen, and
- (5) convert the first character to lowercase if it is an uppercase letter.

Aside: If the reduction of an NCName is not the same as the identifier that the NCName replaces, then the identifier will be explicitly given in the translation into ASN.X.

6.2. DefinedType Translation

If a Type is a DefinedType in a ReferencedType, then the translation of the Type is the translation of the DefinedType.

If a DefinedType is not a ParameterizedType, ParameterizedValueSetType, or DummyReference and is not subject to a TYPE-REF or REF-AS-TYPE encoding instruction, then the translation of the DefinedType is either the attribute form translation of a type reference, or the element form translation of a type reference.

The attribute form translation of a type reference is an attribute item with the [local name] "type". The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced type definition (see Section 5.1). The attribute form translation SHALL NOT be used if this expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items (see Section 5.1).

The element form translation of a type reference is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced type definition. If this expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items, then an attribute item with the [local name] "context" SHALL be added to the

[attributes] of the <type> element item; otherwise, if the module containing the referenced type definition has a schema identity URI, then an attribute item with the [local name] "context" MAY be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is the schema identity URI of the module containing the type definition referenced by the DefinedType.

Aside: If a reference name is not distinct, then the module containing the referenced definition must have a schema identity URI (see Section 5.1).

An attribute item with the [local name] "embedded" and [normalized value] "false" or "0" MAY be added to the [attributes] of the <type> element item.

The translation of the DefinedType is the same whether the type definition is referenced by a typereference or an ExternalTypeReference.

If a DefinedType is subject to a TYPE-REF encoding instruction, then the translation of the DefinedType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is the RXER character data translation of the QNameValue in the TYPE-REF encoding instruction. If a ContextParameter is present in the RefParameters in the TYPE-REF encoding instruction, then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is the string value of the AnyURIValue in the ContextParameter. An attribute item with the [local name] "embedded" and [normalized value] "true" or "1" SHALL be added to the [attributes] of the <type> element item.

Aside: The embedded attribute item indicates whether a type is directly referenced as a DefinedType or indirectly referenced through a TYPE-REF encoding instruction. An ASN.1 type can be referenced either way. Type definitions in other schema languages cannot be directly referenced.

If a DefinedType is subject to a REF-AS-TYPE encoding instruction, then the translation of the DefinedType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An attribute item with the [local name] "elementType" SHALL be added to the [attributes] of the <type> element item. The

[normalized value] of this attribute item is the RXER character data translation of the NameValue in the REF-AS-TYPE encoding instruction. If a ContextParameter is present in the RefParameters in the REF-AS-TYPE encoding instruction, then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is the string value of the AnyURIValue in the ContextParameter.

Example

```
CHOICE {
  one      Foo,
  two      [RXER:TYPE-REF
            { namespace-name "http://www.example.com/PO1",
              local-name "PurchaseOrderType" }]
            Markup,
  three    [RXER:REF-AS-TYPE "product"
            CONTEXT "http://www.example.com/inventory"]
            Markup
}

<type>
  <choice>
    <element name="one" type="tns:Foo"/>
    <element name="two" xmlns:po="http://www.example.com/PO1">
      <type ref="po:PurchaseOrderType" embedded="true"/>
    </element>
    <element name="three">
      <type elementType="product"
        context="http://www.example.com/inventory"/>
    </element>
  </choice>
</type>
```

If a DefinedType is a DummyReference, ParameterizedType, or ParameterizedValueSetType, then the translation of the Type is the translation of that DummyReference, ParameterizedType, or ParameterizedValueSetType (see Section 13).

6.3. Translation of Built-in Types

If a Type is a BuiltinType or ReferencedType that is one of the productions in Table 1 in Section 5 of the specification for RXER [RXER], then the translation of the Type is either the attribute form or element form translation of that type.

The attribute form translation of a Type that is a BuiltinType or ReferencedType that is one of the productions in Table 1 is an attribute item with the [local name] "type". The [normalized value] of this attribute item is a qualified name for the expanded name of the built-in type (see Section 5 of the specification for RXER [RXER]).

The element form translation of a Type that is a BuiltinType or ReferencedType that is one of the productions in Table 1 is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the built-in type.

Example

BOOLEAN

```
<type ref="asn:BOOLEAN"/>
```

Usually the translator is free to choose either the attribute form or element form translation for a Type; however, in some contexts attribute forms for a Type are explicitly disallowed.

6.4. BitStringType Translation

The translation of a BitStringType with a NamedBitList is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "namedBitList" SHALL be appended to the [children] of the <type> element item. The translation of each NamedBit in the NamedBitList SHALL be appended to the [children] of the <namedBitList> element item.

The translation of a NamedBit is an element item with the [local name] "namedBit". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedBit> element item. If the BitStringType is subject to a VALUES encoding instruction, then the [normalized value] of this attribute item is the replacement name [RXEREI] for the identifier in the NamedBit; otherwise, it is the identifier in the NamedBit. If the BitStringType is subject to a VALUES encoding instruction and the reduction of the replacement name (see Section 6.1) is not the same as the identifier, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <namedBit>

element item; otherwise, an attribute item with the [local name] "identifier" MAY be added to the [attributes] of the <namedBit> element item. The [normalized value] of this attribute item is the identifier in the NamedBit. An attribute item with the [local name] "bit" SHALL be added to the [attributes] of the <namedBit> element item. The [normalized value] of this attribute item is the digit string representation of the integer value of the number or DefinedValue in the NamedBit.

Examples

```
BIT STRING { zero(0), one(1), two(2) }
```

```
<type>
  <namedBitList>
    <namedBit name="zero" bit="0"/>
    <namedBit name="one" bit="1"/>
    <namedBit name="two" bit="2"/>
  </namedBitList>
</type>
```

```
[RXER:VALUES ALL CAPITALIZED, wednesday AS "Midweek"]
  BIT STRING {
    monday(0), tuesday(1), wednesday(2),
    thursday(3), friday(4)
  }
```

```
<type>
  <namedBitList>
    <namedBit name="Monday" bit="0"/>
    <namedBit name="Tuesday" bit="1"/>
    <namedBit name="Midweek" identifier="wednesday" bit="2"/>
    <namedBit name="Thursday" bit="3"/>
    <namedBit name="Friday" bit="4"/>
  </namedBitList>
</type>
```

6.5. IntegerType Translation

The translation of an IntegerType with a NamedNumberList is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "namedNumberList" SHALL be appended to the [children] of the <type> element item. The translation of each NamedNumber in the NamedNumberList SHALL be appended to the [children] of the <namedNumberList> element item.

The translation of a NamedNumber is an element item with the [local name] "namedNumber". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <namedNumber> element item. If the IntegerType is subject to a VALUES encoding instruction, then the [normalized value] of this attribute item is the replacement name [RXEREI] for the identifier in the NamedNumber; otherwise, it is the identifier in the NamedNumber. If the IntegerType is subject to a VALUES encoding instruction and the reduction of the replacement name (see Section 6.1) is not the same as the identifier, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <namedNumber> element item; otherwise, an attribute item with the [local name] "identifier" MAY be added to the [attributes] of the <namedNumber> element item. The [normalized value] of this attribute item is the identifier in the NamedNumber. An attribute item with the [local name] "number" SHALL be added to the [attributes] of the <namedNumber> element item. The [normalized value] of this attribute item is the digit string representation of the integer value of the SignedNumber or DefinedValue in the NamedNumber.

Examples

```
INTEGER { nothing(0), a-little(1), a-lot(100) }
```

```
<type>
  <namedNumberList>
    <namedNumber name="nothing" number="0"/>
    <namedNumber name="a-little" number="1"/>
    <namedNumber name="a-lot" number="100"/>
  </namedNumberList>
</type>
```

```
[RXER:VALUES ALL CAPITALIZED, very-high AS "DANGEROUS"]
  INTEGER { low(25), medium(50), high(75), very-high(100) }
```

```
<type>
  <namedNumberList>
    <namedNumber name="Low" number="25"/>
    <namedNumber name="Medium" number="50"/>
    <namedNumber name="High" number="75"/>
    <namedNumber name="DANGEROUS" identifier="very-high"
      number="100"/>
  </namedNumberList>
</type>
```

6.6. EnumeratedType Translation

The translation of an EnumeratedType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "enumerated" SHALL be appended to the [children] of the <type> element item. The translation of each EnumerationItem nested in the RootEnumeration in the Enumerations instance in the EnumeratedType SHALL be appended to the [children] of the <enumerated> element item.

If an ellipsis ("...") is present in the Enumerations instance, then an element item with the [local name] "extension" SHALL be appended to the [children] of the <enumerated> element item and the translation of the ExceptionSpec (possibly empty) SHALL be added to the [children] of the <extension> element item. If an AdditionalEnumeration is present in the Enumerations instance, then the translation of each EnumerationItem nested in the AdditionalEnumeration SHALL be appended to the [children] of the <extension> element item.

The translation of an EnumerationItem is an element item with the [local name] "enumeration".

If the EnumerationItem is of the "identifier" form, then an attribute item with the [local name] "name" SHALL be added to the [attributes] of the <enumeration> element item. If the EnumeratedType is subject to a VALUES encoding instruction, then the [normalized value] of this attribute item is the replacement name [RXEREI] for the identifier; otherwise, it is the identifier. If the EnumeratedType is subject to a VALUES encoding instruction and the reduction of the replacement name (see Section 6.1) is not the same as the identifier, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <enumeration> element item; otherwise, an attribute item with the [local name] "identifier" MAY be added to the [attributes] of the <enumeration> element item. The [normalized value] of this attribute item is the identifier.

If the EnumerationItem is of the "NamedNumber" form, then an attribute item with the [local name] "name" SHALL be added to the [attributes] of the <enumeration> element item. If the EnumeratedType is subject to a VALUES encoding instruction, then the [normalized value] of this attribute item is the replacement name [RXEREI] for the identifier in the NamedNumber; otherwise, it is the identifier in the NamedNumber. If the EnumeratedType is subject to a VALUES encoding instruction and the reduction of the replacement name is not the same as the identifier, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the

<enumeration> element item; otherwise, an attribute item with the [local name] "identifier" MAY be added to the [attributes] of the <enumeration> element item. The [normalized value] of this attribute item is the identifier in the NamedNumber. An attribute item with the [local name] "number" SHALL be added to the [attributes] of the <enumeration> element item. The [normalized value] of this attribute item is the digit string representation of the integer value of the SignedNumber or DefinedValue in the NamedNumber.

Examples

```
ENUMERATED { red(0), green(1), ..., blue(2) }
```

```
<type>
  <enumerated>
    <enumeration name="red" number="0"/>
    <enumeration name="green" number="1"/>
    <extension>
      <enumeration name="blue" number="2"/>
    </extension>
  </enumerated>
</type>
```

```
[RXER:VALUES ALL CAPITALIZED, red AS "Crimson"]
  ENUMERATED { red, yellow, green, blue }
```

```
<type>
  <enumerated>
    <enumeration name="Crimson" identifier="red"/>
    <enumeration name="Yellow"/>
    <enumeration name="Green"/>
    <enumeration name="Blue"/>
  </enumerated>
</type>
```

6.7. PrefixedType Translation

The translation of a PrefixedType [X.680-1] that is a TaggedType is either the short form translation (Section 6.7.1) or long form translation (Section 6.7.2) of the TaggedType.

Aside: The short form translation is provided because TaggedType notation is heavily used in existing ASN.1 specifications. The long form translation has the same structure as the translation of an EncodingPrefixedType and can be simplified where there is a series of nested PrefixedType instances.

If a `PrefixedType` is an `EncodingPrefixedType` and the `EncodingReference` is `RXER`, or the `EncodingReference` is empty and the default encoding reference [X.680-1] for the module is `RXER`, then the translation of the `PrefixedType` is the translation of the `Type` in the `EncodingPrefixedType`.

Aside: This is not suggesting that `RXER` encoding instructions are ignored. Encoding instructions for `RXER` are not explicitly represented in `ASN.X`, but rather affect how an `ASN.1` module is translated into an `ASN.X` module (since the content of an `ASN.X` module is also the `RXER` encoding of an abstract value of the `ModuleDefinition` `ASN.1` type in Appendix A). The individual effects of `RXER` encoding instructions on the translation are addressed in other parts of this specification. Encoding instructions for other encoding rules have explicit representations in `ASN.X`.

If a `PrefixedType` is an `EncodingPrefixedType` and the `EncodingReference` is not `RXER`, or the `EncodingReference` is empty and the default encoding reference for the module is not `RXER`, then the translation of the `PrefixedType` is an element item with the [local name] `"prefixed"`. The translation of the `EncodingPrefix` in the `EncodingPrefixedType` SHALL be added to the [children] of the `<prefixed>` element item.

If the `EncodingReference` of an `EncodingPrefix` is not empty, then the translation of the `EncodingPrefix` is an element item with the `encodingreference` in the `EncodingReference` as the [local name]. The translation of the `EncodingInstruction` in the `EncodingPrefix` SHALL be added to the [children] of this element item.

If the `EncodingReference` of an `EncodingPrefix` is empty, then the translation of the `EncodingPrefix` is an element item with the default encoding reference for the module as the [local name]. The translation of the `EncodingInstruction` in the `EncodingPrefix` SHALL be added to the [children] of this element item.

The `EncodingInstruction` notation is different for each set of encoding instructions, and their translations into `ASN.X` are specified in separate documents [GSEREIT][XEREIT]. At the time of writing, only three sets of encoding instructions have been defined (for `RXER` [RXEREI], `GSER` [GSEREI], and `EXTENDED-XER` [X.693-1]).

If the child `<type>` element item of a `<prefixed>` element item has no attribute items and has a child `<prefixed>` element item, then that child `<type>` element item MAY be replaced by the [children] and [attributes] of the inner `<prefixed>` element item. Note that the long form translation of a `TaggedType` is also eligible for this

rewriting step. This rewriting step MAY be applied to the result of a previous rewriting step if the necessary condition still holds.

Example

These three definitions are equivalent.

```
[XER:ATTRIBUTE] [XER:USE-UNION] [GSER:CHOICE-OF-STRINGS] CHOICE {
    one PrintableString,
    two UTF8String
}
```

```
<type>
  <prefixed>
    <XER><attribute/></XER>
    <type>
      <prefixed>
        <XER><useUnion/></XER>
        <type>
          <prefixed>
            <GSER><choiceOfStrings/></GSER>
            <type>
              <choice>
                <element name="one" type="asn:PrintableString"/>
                <element name="two" type="asn:UTF8String"/>
              </choice>
            </type>
          </prefixed>
        </type>
      </prefixed>
    </type>
  </prefixed>
</type>
```

```
<type>
  <prefixed>
    <XER><attribute/></XER>
    <XER><useUnion/></XER>
    <GSER><choiceOfStrings/></GSER>
    <type>
      <choice>
        <element name="one" type="asn:PrintableString"/>
        <element name="two" type="asn:UTF8String"/>
      </choice>
    </type>
  </prefixed>
</type>
```

6.7.1. Short Form TaggedType Translation

The short form translation of a TaggedType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "tagged" SHALL be appended to the [children] of the <type> element item.

If the Class in the Tag in the TaggedType is not empty, then an attribute item with the [local name] "tagClass" SHALL be added to the [attributes] of the <tagged> element item. The [normalized value] of this attribute item is the Class of the Tag with all letters downcased, i.e., either "universal", "application", or "private".

An attribute item with the [local name] "number" SHALL be added to the [attributes] of the <tagged> element item. The [normalized value] of this attribute item is the digit string representation of the integer value of the number or DefinedValue in the ClassNumber in the Tag.

If the Tag is immediately followed by the "IMPLICIT" keyword, then an attribute item with the [local name] "tagging" and [normalized value] "implicit" SHALL be added to the [attributes] of the <tagged> element item.

If the Tag is immediately followed by the "EXPLICIT" keyword, then an attribute item with the [local name] "tagging" and [normalized value] "explicit" SHALL be added to the [attributes] of the <tagged> element item.

The translation of the Type in the TaggedType SHALL be added to the [children] or [attributes] of the <tagged> element item.

Examples

[0] INTEGER

```
<type>
  <tagged number="0" type="asn:INTEGER"/>
</type>
```

[APPLICATION 10] IMPLICIT BOOLEAN

```
<type>
  <tagged tagClass="application" number="10" tagging="implicit"
    type="asn:BOOLEAN"/>
</type>
```

6.7.2. Long Form TaggedType Translation

The long form translation of a TaggedType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "prefixed" SHALL be appended to the [children] of the <type> element item. The translation of the Tag in the TaggedType SHALL be added to the [children] of the <prefixed> element item.

The translation of a Tag is an element item with the [local name] "TAG".

If the Class of the Tag is not empty, then an attribute item with the [local name] "tagClass" SHALL be added to the [attributes] of the <TAG> element item. The [normalized value] of this attribute item is the Class of the Tag with all letters downcased, i.e., either "universal", "application", or "private".

An attribute item with the [local name] "number" SHALL be added to the [attributes] of the <TAG> element item. The [normalized value] of this attribute item is the digit string representation of the integer value of the number or DefinedValue in the ClassNumber in the Tag.

If the Tag is immediately followed by the "IMPLICIT" keyword, then an attribute item with the [local name] "tagging" and [normalized value] "implicit" SHALL be added to the [attributes] of the <TAG> element item.

If the Tag is immediately followed by the "EXPLICIT" keyword, then an attribute item with the [local name] "tagging" and [normalized value] "explicit" SHALL be added to the [attributes] of the <TAG> element item.

The translation of the Type in the TaggedType SHALL be added to the [children] or [attributes] of the <prefixed> element item.

Examples

```
[0] INTEGER

<type>
  <prefixed type="asn1:INTEGER">
    <TAG number="0"/>
  </prefixed>
</type>
```

[APPLICATION 10] IMPLICIT BOOLEAN

```
<type>
  <prefixed type="asn:BOOLEAN">
    <TAG tagClass="application" number="10" tagging="implicit"/>
  </prefixed>
</type>
```

6.8. SelectionType Translation

The translation of a SelectionType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "selection" SHALL be appended to the [children] of the <type> element item.

The identifier in a SelectionType identifies a NamedType in the definition of the Type in the SelectionType. The translation of that NamedType will be an element item with the [local name] either "attribute", "element", "component", "group", or "member". An attribute item with the same [local name] as the translation of the NamedType SHALL be added to the [attributes] of the <selection> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the NamedType [RXEREI].

The translation of the Type in the SelectionType SHALL be added to the [children] or [attributes] of the <selection> element item.

Examples

```
field1 < MyChoiceType

<type>
  <selection element="field1" type="tns:MyChoiceType"/>
</type>

field2 < CHOICE {
  field2 [RXER:ATTRIBUTE][RXER:NAME AS "field-two"] INTEGER
}

<type>
  <selection attribute="field-two">
    <type>
      <choice>
        <attribute name="field-two" identifier="field2"
          type="asn:INTEGER"/>
      </choice>
    </type>
```

```
</selection>
</type>
```

6.9. InstanceOfType Translation

The translation of an InstanceOfType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "instanceOf" SHALL be appended to the [children] of the <type> element item. The translation of the DefinedObjectClass in the InstanceOfType SHALL be added to the [children] or [attributes] of the <instanceOf> element item.

Example

INSTANCE OF TYPE-IDENTIFIER

```
<type>
  <instanceOf class="asn:TYPE-IDENTIFIER"/>
</type>
```

6.10. ObjectClassFieldType Translation

The translation of an ObjectClassFieldType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "fromClass" SHALL be appended to the [children] of the <type> element item. The translation of the DefinedObjectClass in the ObjectClassFieldType SHALL be added to the [children] or [attributes] of the <fromClass> element item. The translation of the FieldName (see Section 9.2.6) in the ObjectClassFieldType SHALL be added to the [children] or [attributes] of the <fromClass> element item.

Example

OPERATION.&Linked.&ArgumentType

```
<type>
  <fromClass class="tns:OPERATION"
            fieldName="Linked/ArgumentType"/>
</type>
```

6.11. TypeFromObject and ValueSetFromObjects Translation

The translation of a TypeFromObject or ValueSetFromObjects is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "fromObjects" SHALL be appended to the [children] of the <type> element item.

The translation of the ReferencedObjects instance in the TypeFromObject or ValueSetFromObjects SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

The translation of the FieldName in the TypeFromObject or ValueSetFromObjects SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

Example

```
invertMatrix.&Errors.&errorCode

<type>
  <fromObjects object="tns:invertMatrix"
                fieldName="Errors/errorCode"/>
</type>
```

6.12. Translation of Combining Types

This section details the translation of the ASN.1 combining types: SET, SEQUENCE, CHOICE, SET OF, and SEQUENCE OF. The combining type definitions all make use of the NamedType notation.

6.12.1. NamedType Translation

A NamedType is translated in one of three ways depending on the context. These are the normal translation, the member translation, and the item translation. These translations are not interchangeable. One of the three will be explicitly invoked as part of the translation of an enclosing combining type.

The normal translation of a NamedType is an element item with the [local name] determined as follows:

- (1) if the NamedType is subject to an ATTRIBUTE or ATTRIBUTE-REF encoding instruction, or subject to a COMPONENT-REF encoding instruction that references a top-level NamedType that is subject to an ATTRIBUTE encoding instruction, then the [local name] is "attribute",

- (2) else if the NamedType is subject to a GROUP encoding instruction, then the [local name] is "group",
- (3) else if the NamedType is subject to a SIMPLE-CONTENT encoding instruction, then the [local name] is "simpleContent",
- (4) otherwise, the [local name] is "element" or "component" (translator's choice).

Aside: The local names "element" and "component" are synonymous. The "component" alternative is offered for specifying applications that don't use RXER (except for the ASN.X specification itself, of course), where referring to parts of an encoding as elements would seem incongruous.

The member translation of a NamedType is an element item with the [local name] "member".

The item translation of a NamedType is an element item with the [local name] "item".

Aside: A Namedtype for which the member or item translation is invoked will never be subject to an ATTRIBUTE, ATTRIBUTE-REF, COMPONENT-REF, GROUP, SIMPLE-CONTENT, or TYPE-AS-VERSION encoding instruction. These encoding instructions are also mutually exclusive [RXEREI].

An element item with the [local name] "annotation" MAY be added to the [children] of the <attribute>, <element>, <component>, <group>, <item>, <member>, or <simpleContent> element item.

If a NamedType is subject to a TYPE-AS-VERSION encoding instruction, then an attribute item with the [local name] "typeAsVersion" and [normalized value] "true" or "1" SHALL be added to the <element> or <component> element item. For the normal translation, if a NamedType is not subject to an ATTRIBUTE, ATTRIBUTE-REF, COMPONENT-REF, GROUP, SIMPLE-CONTENT, or TYPE-AS-VERSION encoding instruction, then an attribute item with the [local name] "typeAsVersion" and [normalized value] "false" or "0" MAY be added to the <element> or <component> element item.

For the normal, member, and item translations, if a NamedType is not subject to an ATTRIBUTE-REF, COMPONENT-REF, ELEMENT-REF, or REF-AS-ELEMENT encoding instruction, then an attribute item with the [local name] "name" SHALL be added to the [attributes] of the <attribute>, <element>, <component>, <group>, <item>, <member>, or

<simpleContent> element item. The [normalized value] of this attribute item is the local name of the expanded name of the NamedType [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 in the NamedType.

If the reduction of the local name (an NCName) of the expanded name of a NamedType is not the same as the identifier in the NamedType, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <attribute>, <element>, <component>, <group>, <item>, <member>, or <simpleContent> element item; otherwise, an attribute item with the [local name] "identifier" MAY be added to the [attributes] of the aforementioned element item. The [normalized value] of this attribute item is the identifier in the NamedType.

Aside: The identifier attribute is not contingent on there being a name attribute. That is, an element item can have an identifier attribute item without having a name attribute item.

If a NamedType is subject to a COMPONENT-REF encoding instruction, then an attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <attribute>, <element>, or <component> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the top-level NamedType referenced by the encoding instruction. If the expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items (see Section 5.1), then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <attribute>, <element>, or <component> element item; otherwise, if the module containing the referenced top-level NamedType has a schema identity URI, then an attribute item with the [local name] "context" MAY be added to the [attributes] of the <attribute>, <element>, or <component> element item. The [normalized value] of this attribute item is the schema identity URI of the module containing the referenced top-level NamedType.

Aside: If an expanded name is not distinct, then the module containing the referenced top-level NamedType must have a schema identity URI (see Section 5.1).

If a NamedType is subject to a COMPONENT-REF encoding instruction, then an attribute item with the [local name] "embedded" and [normalized value] "false" or "0" MAY be added to the [attributes] of the <attribute>, <element>, or <component> element item.

If a NamedType is subject to an ATTRIBUTE-REF or ELEMENT-REF encoding instruction, then an attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <attribute>, <element>, or <component> element item. The [normalized value] of this attribute item is the RXER character data translation of the QNameValue in the encoding instruction. An attribute item with the [local name] "embedded" and [normalized value] "true" or "1" SHALL be added to the [attributes] of the <attribute>, <element>, or <component> element item.

If a NamedType is subject to a REF-AS-ELEMENT encoding instruction, then an attribute item with the [local name] "elementType" SHALL be added to the [attributes] of the <element> or <component> element item. The [normalized value] of this attribute item is the RXER character data translation of the NameValue in the REF-AS-ELEMENT encoding instruction. If a Namespace is present in the REF-AS-ELEMENT encoding instruction, then an attribute item with the [local name] "namespace" SHALL be added to the [attributes] of the <element> or <component> element item. The [normalized value] of this attribute item is the string value of the AnyURIValue in the Namespace.

If a ContextParameter is present in the RefParameters in the ATTRIBUTE-REF, ELEMENT-REF, or REF-AS-ELEMENT encoding instruction, then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <attribute>, <element>, or <component> element item. The [normalized value] of this attribute item is the string value of the AnyURIValue in the ContextParameter.

If a NamedType is subject to both an ATTRIBUTE encoding instruction and a VERSION-INDICATOR encoding instruction, then an attribute item with the [local name] "versionIndicator" and [normalized value] "true" or "1" SHALL be added to the <attribute> element item. If a NamedType is subject to an ATTRIBUTE encoding instruction and not subject to a VERSION-INDICATOR encoding instruction, then an attribute item with the [local name] "versionIndicator" and [normalized value] "false" or "0" MAY be added to the <attribute> element item.

If a NamedType is not subject to an ATTRIBUTE-REF, COMPONENT-REF, ELEMENT-REF, or REF-AS-ELEMENT encoding instruction, then the translation of the Type in the NamedType SHALL be added to the [children] or [attributes] of the <attribute>, <element>, <component>, <group>, <item>, <member>, or <simpleContent> element item.

If a NamedType is subject to an ATTRIBUTE-REF, COMPONENT-REF, ELEMENT-REF, or REF-AS-ELEMENT encoding instruction, then the translation of each EncodingPrefix (Section 6.7) and Tag (Section 6.7.2) textually within the NamedType SHALL be added in order to the [children] of the <attribute>, <element>, or <component> element item.

Example

```
CHOICE {
  one    INTEGER,
  two    [RXER:ATTRIBUTE] BOOLEAN,
  three  [RXER:ATTRIBUTE-REF
        { namespace-name "http://www.example.com/schema",
          local-name "foo" }]
        UTF8String,
  bar    [RXER:ELEMENT-REF
        { namespace-name "http://www.example.com/schema",
          local-name "bar" }]
        Markup,
  five   [0] [RXER:REF-AS-ELEMENT "product"
        CONTEXT "http://www.example.com/inventory"]
        Markup,
  six    [RXER:GROUP] MySequence
}

<type>
  <choice xmlns:ex="http://www.example.com/schema">
    <element name="one" type="asn:INTEGER"/>
    <attribute name="two" type="asn:BOOLEAN"/>
    <attribute ref="ex:foo" identifier="three" embedded="true"/>
    <element ref="ex:bar" embedded="true"/>
    <element elementType="product"
      context="http://www.example.com/inventory"
      identifier="five">
      <TAG number="0"/>
    </element>
    <group name="six" type="tns:MySequence"/>
  </choice>
</type>
```

6.12.2. SequenceType Translation

The translation of a SequenceType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "sequence" SHALL be appended to the [children] of the <type> element item. The

translation of each `ComponentType` nested in the `ComponentTypeList` in the initial `RootComponentTypeList`, if present, SHALL be appended to the `[children]` of the `<sequence>` element item.

If an `ExtensionAndException` is present, then an element item with the `[local name]` "extension" SHALL be appended to the `[children]` of the `<sequence>` element item. If an `ExceptionSpec` is present in the `ExtensionAndException`, then the translation of the `ExceptionSpec` (possibly empty) SHALL be added to the `[children]` of the `<extension>` element item.

If an `ExtensionAdditions` instance is present, then the translation of each `ExtensionAdditionGroup` or `ComponentType` nested in the `ExtensionAdditions` (if any) SHALL be appended to the `[children]` of the `<extension>` element item.

If an `ExtensionEndMarker` is present, then the translation of each `ComponentType` nested in the `ComponentTypeList` in the final `RootComponentTypeList` SHALL be appended to the `[children]` of the `<sequence>` element item.

The translation of an `ExtensionAdditionGroup` is an element item with the `[local name]` "extensionGroup". If the `VersionNumber` in the `ExtensionAdditionGroup` is not empty, then an attribute item with the `[local name]` "version" SHALL be added to the `[attributes]` of the `<extensionGroup>` element item. The `[normalized value]` of this attribute item is the number in the `VersionNumber`. The translation of each `ComponentType` nested in the `ExtensionAdditionGroup` SHALL be appended to the `[children]` of the `<extensionGroup>` element item.

The translation of a `ComponentType` of the "NamedType" form is the normal translation of the `NamedType`.

The translation of a `ComponentType` of the "NamedType OPTIONAL" form is an element item with the `[local name]` "optional". The normal translation of the `NamedType` SHALL be added to the `[children]` of the `<optional>` element item.

The translation of a `ComponentType` of the "NamedType DEFAULT Value" form is an element item with the `[local name]` "optional". The normal translation of the `NamedType` SHALL be added to the `[children]` of the `<optional>` element item. An element item with the `[local name]` "default" SHALL be appended to the `[children]` of the `<optional>` element item. The translation of the `Value` SHALL be added to the `[children]` or `[attributes]` of the `<default>` element item.

The translation of a `ComponentType` of the "COMPONENTS OF Type" form is an element item with the [local name] "componentsOf". The translation of the Type SHALL be added to the [children] or [attributes] of the <componentsOf> element item.

Example

```
SEQUENCE {
    one    INTEGER,
    two    [RXER:ATTRIBUTE] BOOLEAN OPTIONAL,
    ...,
    [[ 2:
        four    NULL
    ]],
    COMPONENTS OF MySequence,
    ...,
    three    PrintableString DEFAULT "third"
}

<type>
<sequence>
  <element name="one" type="asn:INTEGER"/>
  <optional>
    <attribute name="two" type="asn:BOOLEAN"/>
  </optional>
  <extension>
    <extensionGroup version="2">
      <element name="four" type="asn:NULL"/>
    </extensionGroup>
    <componentsOf type="tns:MySequence"/>
  </extension>
  <optional>
    <element name="three" type="asn:PrintableString"/>
    <default literalValue="third"/>
  </optional>
</sequence>
</type>
```

6.12.3. SetType Translation

The translation of a `SetType` follows the same procedure as the translation of a `SequenceType` except that `SetType` replaces `SequenceType`, "SET" replaces "SEQUENCE", and the [local name] "set" is used instead of "sequence".

6.12.4. ChoiceType Translation

The translation of a ChoiceType that is not subject to a UNION encoding instruction is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "choice" SHALL be appended to the [children] of the <type> element item. The normal translation of each NamedType nested in the AlternativeTypeList in the RootAlternativeTypeList in the AlternativeTypeLists SHALL be appended to the [children] of the <choice> element item.

If an ExtensionAndException is present in the AlternativeTypeLists, then an element item with the [local name] "extension" SHALL be appended to the [children] of the <choice> element item. If an ExceptionSpec is present in the ExtensionAndException, then the translation of the ExceptionSpec (possibly empty) is added to the [children] of the <extension> element item.

If an ExtensionAdditionAlternatives instance is present in the AlternativeTypeLists, then the translation of each ExtensionAdditionAlternativesGroup or NamedType (if any) nested in the ExtensionAdditionAlternatives SHALL be appended in order to the [children] of the <extension> element item. The normal translation of the NamedType is used.

The translation of an ExtensionAdditionAlternativesGroup is an element item with the [local name] "extensionGroup". If the VersionNumber in the ExtensionAdditionAlternativesGroup is not empty, then an attribute item with the [local name] "version" SHALL be added to the [attributes] of the <extensionGroup> element item. The [normalized value] of this attribute item is the number in the VersionNumber. The normal translation of each NamedType nested in the AlternativeTypeList in the ExtensionAdditionAlternativesGroup SHALL be appended to the [children] of the <extensionGroup> element item.

Example

```

CHOICE {
    one  INTEGER,
    two  [RXER:NAME AS "Two"] BOOLEAN,
    ...
    [[ 2:
        three  NULL
    ]],
    four  PrintableString,
    ...
}

<type>
<choice>
  <element name="one" type="asn:INTEGER"/>
  <element name="Two" type="asn:BOOLEAN"/>
  <extension>
    <extensionGroup version="2">
      <element name="three" type="asn:NULL"/>
    </extensionGroup>
    <element name="four" type="asn:PrintableString"/>
  </extension>
</choice>
</type>

```

6.12.5. Translation of UNION Types

The translation of a ChoiceType that is subject to a UNION encoding instruction follows the same procedure as the translation of a ChoiceType that is not subject to a UNION encoding instruction except that the [local name] "union" is used instead of "choice", and the member translation of each NamedType is used instead of the normal translation.

In addition, if the UNION encoding instruction has a PrecedenceList, then an attribute item with the [local name] "precedence" SHALL be added to the [attributes] of the <union> element item. The [normalized value] of this attribute item is the white space separated list of qualified names for the expanded names of the NamedType instances [RXEREI] corresponding to the identifiers in the PrecedenceList. A white space separator is one or more of the white space characters.

Example

```

[RXER:UNION PRECEDENCE utf8 visible] CHOICE {
    printable PrintableString,
    teletex    TeletexString,
    visible    [RXER:NAME AS "ascii"] VisibleString,
    ...,
    utf8       UTF8String
}

<type>
  <union precedence="utf8 ascii">
    <member name="printable" type="asn:PrintableString"/>
    <member name="teletex" type="asn:TeletexString"/>
    <member name="ascii" identifier="visible"
        type="asn:VisibleString"/>
  <extension>
    <member name="utf8" type="asn:UTF8String"/>
  </extension>
</union>
</type>

```

6.12.6. SequenceOfType Translation

The translation of a SequenceOfType that is not subject to a LIST encoding instruction is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "sequenceOf" SHALL be appended to the [children] of the <type> element item.

If the SequenceOfType is of the "SEQUENCE OF NamedType" form, then the normal translation of the NamedType SHALL be added to the [children] of the <sequenceOf> element item.

If the SequenceOfType is of the "SEQUENCE OF Type" form, then an element item with the [local name] "element" or "component" (translator's choice) SHALL be added to the [children] of the <sequenceOf> element item. An attribute item with the [local name] "name" and [normalized value] "item" SHALL be added to the [attributes] of the <element> or <component> element item. An attribute item with the [local name] "identifier" and empty [normalized value] SHALL be added to the [attributes] of the <element> or <component> element item. The translation of the Type SHALL be added to the [children] or [attributes] of the <element> or <component> element item.

Examples

SEQUENCE OF INTEGER

```
<type>
  <sequenceOf>
    <element name="item" identifier="" type="asn:INTEGER"/>
  </sequenceOf>
</type>
```

SEQUENCE OF counter INTEGER

```
<type>
  <sequenceOf>
    <element name="counter" type="asn:INTEGER"/>
  </sequenceOf>
</type>
```

6.12.7. Translation of LIST Types

The translation of a SequenceOfType that is subject to a LIST encoding instruction is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "list" SHALL be appended to the [children] of the <type> element item. The item translation of the NamedType in the SequenceOfType SHALL be added to the [children] of the <list> element item.

Aside: A SequenceOfType is necessarily of the "SEQUENCE OF NamedType" form for a LIST encoding instruction.

Example

[RXER:LIST] SEQUENCE OF number INTEGER

```
<type>
  <list>
    <item name="number" type="asn:INTEGER"/>
  </list>
</type>
```

6.12.8. SetOfType Translation

The translation of a SetOfType follows the same procedure as the translation of a SequenceOfType except that SetOfType replaces SequenceOfType, "SET" replaces "SEQUENCE", and the [local name] "setOf" is used instead of "sequenceOf".

6.12.9. Effect of Insertion Encoding Instructions

If a Type is subject to a NO-INSERTIONS, HOLLOW-INSERTIONS, SINGULAR-INSERTIONS, UNIFORM-INSERTIONS, or MULTIFORM-INSERTIONS encoding instruction, then an attribute item with the [local name] "insertions" SHALL be added to the [attributes] of the <choice>, <sequence> or <set> element item in the [children] of the <type> element item resulting from the translation of the Type. The [normalized value] of this attribute item is "none" in the case of a NO-INSERTIONS encoding instruction, "hollow" in the case of a HOLLOW-INSERTIONS encoding instruction, "singular" in the case of a SINGULAR-INSERTIONS encoding instruction, "uniform" in the case of a UNIFORM-INSERTIONS encoding instruction, and "multiform" in the case of a MULTIFORM-INSERTIONS encoding instruction.

Example

```
[NO-INSERTIONS] CHOICE {
    one [RXER:GROUP] [RXER:SINGULAR-INSERTIONS] CHOICE {
        two INTEGER,
        ...
    },
    ...
}

<type>
<choice insertions="none">
  <group name="one">
    <type>
      <choice insertions="singular">
        <element name="two" type="asn:INTEGER"/>
        <extension/>
      </choice>
    </type>
  </group>
  <extension/>
</choice>
</type>
```

6.13. Translation of Constrained Types

If a ConstrainedType is of the "Type Constraint" form, then the translation of the ConstrainedType is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item. An element item with the [local name] "constrained" SHALL be appended to the [children] of the <type> element item. The translation of the Type SHALL be added to the [children] or

[attributes] of the <constrained> element item. The translation of the Constraint SHALL be added to the [children] of the <constrained> element item.

The translation of a ContainedType that is a TypeWithConstraint is the translation of the TypeWithConstraint.

Definition (simple endpoint): A LowerEndpoint or UpperEndpoint is a simple endpoint if it is closed and its value is "MIN", "MAX", or a SignedNumber in an IntegerValue in a BuiltinValue in the Value of the endpoint.

Definition (simple range SizeConstraint): A SizeConstraint is a simple range if the Constraint in the SizeConstraint contains only a ValueRange (i.e., a ValueRange in a SubtypeElements instance in an Elements instance in a lone IntersectionElements instance in a lone Intersections instance in a Unions instance in an ElementSetSpec in a RootElementSetSpec in an ElementSetSpecs instance without an AdditionalElementSetSpec in a SubtypeConstraint in a ConstraintSpec in the Constraint) and both endpoints are simple.

Definition (simple range Constraint): A Constraint is a simple range if contains only a SizeConstraint that is a simple range (i.e., a simple range SizeConstraint in a SubtypeElements instance in an Elements instance in a lone IntersectionElements instance in a lone Intersections instance in a Unions instance in an ElementSetSpec in a RootElementSetSpec in an ElementSetSpecs instance without an AdditionalElementSetSpec in a SubtypeConstraint in a ConstraintSpec in the Constraint).

If the Constraint or SizeConstraint in a TypeWithConstraint is a simple range, then the compact translation of the TypeWithConstraint MAY be used; otherwise, the full translation of the TypeWithConstraint is used.

The compact translation of a TypeWithConstraint is initially the translation of its notional parent type. If the value of the lower endpoint is not "MIN" or "0", then an attribute item with the [local name] "minSize" SHALL be added to the [attributes] of the <sequenceOf>, <setOf>, or <list> element item resulting from the translation of the parent type. The [normalized value] of this attribute item is the value of the lower endpoint. If the value of the lower endpoint is "MIN" or "0", then an attribute item with the [local name] "minSize" and [normalized value] "0" MAY be added to the [attributes] of the <sequenceOf>, <setOf>, or <list> element item. If the value of the upper endpoint is not "MAX", then an attribute item with the [local name] "maxSize" SHALL be added to the [attributes] of the <sequenceOf>, <setOf>, or <list> element item.

The [normalized value] of this attribute item is the value of the upper endpoint.

The full translation of a `TypeWithConstraint` is an element item with the [local name] "type". An element item with the [local name] "annotation" MAY be added to the [children] of the `<type>` element item. An element item with the [local name] "constrained" SHALL be appended to the [children] of the `<type>` element item. The translation of the notional parent type of the `TypeWithConstraint` SHALL be added to the [children] or [attributes] of the `<constrained>` element item. The translation of the `Constraint` or `SizeConstraint` in the `TypeWithConstraint` SHALL be added to the [children] of the `<constrained>` element item.

Examples

SEQUENCE (SIZE(1..MAX)) OF number INTEGER

```
<type>
  <sequenceOf minSize="1">
    <element name="number" type="asn:INTEGER"/>
  </sequenceOf>
</type>
```

SEQUENCE SIZE(0..10) OF number INTEGER

```
<type>
  <sequenceOf maxSize="10">
    <element name="number" type="asn:INTEGER"/>
  </sequenceOf>
</type>
```

SEQUENCE SIZE(1..limit) OF number INTEGER

```
<type>
  <constrained>
    <type>
      <sequenceOf>
        <element name="number" type="asn:INTEGER"/>
      </sequenceOf>
    </type>
    <size>
      <range>
        <minInclusive literalValue="1"/>
        <maxInclusive value="tns:limit"/>
      </range>
    </size>
  </constrained>
</type>
```

```
    </size>
  </constrained>
</type>
```

6.13.1. Constraint Translation

The translation of a Constraint is the translation of the ConstraintSpec in the Constraint followed by the translation of the ExceptionSpec (possibly empty) in the Constraint.

The translation of a ConstraintSpec is the translation of the SubtypeConstraint or GeneralConstraint in the ConstraintSpec.

The translation of a SubtypeConstraint is the translation of the ElementSetSpecs in the SubtypeConstraint.

The translation of a GeneralConstraint [X.682] is the translation of the UserDefinedConstraint, TableConstraint, or ContentsConstraint in the GeneralConstraint.

6.13.2. UserDefinedConstraint Translation

The translation of a UserDefinedConstraint is an element item with the [local name] "constrainedBy". An element item with the [local name] "annotation" MAY be added to the [children] of the <constrainedBy> element item. The translation of each UserDefinedConstraintParameter in the UserDefinedConstraint SHALL be appended to the [children] of the <constrainedBy> element item.

The translation of a UserDefinedConstraintParameter of the "Governor : Value" form is an element item with the [local name] "valueParameter". The translation of the Type in the Governor SHALL be added to the [children] or [attributes] of the <valueParameter> element item. The translation of the Value SHALL be added to the [children] or [attributes] of the <valueParameter> element item.

The translation of a UserDefinedConstraintParameter of the "Governor : ValueSet" form is an element item with the [local name] "valueSetParameter". The translation of the Type in the Governor SHALL be added to the [children] or [attributes] of the <valueSetParameter> element item. The translation of the ValueSet SHALL be added to the [children] of the <valueSetParameter> element item.

The translation of a UserDefinedConstraintParameter of the "Governor : Object" form is an element item with the [local name] "objectParameter". The translation of the DefinedObjectClass in the Governor SHALL be added to the [children] or [attributes] of the

<objectParameter> element item. The translation of the Object SHALL be added to the [children] or [attributes] of the <objectParameter> element item.

The translation of a UserDefinedConstraintParameter of the "Governor : ObjectSet" form is an element item with the [local name] "objectSetParameter". The translation of the DefinedObjectClass in the Governor SHALL be added to the [children] or [attributes] of the <objectSetParameter> element item. The translation of the ObjectSet SHALL be added to the [children] or [attributes] of the <objectSetParameter> element item.

The translation of a UserDefinedConstraintParameter that is a Type is an element item with the [local name] "typeParameter". The translation of the Type SHALL be added to the [children] or [attributes] of the <typeParameter> element item.

The translation of a UserDefinedConstraintParameter that is a DefinedObjectClass is an element item with the [local name] "classParameter". The translation of the DefinedObjectClass SHALL be added to the [children] or [attributes] of the <classParameter> element item.

Example

```
OCTET STRING
  (CONSTRAINED BY {
    -- contains the hash of the value -- MyType:myValue })

<type>
  <constrained type="asn:OCTET-STRING">
    <constrainedBy>
      <annotation> contains the hash of the value </annotation>
      <valueParameter type="tns:MyType" value="tns:myValue"/>
    </constrainedBy>
  </constrained>
</type>
```

6.13.3. TableConstraint Translation

The translation of a TableConstraint that is a SimpleTableConstraint is an element item with the [local name] "table". The translation of the ObjectSet in the SimpleTableConstraint SHALL be added to the [children] or [attributes] of the <table> element item.

The translation of a TableConstraint that is a ComponentRelationConstraint is an element item with the [local name] "table". The translation of the DefinedObjectSet in the

ComponentRelationConstraint SHALL be added to the [children] or [attributes] of the <table> element item. The translation of each AtNotation in the ComponentRelationConstraint SHALL be appended to the [children] of the <table> element item.

The translation of an AtNotation is an element item with the [local name] "restrictBy". The [children] property of the <restrictBy> element item is set to the sequence of character items for the character string formed by the concatenation of zero, one, or more "../" strings, one for each Level in the AtNotation (including the empty one), followed by a solidus ('/', U+002F) separated list of qualified names for the expanded names of the NamedType instances [RXEREI] identified by the identifiers in the ComponentIdList in the AtNotation. If a NamedType is subject to an ATTRIBUTE or ATTRIBUTE-REF encoding instruction, or subject to a COMPONENT-REF encoding instruction that references a top-level NamedType that is subject to an ATTRIBUTE encoding instruction, then the qualified name for the expanded name is prefixed with the commercial at character ('@', U+0040). Leading and/or trailing white space character items MAY be added to the [children] of the <restrictBy> element item. White space character items MAY be added immediately before and/or after any character item for the solidus character ('/', U+002F).

Examples

```
ERROR.&Type({Errors}){@severity,@...errorId})
```

```
<type>
  <constrained>
    <type>
      <fromClass class="tns:ERROR" fieldName="Type"/>
    </type>
    <table objectset="tns:Errors">
      <restrictBy>severity</restrictBy>
      <restrictBy>../../../../errorId</restrictBy>
    </table>
  </constrained>
</type>

SEQUENCE {
  id-att  [RXER:NAME AS "ID"] [RXER:ATTRIBUTE]
          TYPE-IDENTIFIER.&id({AllTypes}),
  value   TYPE-IDENTIFIER.&Type({AllTypes}){@id-att})
}

<type>
  <sequence>
    <attribute name="ID" identifier="id-att">
```



```

    <type>
      <constrained>
        <type>
          <fromClass class="asn:TYPE-IDENTIFIER" fieldName="id"/>
        </type>
        <table objectset="tns:AllTypes"/>
      </constrained>
    </type>
  </attribute>
  <element name="value">
    <type>
      <constrained>
        <type>
          <fromClass class="asn:TYPE-IDENTIFIER" fieldName="Type"/>
        </type>
        <table objectset="tns:AllTypes">
          <restrictBy>@ID</restrictBy>
        </table>
      </constrained>
    </type>
  </element>
</sequence>
</type>

```

The <restrictBy> element item is required to be self-contained [RXER].

Aside: An element item is self-contained if all namespace prefixes used by the element item and its contents are declared within the element item.

6.13.4. ContentsConstraint Translation

The translation of a ContentsConstraint is an element item with the [local name] "contents".

If the ContentsConstraint is of the "CONTAINING Type" form, then an element item with the [local name] "containing" SHALL be added to the [children] of the <contents> element item. The translation of the Type SHALL be added to the [children] or [attributes] of the <containing> element item.

If the ContentsConstraint is of the "ENCODED BY Value" form, then an element item with the [local name] "encodedBy" SHALL be added to the [children] of the <contents> element item. The translation of the Value SHALL be added to the [children] or [attributes] of the <encodedBy> element item.

If the ContentsConstraint is of the "CONTAINING Type ENCODED BY Value" form, then an element item with the [local name] "containing" and an element item with the [local name] "encodedBy" SHALL be added to the [children] of the <contents> element item. The translation of the Type SHALL be added to the [children] or [attributes] of the <containing> element item. The translation of the Value SHALL be added to the [children] or [attributes] of the <encodedBy> element item.

Example

```
OCTET STRING
  (CONTAINING MyType
   ENCODED BY { joint-iso-itu-t asn1(1) basic-encoding(1) })

<type>
  <constrained type="asn:OCTET-STRING">
    <contents>
      <containing type="tns:MyType"/>
      <encodedBy literalValue="2.1.1"/>
    </contents>
  </constrained>
</type>
```

6.13.5. ExceptionSpec Translation

The translation of an empty ExceptionSpec is empty.

The translation of a non-empty ExceptionSpec is an element item with the [local name] "exception".

If the ExceptionIdentification in a non-empty ExceptionSpec is a SignedNumber, then the translation of a notional INTEGER Type SHALL be added to the [children] or [attributes] of the <exception> element item, and the translation of a notional Value of the INTEGER type with the SignedNumber as its IntegerValue SHALL be added to the [children] or [attributes] of the <exception> element item.

If the ExceptionIdentification in a non-empty ExceptionSpec is a DefinedValue, then the translation of a notional INTEGER Type SHALL be added to the [children] or [attributes] of the <exception> element item, and the translation of the DefinedValue SHALL be added to the [children] or [attributes] of the <exception> element item.

If the `ExceptionIdentification` in a non-empty `ExceptionSpec` is of the "Type : Value" form, then the translation of the Type SHALL be added to the [children] or [attributes] of the <exception> element item, and the translation of the Value SHALL be added to the [children] or [attributes] of the <exception> element item.

Examples

```
!10
```

```
<exception type="asn:INTEGER" literalValue="10"/>
```

```
!myValue
```

```
<exception type="asn:INTEGER" value="tns:myValue"/>
```

```
!PrintableString:"failure"
```

```
<exception type="asn:PrintableString" literalValue="failure"/>
```

7. Translation of Values

A Value in an ASN.1 specification is a mix of literal values (e.g., numbers and character strings) and notations for referencing defined values. Likewise, the ASN.X translation of a Value is a mix of markup for literal values and markup for referencing notations (notational values). A Value is categorized by the following definitions.

Definition (literal value): A Value is a literal value if and only if it is not a notational value.

Definition (notational value): A Value is a notational value if and only if:

- (1) the Value is a `BuiltinValue`, and
 - (a) the `BuiltinValue` is a `TaggedValue` and the Value in the `TaggedValue` is a notational value, or
 - (b) the `BuiltinValue` is a `SequenceValue` or `SetValue` with a `ComponentValueList` that contains a `NamedValue` where the Value in the `NamedValue` is a notational value and the translation of the corresponding `NamedType` (from the governing type of the outer Value) is not an <element> or <component> element item, or

- (c) the BuiltinValue is a ChoiceValue where the Value of the ChoiceValue is a notational value and the translation of the NamedType corresponding to the identifier in the ChoiceValue is not an <element> or <component> element item, or
 - (d) the BuiltinValue is a SequenceOfValue or SetOfValue with a NamedValueList that contains a NamedValue where the Value of the NamedValue is a notational value and the translation of the corresponding NamedType (from the governing type of the outer Value) is not an <element> or <component> element item, or
- (2) the Value is a ReferencedValue, and
- (a) the ReferencedValue is a ValueFromObject, or
 - (b) the ReferencedValue is a DefinedValue, and
 - (i) the DefinedValue is a valuereference (not a DummyReference) or an ExternalValueReference, or
 - (ii) the DefinedValue is a DummyReference or ParameterizedValue and the substitute definition for the DummyReference or ParameterizedValue (see Section 13) is a notational value, or
 - (iii) the DefinedValue is a DummyReference or ParameterizedValue where the translation of the DummyReference or ParameterizedValue will use a fully expanded reference (see Section 13), or
- (3) the Value is an ObjectClassFieldValue, and
- (a) the ObjectClassFieldValue is an OpenTypeFieldVal, or
 - (b) the ObjectClassFieldValue is a FixedTypeFieldVal, and
 - (i) the FixedTypeFieldVal is a BuiltinValue that satisfies case (1), or
 - (ii) the FixedTypeFieldVal is a ReferencedValue that satisfies case (2).

A literal value that is a BuiltinValue that is a SequenceValue, SetValue, ChoiceValue, SequenceOfValue, or SetOfValue MAY be translated as a notational value.

Definition (directly nested): A notational value is directly nested (within a literal value) if the innermost enclosing Value is a literal value.

7.1. Translation of Literal Values

The translation of a literal value is either the attribute form translation of a literal value, or the element form translation of a literal value.

The attribute form translation of a literal value is an attribute item with the [local name] "literalValue". The [normalized value] of this attribute item is the RXER character data translation [RXER] of the literal value.

The attribute form translation of a literal value SHALL NOT be used if:

- (1) the RXER Infoset translation of the literal value is not a character data translation [RXER] or is a character data translation that contains qualified names [XMLNS10][XMLNS11], or
- (2) attribute form translations of Value have been explicitly disallowed in the context where the literal value appears, or
- (3) the literal value has a nested notational value.

The element form translation of a literal value is an element item with the [local name] "literalValue". The [children] and [attributes] of the <literalValue> element item are set to the RXER Infoset translation of the literal value, except that a value of the EXTERNAL type (or a subtype thereof) is translated according to the associated type defined in Clause 34.5 of X.680 [X.680]. In addition, where the [children] and [attributes] of an element item in the translation correspond to a directly nested notational value, the translation specified in Section 7.2 MUST be used for the [children] and [attributes] of that element item, and an attribute item with the [local name] "literal", [namespace name] "urn:ietf:params:xml:ns:asn", and [normalized value] "false" or "0" (i.e., `asn:literal="false"`) MUST be added to the [attributes] of that element item.

Each outermost <literalValue> element item is required to be self-contained [RXER].

Aside: An element item is self-contained if all namespace prefixes used by the element item and its contents are declared within the element item.

Aside: A `<literalValue>` element item nested within another `<literalValue>` element item is not required to be self-contained.

An attribute item with the [local name] "literal", [namespace name] "urn:ietf:params:xml:ns:asn1" and [normalized value] "true" or "1" (i.e., `asn1:literal="true"`) MAY be added to the [attributes] of the `<literalValue>` element item and/or any nested element item with content and attributes that correspond to a literal value.

Aside: The `asn1:literal` attribute operates as a switch that indicates whether the content and other attributes of the element containing the attribute are interpreted as ASN.1 notation (a notational value) or as an RXER encoding (a literal value).

Example

```
zero INTEGER ::= 0
```

```
<namedValue name="zero" type="asn1:INTEGER" literalValue="0"/>
```

OR

```
<namedValue name="zero" type="asn1:INTEGER">  
  <literalValue>0</literalValue>  
</namedValue>
```

From the perspective of an ASN.1 module as the RXER encoding of an ASN.1 value (an abstract value of the `ModuleDefinition` type in Appendix A), the type of the `<literalValue>` element is the unconstrained Markup type [RXER], not the governing type of the Value according to the ASN.1 specification. This means that the Infoset representation of the `<literalValue>` element must be preserved in re-encodings of the ASN.1 module.

Similarly, the type of the `literalValue` attribute is a `UTF8String`, not the governing type of the Value according to the ASN.1 specification. This means that the exact characters of the [normalized value] of the attribute must be preserved in re-encodings of the ASN.1 module.

7.2. Translation of Notational Values

The translation of a notational value is the translation of the `BuiltinValue`, `ReferencedValue`, or `ObjectClassFieldValue` in the notational value.

The translation of a `ReferencedValue` is the translation of the `DefinedValue` or `ValueFromObject` in the `ReferencedValue`.

The translation for each of these cases is described as creating an element item with the [local name] "value", which is appropriate for a notational value that stands on its own. However, a notational value may also be directly nested within a literal value, in which case the [local name] will be determined according to RXER and the governing ASN.1 type of the enclosing literal value.

Aside: In the latter case, the element item will also have a literal attribute item with the [normalized value] "false" or "0".

A notational value that is not directly nested within a literal value MAY instead have the [local name] "literalValue" provided an attribute item with the [local name] "literal", [namespace name] "urn:ietf:params:xml:ns:asn1", and [normalized value] "false" or "0" is added to the [attributes] of the <literalValue> element item.

Examples

```
nothing INTEGER ::= zero
```

```
<namedValue name="nothing" type="asn1:INTEGER" value="tns:zero"/>
```

OR

```
<namedValue name="nothing" type="asn1:INTEGER">
  <value ref="tns:zero"/><!-- A notational value. -->
</namedValue>
```

OR

```
<namedValue name="nothing" type="asn1:INTEGER">
  <literalValue xmlns:asn1="urn:ietf:params:xml:ns:asn1"
    xmlns:tns="http://example.com/ns/MyModule"
    asn1:literal="false"
    ref="tns:zero"/><!-- A notational value. -->
</namedValue>
```

```
integerList SEQUENCE OF number INTEGER ::= { zero, 3, 7 }
```

```
<namedValue name="integerList">
  <type>
    <sequenceOf>
      <element name="number" type="asn1:INTEGER"/>
    </sequenceOf>
  </type>
  <literalValue xmlns:asn1="urn:ietf:params:xml:ns:asn1"
    xmlns:tns="http://example.com/ns/MyModule">
    <number asn1:literal="false">
```

```

        ref="tns:zero"/><!-- A notational value. -->
    <number>3</number><!-- A literal value. -->
    <number>7</number><!-- A literal value. -->
</literalValue>
</namedValue>

```

7.2.1. DefinedValue Translation

If a DefinedValue is a valuereference (not a DummyReference) or an ExternalValueReference, then the translation of the DefinedValue is either the attribute form translation of a value reference, or the element form translation of a value reference.

The attribute form translation of a value reference is an attribute item with the [local name] "value". The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced value definition (see Section 5.1). The attribute form translation SHALL NOT be used if this expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items (see Section 5.1).

The element form translation of a value reference is an element item with the [local name] "value". An element item with the [local name] "annotation" MAY be added to the [children] of the <value> element item. An attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <value> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced value definition. If this expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items, then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <value> element item; otherwise, if the module containing the referenced value definition has a schema identity URI, then an attribute item with the [local name] "context" MAY be added to the [attributes] of the <value> element item. The [normalized value] of this attribute item is the schema identity URI of the module containing the value definition referenced by the DefinedValue.

Aside: If a reference name is not distinct, then the module containing the referenced definition must have a schema identity URI (see Section 5.1).

Usually the translator is free to choose either an attribute form or element form translation for a DefinedValue; however, in some contexts attribute forms of Value are explicitly disallowed. In

particular, the attribute form translation SHALL NOT be used for a DefinedValue in a ReferencedValue in a Value that is directly nested in a literal value.

If a DefinedValue is a DummyReference or ParameterizedValue, then the translation of the DefinedValue is the translation of that DummyReference or ParameterizedValue (see Section 13).

7.2.2. BuiltinValue Translation

The translation of a BuiltinValue is the translation of the ChoiceValue, SequenceValue, SetValue, SequenceOfValue, SetOfValue, or TaggedValue in the BuiltinValue.

Aside: There are other possibilities for a BuiltinValue, but these will all be literal values. This section applies to a BuiltinValue that is a notational value.

The translation of a TaggedValue is the translation of the Value in the TaggedValue (which is necessarily a notational value).

The translation of a ChoiceValue is an element item with the [local name] "value". An element item with the [local name] "annotation" MAY be added to the [children] of the <value> element item. An element item with the same [local name] (i.e., "attribute", "element", "component", "group", or "member") as the translation of the NamedType corresponding to the identifier in the ChoiceValue SHALL be appended to the [children] of the <value> element item. An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <attribute>, <element>, <component>, <group>, or <member> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the NamedType. The translation of the Value in the ChoiceValue SHALL be added to the [children] or [attributes] of the <attribute>, <element>, <component>, <group>, or <member> element item.

The translation of a SequenceValue or SetValue is an element item with the [local name] "value". An element item with the [local name] "annotation" MAY be added to the [children] of the <value> element item. If the SequenceValue or SetValue has a ComponentValueList, then the translation of each NamedValue nested in the ComponentValueList SHALL be appended to the [children] of the <value> element item in the order in which their corresponding NamedType instances appear in the definition of the governing type.

The translation of a `SequenceOfValue` or `SetOfValue` is an element item with the [local name] "value". An element item with the [local name] "annotation" MAY be added to the [children] of the <value> element item.

If the `SequenceOfValue` or `SetOfValue` has a `NamedValueList`, then the translation of each `NamedValue` nested in the `NamedValueList` SHALL be appended to the [children] of the <value> element item.

If the `SequenceOfValue` or `SetOfValue` has a `ValueList`, then an element item with the same [local name] (i.e., "element" or "component") as the element item in the [children] of the <sequenceOf> or <setOf> element item in the translation of the governing type SHALL be appended to the [children] of the <value> element item for each `Value` nested in the `ValueList`. An attribute item with the [local name] "name" and [normalized value] "item" SHALL be added to the [attributes] of the <element> or <component> element item. The translation of the `Value` (from the `ValueList`) SHALL be added to the [children] or [attributes] of the <element> or <component> element item.

The translation of a `NamedValue` is an element item with the same [local name] as the translation of the corresponding `NamedType`, i.e., "attribute", "element", "component", "group", "item", or "simpleContent". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the `NamedType`. The translation of the `Value` in the `NamedValue` SHALL be added to the [children] or [attributes] of the element item.

Examples

```
-- This is the governing type.
MyType ::= SEQUENCE {
    one      [ATTRIBUTE] INTEGER,
    two      INTEGER,
    three    [ATTRIBUTE][LIST] SEQUENCE OF number INTEGER
}

<namedType name="MyType">
  <type>
    <sequence>
      <attribute name="one" type="asn:INTEGER"/>
      <element name="two" type="asn:INTEGER"/>
      <attribute name="three">
        <type>
          <list>
```

```

        <item name="number" type="asn:INTEGER"/>
      </list>
    </type>
  </attribute>
</sequence>
</type>
</namedType>

myValue1 MyType ::= {
  one      456,
  two      123,
  three    { number 123, number 456 }
}
-- All literal values.

<namedValue name="myValue1" type="tns:MyType">
  <literalValue one="456" three="123 456">
    <two>123</two>
  </literalValue>
</namedValue>

myValue2 MyType ::= {
  one      456,
  two      myObject.&number,
  -- only the value for component "two" is a notational value
  three    { number 123, number 456 }
}

<namedValue name="myValue2" type="tns:MyType">
  <literalValue xmlns:asn="urn:ietf:params:xml:ns:asn"
    xmlns:tns="http://example.com/ns/MyModule"
    one="456" three="123 456">
    <two asn:literal="false">
      <fromObjects object="tns:myObject" fieldName="number"/>
    </two>
  </literalValue>
</namedValue>

myValue3 MyType ::= {
  one      myObject.&number,
  two      123,
  three    { number 123, number myObject.&number }
}

<namedValue name="myValue3" type="tns:MyType">
  <value>
    <attribute name="one">
      <value>

```

```

    <fromObjects object="tns:myObject" fieldName="number"/>
  </value>
</attribute>
<element name="two" literalValue="123"/>
<attribute name="three">
  <value>
    <item name="number" literalValue="123"/>
    <item name="number">
      <value>
        <fromObjects object="tns:myObject" fieldName="number"/>
      </value>
    </item>
  </value>
</attribute>
</value>
</namedValue>

```

7.2.3. ValueFromObject Translation

The translation of a ValueFromObject is an element item with the [local name] "value". An element item with the [local name] "annotation" MAY be added to the [children] of the <value> element item. An element item with the [local name] "fromObjects" SHALL be appended to the [children] of the <value> element item.

The translation of the ReferencedObjects instance in the ValueFromObject SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

The translation of the FieldName in the ValueFromObject SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

7.2.4. ObjectClassFieldValue Translation

If an ObjectClassFieldValue is a BuiltinValue in a FixedTypeFieldVal, then the translation of the ObjectClassFieldValue is the translation of the BuiltinValue.

If an ObjectClassFieldValue is a ReferencedValue in a FixedTypeFieldVal, then the translation of the ObjectClassFieldValue is the translation of the ReferencedValue.

If an ObjectClassFieldValue is an OpenTypeFieldVal, then the translation of the ObjectClassFieldValue is an element item with the [local name] "value". An element item with the [local name] "annotation" MAY be added to the [children] of the <value> element item. An element item with the [local name] "openTypeValue" SHALL be

appended to the [children] of the <value> element item. The translation of the Type in the OpenTypeFieldVal SHALL be added to the [children] or [attributes] of the <openTypeValue> element item. The translation of the Value in the OpenTypeFieldVal SHALL be added to the [children] or [attributes] of the <openTypeValue> element item.

Example

```
myValue TYPE-IDENTIFIER.&Type ::= INTEGER:123

<namedValue name="myValue">
  <type>
    <fromClass class="asn1:TYPE-IDENTIFIER" fieldName="Type"/>
  </type>
  <value>
    <openTypeValue type="asn1:INTEGER" literalValue="123"/>
  </value>
</namedValue>
```

8. Translation of Value Sets

The translation of a ValueSet is an element item with the [local name] "valueSet". An element item with the [local name] "annotation" MAY be added to the [children] of the <valueSet> element item. The translation of the ElementSetSpecs instance in the ValueSet SHALL be appended to the [children] of the <valueSet> element item.

Example

```
{ 1 | 3..7, ..., 9..19 EXCEPT ( 11 | 12 ) }

<valueSet>
  <union>
    <literalValue>1</literalValue>
    <range>
      <minInclusive literalValue="3"/>
      <maxInclusive literalValue="7"/>
    </range>
  </union>
  <extension>
    <all>
      <range>
        <minInclusive literalValue="9"/>
        <maxInclusive literalValue="19"/>
      </range>
      <except>
        <union>
```

```
        <literalValue>11</literalValue>
        <literalValue>12</literalValue>
    </union>
</except>
</all>
</extension>
</valueSet>
```

8.1. ElementSetSpecs Translation

The translation of an ElementSetSpecs instance where the ellipsis ("...") is not present is the translation of the ElementSetSpec in the RootElementSetSpec.

The translation of an ElementSetSpecs instance where the ellipsis ("...") is present is the translation of the ElementSetSpec in the RootElementSetSpec followed by an element item with the [local name] "extension". If an AdditionalElementSetSpec is present in the ElementSetSpecs, then the translation of the ElementSetSpec in the AdditionalElementSetSpec SHALL be added to the [children] of the <extension> element item.

8.2. ElementSetSpec Translation

If an ElementSetSpec is of the "ALL Exclusions" form, then the translation of the ElementSetSpec is an element item with the [local name] "all". An element item with the [local name] "except" SHALL be added to the [children] of the <all> element item. The translation of the Elements instance in the Exclusions SHALL be added to the [children] of the <except> element item.

If an ElementSetSpec is a Unions instance, then the translation of the ElementSetSpec is the translation of the Unions instance.

If a Unions instance has only one nested Intersections instance, then the translation of the Unions instance is the translation of that Intersections instance; otherwise, the translation of the Unions instance is an element item with the [local name] "union". In the latter case, the translation of each nested Intersections instance SHALL be appended to the [children] of the <union> element item.

If an Intersections instance has only one nested IntersectionElements instance, then the translation of the Intersections instance is the translation of that IntersectionElements instance; otherwise, the translation of the Intersections instance is an element item with the [local name] "intersection". In the latter case, the translation of each nested IntersectionElements instance SHALL be appended to the [children] of the <intersection> element item.

If an IntersectionElements instance is of the "Elems Exclusions" form, then the translation of the IntersectionElements instance is an element item with the [local name] "all". The translation of the Elements instance in the Elems SHALL be added to the [children] of the <all> element item. An element item with the [local name] "except" SHALL be appended to the [children] of the <all> element item. The translation of the Elements instance in the Exclusions SHALL be added to the [children] of the <except> element item.

If an IntersectionElements instance is an Elements instance, then the translation of the IntersectionElements instance is the translation of the Elements instance.

The translation of an Elements instance is the translation of the SubtypeElements, ObjectSetElements, or ElementSetSpec in the Elements instance.

8.3. SubtypeElements Translation

If a SubtypeElements instance is a SingleValue, then the translation of the SubtypeElements instance is the translation of the Value in the SingleValue, except that an attribute form of the Value translation SHALL NOT be used.

If a SubtypeElements instance is a ContainedSubtype, then the translation of the SubtypeElements instance is an element item with the [local name] "includes". The translation of the Type in the ContainedSubtype SHALL be added to the [children] or [attributes] of the <includes> element item.

If a SubtypeElements instance is a ValueRange, then the translation of the SubtypeElements instance is the translation of the ValueRange.

If a SubtypeElements instance is a SizeConstraint, then the translation of the SubtypeElements instance is an element item with the [local name] "size". The translation of the Constraint in the SizeConstraint SHALL be added to the [children] of the <size> element item.

If a SubtypeElements instance is a TypeConstraint, then the translation of the SubtypeElements instance is an element item with the [local name] "typeConstraint". The translation of the Type in the TypeConstraint SHALL be added to the [children] or [attributes] of the <typeConstraint> element item.

If a SubtypeElements instance is a PermittedAlphabet, then the translation of the SubtypeElements instance is an element item with the [local name] "from". The translation of the Constraint in the PermittedAlphabet SHALL be added to the [children] of the <from> element item.

If a SubtypeElements instance is an InnerTypeConstraints instance, then the translation of the SubtypeElements instance is the translation of the InnerTypeConstraints instance.

If a SubtypeElements instance is a PatternConstraint, then the translation of the SubtypeElements instance is an element item with the [local name] "pattern". The translation of the Value in the PatternConstraint SHALL be added to the [children] or [attributes] of the <pattern> element item.

8.3.1. ValueRange Translation

The translation of a ValueRange is an element item with the [local name] "range".

If the LowerEndpoint in the ValueRange is of the "LowerEndValue <" form, then an element item with the [local name] "minExclusive" SHALL be added to the [children] of the <range> element item. If the LowerEndValue is a Value, then the translation of the Value SHALL be added to the [children] or [attributes] of the <minExclusive> element item.

If the LowerEndpoint in the ValueRange is of the "LowerEndValue" form and the LowerEndValue is a Value, then an element item with the [local name] "minInclusive" SHALL be added to the [children] of the <range> element item. The translation of the Value in the LowerEndValue SHALL be added to the [children] or [attributes] of the <minInclusive> element item.

If the LowerEndpoint in the ValueRange is of the "LowerEndValue" form and the LowerEndValue is "MIN", then an element item with the [local name] "minInclusive" MAY be added to the [children] of the <range> element item.

If the UpperEndpoint in the ValueRange is of the "< UpperEndValue" form, then an element item with the [local name] "maxExclusive" SHALL be added to the [children] of the <range> element item. If the UpperEndValue is a Value, then the translation of the Value SHALL be added to the [children] or [attributes] of the <maxExclusive> element item.

If the UpperEndpoint in the ValueRange is of the "UpperEndValue" form and the UpperEndValue is a Value, then an element item with the [local name] "maxInclusive" SHALL be added to the [children] of the <range> element item. The translation of the Value in the UpperEndValue SHALL be added to the [children] or [attributes] of the <maxInclusive> element item.

If the UpperEndpoint in the ValueRange is of the "UpperEndValue" form and the UpperEndValue is "MAX", then an element item with the [local name] "maxInclusive" MAY be added to the [children] of the <range> element item.

Examples

1..10

```
<range>
  <minInclusive literalValue="1"/>
  <maxInclusive literalValue="10"/>
</range>
```

0..MAX

```
<range>
  <minInclusive literalValue="0"/>
</range>
```

0<..

```
<range>
  <minExclusive literalValue="0"/>
  <maxExclusive/>
</range>
```

8.3.2. InnerTypeConstraints Translation

The translation of an InnerTypeConstraints instance that has a SingleTypeConstraint is an element item with the [local name] "withComponent". The translation of the Constraint in the SingleTypeConstraint SHALL be added to the [children] of the <withComponent> element item.

The translation of an InnerTypeConstraints instance that has a MultipleTypeConstraints instance is an element item with the [local name] "withComponents". If the MultipleTypeConstraints instance is a PartialSpecification, then an attribute item with the [local name] "partial" and the [normalized value] "true" or "1" SHALL be added to the [attributes] of the <withComponents> element item.

If the `MultipleTypeConstraints` instance is a `FullSpecification`, then an attribute item with the [local name] "partial" and the [normalized value] "false" or "0" MAY be added to the [attributes] of the `<withComponents>` element item. The translation of each `NamedConstraint` nested in the `TypeConstraints` instance in the `FullSpecification` or `PartialSpecification` SHALL be appended to the [children] of the `<withComponents>` element item.

The translation of a `NamedConstraint` is an element item with the same [local name] (i.e., "attribute", "element", "component", "group", "member", or "simpleContent") as the translation of the `NamedType` corresponding to the identifier in the `NamedConstraint`. An attribute item with the [local name] "name" SHALL be added to the [attributes] of the `<attribute>`, `<element>`, `<component>`, `<group>`, `<member>`, or `<simpleContent>` element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the `NamedType` corresponding to the identifier in the `NamedConstraint`.

If the `PresenceConstraint` in the `ComponentConstraint` in the `NamedConstraint` is not empty, then an attribute item with the [local name] "use" SHALL be added to the [attributes] of the `<attribute>`, `<element>`, `<component>`, `<group>`, `<member>`, or `<simpleContent>` element item. The [normalized value] of this attribute item is the text of the `PresenceConstraint` with all letters downcased, i.e., either "present", "absent", or "optional".

If the `ValueConstraint` in the `ComponentConstraint` in the `NamedConstraint` is not empty, then the translation of the `Constraint` in the `ValueConstraint` SHALL be added to the [children] of the `<attribute>`, `<element>`, `<component>`, `<group>`, `<member>`, or `<simpleContent>` element item.

9. Translation of Object Classes

The translation of an `ObjectClass` is the translation of the `DefinedObjectClass`, `ObjectClassDefn`, or `ParameterizedObjectClass` in the `ObjectClass`.

The translation of a `ParameterizedObjectClass` is described in Section 13.

9.1. DefinedObjectClass Translation

If a `DefinedObjectClass` is an `objectclassreference` (not a `DummyReference`), an `ExternalObjectClassReference`, or a `UsefulObjectClassReference`, then the translation of the

DefinedObjectClass is either the attribute form translation of an object class reference, or the element form translation of an object class reference.

The attribute form translation of an object class reference is an attribute item with the [local name] "class". The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced object class definition (see Section 5.1). In the case of a UsefulObjectClassReference, the namespace name of the expanded name is "urn:ietf:params:xml:ns:asn1", and the local name is either "TYPE-IDENTIFIER" or "ABSTRACT-SYNTAX", as the case may be. The attribute form translation SHALL NOT be used if the expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items (see Section 5.1). Otherwise, the translator is free to choose either the attribute form or element form translation for an object class reference.

The element form translation of an object class reference is an element item with the [local name] "class". An element item with the [local name] "annotation" MAY be added to the [children] of the <class> element item. An attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <class> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced object class definition. In the case of a UsefulObjectClassReference, the namespace name of the expanded name is "urn:ietf:params:xml:ns:asn1" and the local name is either "TYPE-IDENTIFIER" or "ABSTRACT-SYNTAX", as the case may be. If the expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items, then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <class> element item; otherwise, if the module containing the referenced object class definition has a schema identity URI, then an attribute item with the [local name] "context" MAY be added to the [attributes] of the <class> element item. The [normalized value] of this attribute item is the schema identity URI of the module containing the referenced object class definition.

Aside: If a reference name is not distinct, then the module containing the referenced definition must have a schema identity URI (see Section 5.1).

The translation of the DefinedObjectClass is the same whether the object class definition is referenced by an objectclassreference or an ExternalObjectClassReference.

If a DefinedObjectClass is a DummyReference, then the translation of the DefinedObjectClass is the translation of the DummyReference (see Section 13).

9.2. ObjectClassDefn Translation

The translation of an ObjectClassDefn is an element item with the [local name] "class". An element item with the [local name] "annotation" MAY be added to the [children] of the <class> element item. The translation of each FieldSpec in the ObjectClassDefn SHALL be appended to the [children] of the <class> element item.

The translation of a FieldSpec is the translation of the TypeFieldSpec, FixedTypeValueFieldSpec, VariableTypeValueFieldSpec, FixedTypeValueSetFieldSpec, VariableTypeValueSetFieldSpec, ObjectFieldSpec, or ObjectSetFieldSpec in the FieldSpec.

9.2.1. TypeFieldSpec Translation

The translation of a TypeFieldSpec where the TypeOptionalitySpec is absent is an element item with the [local name] "typeField".

The translation of a TypeFieldSpec with a TypeOptionalitySpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "typeField" SHALL be added to the [children] of the <optional> element item.

The translation of a TypeFieldSpec with a TypeOptionalitySpec of the "DEFAULT Type" form is an element item with the [local name] "optional". An element item with the [local name] "typeField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional> element item. The translation of the Type in the TypeOptionalitySpec SHALL be added to the [children] or [attributes] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <typeField> element item. The [normalized value] of this attribute item is the typefieldreference in the TypeFieldSpec, without the ampersand character ('&', U+0026). An element item with the [local name] "annotation" MAY be added to the [children] of the <typeField> element item.

Example

```
CLASS {
    &One,
    &Two    OPTIONAL,
    &Three  DEFAULT OBJECT IDENTIFIER
}
```

```
<class>
  <typeField name="One"/>
  <optional>
    <typeField name="Two"/>
  </optional>
  <optional>
    <typeField name="Three"/>
    <default type="asn:OBJECT-IDENTIFIER"/>
  </optional>
</class>
```

9.2.2. FixedTypeValueFieldSpec Translation

The translation of a FixedTypeValueFieldSpec where the ValueOptionalitySpec is absent is an element item with the [local name] "valueField".

The translation of a FixedTypeValueFieldSpec with a ValueOptionalitySpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "valueField" SHALL be added to the [children] of the <optional> element item.

The translation of a FixedTypeValueFieldSpec with a ValueOptionalitySpec of the "DEFAULT Value" form is an element item with the [local name] "optional". An element item with the [local name] "valueField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional> element item. The translation of the Value in the ValueOptionalitySpec SHALL be added to the [children] or [attributes] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <valueField> element item. The [normalized value] of this attribute item is the valuefieldreference in the FixedTypeValueFieldSpec, without the ampersand character ('&', U+0026). If the "UNIQUE" keyword is present, then an attribute item with the [local name] "unique" and [normalized value] "true" or "1" SHALL be added to the [attributes] of the <valueField> element item; otherwise, an attribute item with the [local name] "unique" and [normalized value] "false" or "0" MAY be added to the [attributes] of the <valueField> element item. An element item with the [local name] "annotation" MAY be added to the [children] of the <valueField> element item. The translation of the Type in the FixedTypeValueFieldSpec SHALL be added to the [children] or [attributes] of the <valueField> element item.

Example

```

CLASS {
    &one      OBJECT IDENTIFIER UNIQUE,
    &two      BOOLEAN OPTIONAL,
    &three    INTEGER DEFAULT 0
}

<class>
  <valueField name="one" unique="true"
    type="asn:OBJECT-IDENTIFIER"/>
  <optional>
    <valueField name="two" type="asn:BOOLEAN"/>
  </optional>
  <optional>
    <valueField name="three" type="asn:INTEGER"/>
    <default literalValue="0"/>
  </optional>
</class>

```

9.2.3. FixedTypeValueSetFieldSpec Translation

The translation of a FixedTypeValueSetFieldSpec where the ValueSetOptionalitySpec is absent is an element item with the [local name] "valueSetField".

The translation of a FixedTypeValueSetFieldSpec with a ValueSetOptionalitySpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "valueSetField" SHALL be added to the [children] of the <optional> element item.

The translation of a FixedTypeValueSetFieldSpec with a ValueSetOptionalitySpec of the "DEFAULT ValueSet" form is an element item with the [local name] "optional". An element item with the [local name] "valueSetField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional> element item. The translation of the ValueSet in the ValueSetOptionalitySpec SHALL be added to the [children] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <valueSetField> element item. The [normalized value] of this attribute item is the valuesetfieldreference in the FixedTypeValueSetFieldSpec, without the ampersand character ('&', U+0026). An element item with the [local name] "annotation" MAY be added to the [children] of the

<valueSetField> element item. The translation of the Type in the FixedTypeValueSetFieldSpec SHALL be added to the [children] or [attributes] of the <valueSetField> element item.

Example

```

CLASS {
    &One      UTF8String,
    &Two      BOOLEAN OPTIONAL,
    &Three    INTEGER DEFAULT { 1 | 2 }
}

<class>
  <valueSetField name="One" type="asn:UTF8String"/>
  <optional>
    <valueSetField name="Two" type="asn:BOOLEAN"/>
  </optional>
  <optional>
    <valueSetField name="Three" type="asn:INTEGER"/>
    <default>
      <valueSet>
        <union>
          <literalValue>1</literalValue>
          <literalValue>2</literalValue>
        </union>
      </valueSet>
    </default>
  </optional>
</class>

```

9.2.4. VariableTypeValueFieldSpec Translation

The translation of a VariableTypeValueFieldSpec where the ValueOptionalitySpec is absent is an element item with the [local name] "valueField".

The translation of a VariableTypeValueFieldSpec with a ValueOptionalitySpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "valueField" SHALL be added to the [children] of the <optional> element item.

The translation of a VariableTypeValueFieldSpec with a ValueOptionalitySpec of the "DEFAULT Value" form is an element item with the [local name] "optional". An element item with the [local name] "valueField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional>

element item. The translation of the Value in the ValueOptionaltySpec SHALL be added to the [children] or [attributes] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <valueField> element item. The [normalized value] of this attribute item is the valuefieldreference in the VariableTypeValueFieldSpec, without the ampersand character ('&', U+0026). An element item with the [local name] "annotation" MAY be added to the [children] of the <valueField> element item. An element item with the [local name] "typeFromField" SHALL be appended to the [children] of the <valueField> element item. The translation of the FieldName in the VariableTypeValueFieldSpec SHALL be added to the [children] or [attributes] of the <typeFromField> element item.

Example

```
CLASS {
    &Syntax DEFAULT INTEGER,
    &one      &Syntax,
    &two      &Syntax OPTIONAL,
    &three    &Syntax DEFAULT 0
}

<class>
  <optional>
    <typeField name="Syntax"/>
    <default type="asn:INTEGER"/>
  </optional>
  <valueField name="one">
    <typeFromField fieldName="Syntax"/>
  </valueField>
  <optional>
    <valueField name="two">
      <typeFromField fieldName="Syntax"/>
    </valueField>
  </optional>
  <optional>
    <valueField name="three">
      <typeFromField fieldName="Syntax"/>
    </valueField>
    <default literalValue="0"/>
  </optional>
</class>
```


9.2.5. VariableTypeValueSetFieldSpec Translation

The translation of a VariableTypeValueSetFieldSpec where the ValueSetOptionalitySpec is absent is an element item with the [local name] "valueSetField".

The translation of a VariableTypeValueSetFieldSpec with a ValueSetOptionalitySpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "valueSetField" SHALL be added to the [children] of the <optional> element item.

The translation of a VariableTypeValueSetFieldSpec with a ValueSetOptionalitySpec of the "DEFAULT ValueSet" form is an element item with the [local name] "optional". An element item with the [local name] "valueSetField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional> element item. The translation of the ValueSet in the ValueSetOptionalitySpec SHALL be added to the [children] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <valueSetField> element item. The [normalized value] of this attribute item is the valuesetfieldreference in the VariableTypeValueSetFieldSpec, without the ampersand character ('&', U+0026). An element item with the [local name] "annotation" MAY be added to the [children] of the <valueSetField> element item. An element item with the [local name] "typeFromField" SHALL be appended to the [children] of the <valueSetField> element item. The translation of the FieldName in the VariableTypeValueSetFieldSpec SHALL be added to the [children] or [attributes] of the <typeFromField> element item.

Example

```

CLASS {
    &Syntax DEFAULT INTEGER,
    &One      &Syntax,
    &Two      &Syntax OPTIONAL,
    &Three    &Syntax DEFAULT { 1 | 2 }
}

<class>
  <optional>
    <typeField name="Syntax"/>
    <default type="asn:INTEGER"/>
  </optional>

```

```
<valueSetField name="One">
  <typeFromField fieldName="Syntax"/>
</valueSetField>
<optional>
  <valueSetField name="Two">
    <typeFromField fieldName="Syntax"/>
  </valueSetField>
</optional>
<optional>
  <valueSetField name="Three">
    <typeFromField fieldName="Syntax"/>
  </valueSetField>
  <default>
    <valueSet>
      <union>
        <literalValue>1</literalValue>
        <literalValue>2</literalValue>
      </union>
    </valueSet>
  </default>
</optional>
</class>
```

9.2.6. FieldName Translation

The translation of a `FieldName` is either, at the translator's option, an attribute item with the [local name] "fieldName" added to the [attributes] of the enclosing element item, or an element item with the [local name] "fieldName" appended to the [children] of the enclosing element item.

The [normalized value] of the `fieldName` attribute item is a solidus ('/', U+002F) separated list of the `PrimitiveFieldName` instances in the `FieldName`, without the ampersand characters ('&', U+0026). Leading and/or trailing white space characters MAY be added to the [normalized value] of the attribute item. White space characters MAY be added immediately before and/or after any solidus character ('/', U+002F) in the [normalized value].

The [children] property of the `<fieldName>` element item is set to the sequence of character items for a solidus ('/', U+002F) separated list of the `PrimitiveFieldName` instances in the `FieldName`, without the ampersand characters ('&', U+0026). Leading and/or trailing white space character items MAY be added to the [children] of the `<fieldName>` element item. White space character items MAY be added immediately before and/or after any character item for the solidus character ('/', U+002F).

Example

```
&Linked.&ArgumentType
```

```
<fieldName>Linked/ArgumentType</fieldName>
```

9.2.7. ObjectFieldSpec Translation

The translation of an ObjectFieldSpec where the ObjectOptionalitySpec is absent is an element item with the [local name] "objectField".

The translation of an ObjectFieldSpec with an ObjectOptionalitySpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "objectField" SHALL be added to the [children] of the <optional> element item.

The translation of an ObjectFieldSpec with an ObjectOptionalitySpec of the "DEFAULT Object" form is an element item with the [local name] "optional". An element item with the [local name] "objectField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional> element item. The translation of the Object in the ObjectOptionalitySpec SHALL be added to the [children] or [attributes] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <objectField> element item. The [normalized value] of this attribute item is the objectfieldreference in the ObjectFieldSpec, without the ampersand character ('&', U+0026). An element item with the [local name] "annotation" MAY be added to the [children] of the <objectField> element item. The translation of the DefinedObjectClass in the ObjectFieldSpec SHALL be added to the [children] or [attributes] of the <objectField> element item.

Example

```
CLASS {
    &one      TYPE-IDENTIFIER,
    &two      ABSTRACT-SYNTAX OPTIONAL,
    &three    TYPE-IDENTIFIER DEFAULT myObject
}
```

```
<class>
  <objectField name="one" class="asn:TYPE-IDENTIFIER"/>
  <optional>
    <objectField name="two" class="asn:ABSTRACT-SYNTAX"/>
  </optional>
  <optional>
    <objectField name="three" class="asn:TYPE-IDENTIFIER"/>
    <default object="tns:myObject"/>
  </optional>
</class>
```

9.2.8. ObjectSetFieldSpec Translation

The translation of an ObjectSetFieldSpec where the ObjectSetOptionalSpec is absent is an element item with the [local name] "objectSetField".

The translation of an ObjectSetFieldSpec with an ObjectSetOptionalSpec of the "OPTIONAL" form is an element item with the [local name] "optional". An element item with the [local name] "objectSetField" SHALL be added to the [children] of the <optional> element item.

The translation of an ObjectSetFieldSpec with an ObjectSetOptionalSpec of the "DEFAULT ObjectSet" form is an element item with the [local name] "optional". An element item with the [local name] "objectSetField" SHALL be added to the [children] of the <optional> element item. An element item with the [local name] "default" SHALL be appended to the [children] of the <optional> element item. The translation of the ObjectSet in the ObjectSetOptionalSpec SHALL be added to the [children] or [attributes] of the <default> element item.

An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <objectSetField> element item. The [normalized value] of this attribute item is the objectsetfieldreference in the ObjectSetFieldSpec, without the ampersand character ('&', U+0026). An element item with the [local name] "annotation" MAY be added to the [children] of the <objectSetField> element item. The translation of the DefinedObjectClass in the ObjectSetFieldSpec SHALL be added to the [children] or [attributes] of the <objectSetField> element item.

Example

```
CLASS {
    &One      TYPE-IDENTIFIER,
    &Two      ABSTRACT-SYNTAX OPTIONAL,
    &Three    TYPE-IDENTIFIER DEFAULT { myObject }
}

<class>
  <objectSetField name="One" class="asn:TYPE-IDENTIFIER"/>
  <optional>
    <objectSetField name="Two" class="asn:ABSTRACT-SYNTAX"/>
  </optional>
  <optional>
    <objectSetField name="Three" class="asn:TYPE-IDENTIFIER"/>
    <default>
      <objectSet>
        <object ref="tns:myObject"/>
      </objectSet>
    </default>
  </optional>
</class>
```

10. Translation of Objects

The translation of an Object is the translation of the DefinedObject, ObjectDefn, ObjectFromObject, or ParameterizedObject in the Object.

The translation of a ParameterizedObject is described in Section 13.

10.1. DefinedObject Translation

If a DefinedObject is an objectreference (not a DummyReference) or an ExternalObjectReference, then the translation of the DefinedObject is either the attribute form translation of an object reference, or the element form translation of an object reference.

The attribute form translation of an object reference is an attribute item with the [local name] "object". The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced object definition (see Section 5.1). The attribute form translation SHALL NOT be used if this expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items (see Section 5.1).

The element form translation of an object reference is an element item with the [local name] "object". An element item with the [local name] "annotation" MAY be added to the [children] of the

<object> element item. An attribute item with the [local name] "ref" SHALL be added to the [attributes] of the <object> element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced object definition. If this expanded name is not distinct with respect to the current module and the modules referenced by its <import> element items, then an attribute item with the [local name] "context" SHALL be added to the [attributes] of the <object> element item; otherwise, if the module containing the referenced object definition has a schema identity URI, then an attribute item with the [local name] "context" MAY be added to the [attributes] of the <object> element item. The [normalized value] of this attribute item is the schema identity URI of the module containing the referenced object definition.

Aside: If a reference name is not distinct, then the module containing the referenced definition must have a schema identity URI (see Section 5.1).

The translation of the DefinedObject is the same whether the object definition is referenced by an objectreference or an ExternalObjectReference.

Usually the translator is free to choose either the attribute form or element form translation for an object reference; however, in some contexts the attribute form is explicitly disallowed.

If a DefinedObject is a DummyReference, then the translation of the DefinedObject is the translation of the DummyReference (see Section 13).

10.2. ObjectDefn Translation

An ObjectDefn that is a DefinedSyntax is first converted to the equivalent DefaultSyntax and then translated.

The translation of an ObjectDefn is an element item with the [local name] "object". An element item with the [local name] "annotation" MAY be added to the [children] of the <object> element item. The translation of each FieldSetting in the DefaultSyntax in the ObjectClassDefn SHALL be appended to the [children] of the <object> element item.

The translation of a FieldSetting is an element item with the [local name] "field". An attribute item with the [local name] "name" SHALL be added to the [attributes] of the <field> element item. The [normalized value] of this attribute item is the PrimitiveFieldName in the FieldSetting, without the ampersand character ('&', U+0026). The translation of the Type, Value, ValueSet, Object, or ObjectSet in

the Setting in the FieldSetting SHALL be added to the [children] or [attributes] of the <field> element item.

Example

```
-- This is the governing object class.
ONE-OF-EVERYTHING ::= CLASS {
    &One,
    &two    INTEGER,
    &Three  INTEGER,
    &four   TYPE-IDENTIFIER,
    &Five   TYPE-IDENTIFIER
}

<namedClass name="ONE-OF-EVERYTHING">
  <class>
    <typeField name="One"/>
    <valueField name="two" type="asn:INTEGER"/>
    <valueSetField name="Three" type="asn:INTEGER"/>
    <objectField name="four" class="asn:TYPE-IDENTIFIER"/>
    <objectSetField name="Five" class="asn:TYPE-IDENTIFIER"/>
  </class>
</namedClass>

mixedBag ONE-OF-EVERYTHING ::= {
    &One    BOOLEAN,
    &two    99,
    &Three  { 1 | 2 },
    &four   myObject,
    &Five   { myObject }
}

<namedObject name="mixedBag" class="tns:ONE-OF-EVERYTHING">
  <object>
    <field name="One" type="asn:BOOLEAN"/>
    <field name="two" literalValue="99"/>
    <field name="Three">
      <valueSet>
        <union>
          <literalValue>1</literalValue>
          <literalValue>2</literalValue>
        </union>
      </valueSet>
    </field>
    <field name="four" object="tns:myObject"/>
    <field name="Five">
      <objectSet>
        <object ref="tns:myObject"/>
      </objectSet>
    </field>
  </object>
</namedObject>
```

```
    </objectSet>
  </field>
</object>
</namedObject>
```

10.3. ObjectFromObject Translation

The translation of an ObjectFromObject is an element item with the [local name] "object". An element item with the [local name] "annotation" MAY be added to the [children] of the <object> element item. An element item with the [local name] "fromObjects" SHALL be appended to the [children] of the <object> element item.

The translation of the ReferencedObjects instance in the ObjectFromObject SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

The translation of the FieldName in the ObjectFromObject SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

11. Translation of Object Sets

If an ObjectSet matches the form "{ DefinedObjectSet }" (i.e., a DefinedObjectSet in an ObjectSetElements instance in an Elements instance in a lone IntersectionElements instance in a lone Intersections instance in a Unions instance in an ElementSetSpec in a RootElementSetSpec in an ObjectSetSpec without an AdditionalElementSetSpec), then the translator MAY use the translation of the DefinedObjectSet as the translation of the ObjectSet; otherwise, the translation of an ObjectSet is an element item with the [local name] "objectSet". An element item with the [local name] "annotation" MAY be added to the [children] of the <objectSet> element item. The translation of the ObjectSetSpec in the ObjectSet SHALL be appended to the [children] of the <objectSet> element item.

Aside: An ObjectSet that is directly a DefinedObjectSet is a notational capability that does not exist in ASN.1, but is allowed in ASN.X to avoid excessive nesting of <objectSet> element items in the expansion of parameterized definitions.

If an ObjectSetSpec contains only a RootElementSetSpec, then the translation of the ObjectSetSpec is the translation of the ElementSetSpec in the RootElementSetSpec.

If an `ObjectSetSpec` contains a `RootElementSetSpec` and an ellipsis ("`...`"), then the translation of the `ObjectSetSpec` is the translation of the `ElementSetSpec` in the `RootElementSetSpec` followed by an element item with the [local name] "`extension`". If an `AdditionalElementSetSpec` is present, then the translation of the `ElementSetSpec` in the `AdditionalElementSetSpec` SHALL be added to the [children] of the `<extension>` element item.

If an `ObjectSetSpec` does not contain a `RootElementSetSpec`, then the translation of the `ObjectSetSpec` is an element item with the [local name] "`extension`". If an `AdditionalElementSetSpec` is present, then the translation of the `ElementSetSpec` in the `AdditionalElementSetSpec` SHALL be added to the [children] of the `<extension>` element item.

Nested within the `ElementSetSpec` will be one or more `ObjectSetElements` instances.

11.1.1. `DefinedObjectSet` Translation

If a `DefinedObjectSet` is an `objectsetreference` (not a `DummyReference`) or an `ExternalObjectSetReference`, then the translation of the `DefinedObjectSet` is either the attribute form translation of an object set reference, or the element form translation of an object set reference.

The attribute form translation of an object set reference is an attribute item with the [local name] "`objectSet`". The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced object set definition (see Section 5.1). The attribute form translation SHALL NOT be used if this expanded name is not distinct with respect to the current module and the modules referenced by its `<import>` element items (see Section 5.1).

The element form translation of an object set reference is an element item with the [local name] "`objectSet`". An element item with the [local name] "`annotation`" MAY be added to the [children] of the `<objectSet>` element item. An attribute item with the [local name] "`ref`" SHALL be added to the [attributes] of the `<objectSet>` element item. The [normalized value] of this attribute item is a qualified name for the expanded name of the referenced object set definition. If this expanded name is not distinct with respect to the current module and the modules referenced by its `<import>` element items, then an attribute item with the [local name] "`context`" SHALL be added to the [attributes] of the `<objectSet>` element item; otherwise, if the module containing the referenced object set definition has a schema identity URI, then an attribute item with the [local name] "`context`"

MAY be added to the [attributes] of the <objectSet> element item. The [normalized value] of this attribute item is the schema identity URI of the module containing the referenced object set definition.

Aside: If a reference name is not distinct, then the module containing the referenced definition must have a schema identity URI (see Section 5.1).

The translation of the DefinedObjectSet is the same whether the object definition is referenced by an objectsetreference or an ExternalObjectSetReference.

Usually the translator is free to choose either the attribute form or element form translation for an object set reference; however, in some contexts the attribute form is explicitly disallowed.

If a DefinedObjectSet is a DummyReference, then the translation of the DefinedObjectSet is the translation of the DummyReference (see Section 13).

11.2. ObjectSetElements Translation

If an ObjectSetElements instance is an Object, then the translation of the ObjectSetElements instance is the translation of the Object, except that the attribute form of the DefinedObject translation SHALL NOT be used if the Object is a DefinedObject.

If an ObjectSetElements instance is a DefinedObjectSet, then the translation of the ObjectSetElements instance is the translation of the DefinedObjectSet, except that the attribute form of the DefinedObjectSet translation SHALL NOT be used.

If an ObjectSetElements instance is an ObjectSetFromObjects, then the translation of the ObjectSetElements instance is the translation of the ObjectSetFromObjects.

If an ObjectSetElements instance is a ParameterizedObjectSet, then the translation of the ObjectSetElements instance is the translation of the ParameterizedObjectSet (see Section 13).

Aside: The in-line expansion of a ParameterizedObjectSet results in an ObjectSet. An ObjectSetElements instance that is an ObjectSet is a notational capability that does not exist in ASN.1, but is allowed in ASN.X to avoid the need to manufacture a reference name for the expanded parameterized definition.

11.2.1. ObjectSetFromObjects Translation

The translation of an ObjectSetFromObjects instance is an element item with the [local name] "objectSet". An element item with the [local name] "annotation" MAY be added to the [children] of the <objectSet> element item. An element item with the [local name] "fromObjects" SHALL be appended to the [children] of the <objectSet> element item.

The translation of the ReferencedObjects instance in the ObjectSetFromObjects SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

The translation of the FieldName in the ObjectSetFromObjects SHALL be added to the [children] or [attributes] of the <fromObjects> element item.

12. Translation of Information From Objects

If a ReferencedObjects instance is a DefinedObject (not a DummyReference), then the translation of the ReferencedObjects instance is the translation of the DefinedObject.

If a ReferencedObjects instance is a DefinedObjectSet (not a DummyReference), then the translation of the ReferencedObjects instance is the translation of the DefinedObjectSet.

If a ReferencedObjects instance is a DummyReference, ParameterizedObject, or ParameterizedObjectSet, then the translation of the ReferencedObjects instance is the translation of that DummyReference, ParameterizedObject, or ParameterizedObjectSet (see Section 13).

Aside: The in-line expansion of a ParameterizedObject or ParameterizedObjectSet results in an Object or ObjectSet, respectively. A ReferencedObjects instance that is an Object or ObjectSet is a notational capability that does not exist in ASN.1, but is allowed in ASN.X to avoid the need to manufacture a reference name for an expanded parameterized definition.

13. Translation of Parameterized Definitions

The translation of an ASN.1 specification into ASN.X replaces any DummyReference [X.683] or reference to a parameterized definition [X.683] with the definition expanded in-line (except for a special case involving recursive parameterized types). For example, a ParameterizedObject is replaced by the Object on the right-hand side of the referenced ParameterizedObjectAssignment.

The definition that substitutes for a DummyReference or parameterized reference (e.g., the Object that substitutes for a ParameterizedObject) potentially comes from a different module from the reference. Expanding a DummyReference or parameterized reference in-line puts the substitute definition into the context of the module containing the reference, which could therefore alter the interpretation of the substitute definition.

A type definition is potentially dependent on the TagDefault and ExtensionDefault of the module in which it appears, and may also be affected by encoding instructions in an XML Encoding Rules (XER) [X.693] encoding control section [X.693-1]. Other kinds of definitions are not dependent on the module context; however, type definitions can be nested within the other kinds of definitions, so a change of context can still be significant.

Aside: Type definitions are not dependent on their module's RXER or Generic String Encoding Rules (GSER) [GSER] encoding control section [RXEREI][GSEREI] (as they are currently defined), so the presence of an encoding control section for RXER or GSER is not significant in a change of context.

The remainder of this section describes how and when a change of context is indicated in the ASN.X translation of a DummyReference or parameterized reference.

In any instance of use, the module containing the DummyReference or parameterized reference is the referencing module, and the module providing the substitute definition is the referenced module. The referenced and referencing modules may be the same module.

In the case of a ParameterizedType, the substitute definition is the Type on the right-hand side of the referenced ParameterizedTypeAssignment.

In the case of a ParameterizedValueSetType, the substitute definition is the constrained type on the right-hand side of the notional ParameterizedTypeAssignment equivalent to the referenced ParameterizedValueSetTypeAssignment (see Clause 15.8 of X.680 [X.680]).

In the case of a ParameterizedValue, the substitute definition is the Value on the right-hand side of the referenced ParameterizedValueAssignment.

In the case of a ParameterizedObjectClass, the substitute definition is the ObjectClass on the right-hand side of the referenced ParameterizedObjectClassAssignment.

In the case of a `ParameterizedObject`, the substitute definition is the `Object` on the right-hand side of the referenced `ParameterizedObjectAssignment`.

In the case of a `ParameterizedObjectSet`, the substitute definition is the `ObjectSet` on the right-hand side of the referenced `ParameterizedObjectSetAssignment`.

If the `ActualParameter` corresponding to a `DummyReference` is not a `ValueSet`, then the substitute definition for that `DummyReference` is the `Type`, `Value`, `DefinedObjectClass`, `Object`, or `ObjectSet` in the `ActualParameter`.

If the `ActualParameter` corresponding to a `DummyReference` is a `ValueSet`, then the substitute definition for that `DummyReference` is the notional constrained type equivalent to the `ValueSet`; the `ElementSetSpecs` of the `ValueSet` contributes to the constraint of the constrained type, and the governor of the `Parameter` corresponding to the `ActualParameter` is used as the parent type that is constrained.

Definition (interchangeable): The contexts of the referencing and referenced modules are interchangeable with respect to interpreting the substitute definition if:

- (1) the referenced module is the referencing module and does not contain an XER encoding control section, or
- (2) the referenced module and referencing module have the same `TagDefault` (where an absent `TagDefault` is taken to be equivalent to "EXPLICIT TAGS"), the referenced module and referencing module have the same `ExtensionDefault`, and neither module has an XER encoding control section.

Aside: A module with an XER encoding control section is not considered to have a context interchangeable with another module, including itself, because the typereference by which a substitute type definition is identified may appear in a `TargetList` in the XER encoding control section of the referenced module, and because the in-line expansion of a substitute definition may cause its text to come within the scope of a `TargetList` in the XER encoding control section of the referencing module that would not apply otherwise.

Definition (recursively contained): A `ParameterizedType` is recursively contained if its translation will be nested within the translation (i.e., in-line expansion) of another `ParameterizedType` to

which it is equivalent. A `ParameterizedValueSetType` is recursively contained if its translation will be nested within the translation of another `ParameterizedValueSetType` to which it is equivalent.

Aside: ASN.1 does not permit the other kinds of parameterized reference to be recursive.

The translation of a `DummyReference`, a `ParameterizedType` that is not recursively contained, a `ParameterizedValue`, a `ParameterizedValueSetType` that is not recursively contained, a `ParameterizedObjectClass`, a `ParameterizedObject`, or a `ParameterizedObjectSet` is either:

- (a) the translation of the substitute definition, or
- (b) an element item with the [local name] "type" if the substitute definition is a `Type`, "value" if the substitute definition is a `Value`, "class" if the substitute definition is an `ObjectClass` or `DefinedObjectClass`, "object" if the substitute definition is an `Object`, or "objectSet" if the substitute definition is an `ObjectSet`. A fully expanded reference (described shortly) SHALL be added to the [children] of the element item.

The translation in case (b) is always allowed and provides information to identify the referenced module and the referenced definition.

The translation in case (a) MAY be used instead if and only if the contexts of the referencing and referenced modules are interchangeable, or the contexts of the referencing and referenced modules are not interchangeable, but the difference between them does not affect how the substitute definition is interpreted.

Aside: There are many ways in which the substitute definition can be unaffected by a difference between the contexts of the referencing and referenced modules. One example would be where the referencing and referenced modules differ only in their `TagDefault`, but the substitute definition does not contain any `TaggedType` notation.

Note that if the translation in case (a) is used, then the referencing module is still the referencing module when considering a nested in-line expansion. If the translation in case (b) is used, then the referenced module becomes the referencing module when considering a nested in-line expansion.

If case (a) is used for the translation of a DummyReference where the substitute definition is a Type, then an attribute form translation of the substitute definition SHALL NOT be used, and an attribute item with the [local name] "explicit" and [normalized value] "true" or "1" SHALL be added to the [attributes] of the <type> element item resulting from the translation of the substitute definition. Where the automatic tagging transformation applies [X.680], this attribute item indicates that explicit tagging applies to the type instead of the usual implicit tagging.

If case (b) is used for the translation of a DummyReference where the substitute definition is a Type, then an attribute item with the [local name] "explicit" and [normalized value] "true" or "1" SHALL be added to the [attributes] of the <type> element item generated by case (b).

A fully expanded reference is an element item with the [local name] "expanded". Except in the case of a DummyReference, the reference name is indicated by an attribute item with the [local name] "name" added to the [attributes] of the <expanded> element item.

In the case of a ParameterizedType or ParameterizedValueSetType, the [normalized value] of this attribute item is the typereference of the ParameterizedType or ParameterizedValueSetType.

In the case of a ParameterizedValue, the [normalized value] of this attribute item is the valuereference of the ParameterizedValue.

In the case of a ParameterizedObjectClass, the [normalized value] of this attribute item is the objectclassreference of the ParameterizedObjectClass.

In the case of a ParameterizedObject, the [normalized value] of this attribute item is the objectreference of the ParameterizedObject.

In the case of a ParameterizedObjectSet, the [normalized value] of this attribute item is the objectsetreference of the ParameterizedObjectSet.

The "name" attribute item MAY be omitted if:

- (1) the conditions permitting the use of the translation in case (a) are satisfied, or
- (2) the reference is not a typereference, or

- (3) the reference is a typereference that does not appear in any TargetList in an XER encoding control section of the referenced module.

The "name" attribute SHALL NOT appear in the translation of a DummyReference.

The referenced module is indicated by an element item with the [local name] "module" added to the [children] of the <expanded> element item. The <module> element item MAY be omitted if the conditions permitting the use of the translation in case (a) are satisfied, or if the referencing module is the referenced module. When the <module> element item is present:

- (1) An attribute item with the [local name] "name" SHOULD be added to the [attributes] of the <module> element item. The [normalized value] of this attribute item is the modulereference in the ModuleIdentifier in the ModuleDefinition of the referenced module.
- (2) If the DefinitiveIdentifier in the ModuleIdentifier in the ModuleDefinition of the referenced module is not empty, then an attribute item with the [local name] "identifier" SHALL be added to the [attributes] of the <module> element item. The [normalized value] of this attribute item is the RXER character data translation of the DefinitiveIdentifier.
- (3) If the referenced module has a schema identity URI, then an attribute item with the [local name] "schemaIdentity" SHALL be added to the [attributes] of the <module> element item. The [normalized value] of this attribute item is the schema identity URI of the referenced module.

The [attributes] of the <module> element item MUST contain at least one of the attribute items specified in cases (1), (2), and (3).

The translation of the substitute definition SHALL be added to the [children] or [attributes] of the <expanded> element item.

Example

Consider these module definitions:

```
Templates
DEFINITIONS
AUTOMATIC TAGS ::=
BEGIN

CollectionOfThings { Thing } ::= SEQUENCE OF thing Thing
    -- the Thing on the right-hand side of the assignment is
    -- a DummyReference

END

ProtocolDefinitions
DEFINITIONS
AUTOMATIC TAGS ::=
BEGIN

IMPORTS
    CollectionOfThings{ }
        FROM Templates
    ;

CollectionOfIntegers ::= CollectionOfThings { INTEGER }
    -- the right-hand side of the assignment is
    -- a ParameterizedType

END
```

Without using the translation in case (a), the translations of these modules are:

```
<asn1:module name="Templates"/>

<asn1:module xmlns:asn1="urn:ietf:params:xml:ns:asn1"
    name="ProtocolDefinitions">

    <namedType name="CollectionOfIntegers">
        <type>
            <expanded name="CollectionOfThings">
                <module name="Templates"/>
                <type>
                    <sequenceOf>
                        <element name="thing">
                            <type>
                                <expanded>
```

```

        <module name="ProtocolDefinitions"/>
        <type ref="asn:INTEGER"/>
        </expanded>
      </type>
    </element>
  </sequenceOf>
</type>
</expanded>
</type>
</namedType>

</asn:module>

```

The translation of the Templates module is empty because the module contains only a parameterized assignment.

Since the contexts of the Templates and ProtocolDefinitions modules are interchangeable, a simpler translation of the ProtocolDefinitions module is permitted:

```

<asn:module xmlns:asn="urn:ietf:params:xml:ns:asn"
  name="ProtocolDefinitions">

  <namedType name="CollectionOfIntegers">
    <type>
      <sequenceOf>
        <element name="thing">
          <type ref="asn:INTEGER" explicit="true"/>
        </element>
      </sequenceOf>
    </type>
  </namedType>

</asn:module>

```

If a ParameterizedType or ParameterizedValueSetType is recursively contained, then its translation is an element item with the [local name] "type". An attribute item with the [local name] "ancestor" SHALL be added to the [attributes] of the <type> element item. The [normalized value] of this attribute item is the decimal digit string representing the integer value of one plus the number of intermediate enclosing <type> element items between the <type> element items resulting from the translations of the two equivalent instances of ParameterizedType or ParameterizedValueSetType. An element item with the [local name] "annotation" MAY be added to the [children] of the <type> element item.

A <type> element item with an ancestor attribute item is a reference to an ancestor <type> element item. This form for a <type> element item SHOULD NOT be used in original specifications written in ASN.X.

Aside: The form is only intended for the purpose of handling recursive parameterized type definitions in an ASN.1 specification being translated into ASN.X. Such definitions are self-referencing, but have no obvious name. It is also not easy to construct a suitable name from the surrounding context because recursive parameterized types can be embedded in other constructs, such as information objects, that are themselves unnamed.

Example

Consider these type definitions, assumed to be defined in a module that does not have an XER encoding control section:

```
Tree { ValueType } ::= SEQUENCE {
    value          [0] ValueType,
    left-subtree   [1] Tree { ValueType } OPTIONAL,
    right-subtree  [2] Tree { ValueType } OPTIONAL
}
```

```
NumberTree ::= [APPLICATION 13] Tree { INTEGER }
```

The assignment for "Tree" is not directly translated because it is a ParameterizedAssignment. The translation for the "NumberTree" assignment, up to but not yet including the Type in the TaggedType, is as follows:

```
<namedType name="NumberTree">
  <type>
    <tagged tagClass="application" number="13"/>
  </type>
</namedType>
```

The Type in the TaggedType is a ParameterizedType. Since the ParameterizedType is not recursively contained, the translation of the ParameterizedType (using the translation in case (a) above) is the translation of the Type on the right-hand side of the referenced ParameterizedTypeAssignment, namely this type:

```
SEQUENCE {
    value          [0] ValueType,
    left-subtree   [1] Tree { ValueType } OPTIONAL,
    right-subtree  [2] Tree { ValueType } OPTIONAL
}
```

ValueType is a DummyReference. The translation of the actual parameter substitutes for the DummyReference. In this case, the actual parameter is the INTEGER type.

The translation for the SEQUENCE type, up to the first component, is added to the <tagged> element:

```
<namedType name="NumberTree">
  <type>
    <tagged tagClass="application" number="13">
      <type><!-- Tree { INTEGER } -->
        <sequence>
          <element name="value">
            <type>
              <tagged number="0">
                <type ref="asn:INTEGER"
                  explicit="true"/><!-- ValueType -->
              </tagged>
            </type>
          </element>
        </sequence>
      </type>
    </tagged>
  </type>
</namedType>
```

The Type in the TaggedType for the second component is a ParameterizedType. Since this ParameterizedType is recursively contained, its translation is a <type> element with the ancestor attribute. The value of the ancestor attribute is "2" because there is one intermediate <type> element (for a TaggedType). Put another way, the translations of the equivalent instances of ParameterizedType are two <type> steps apart.

The translation of the third component of the SEQUENCE type follows the same pattern as the second component. The completed translation is as follows:

```
<namedType name="NumberTree">
  <type>
    <tagged tagClass="application" number="13">
      <type><!-- Tree { INTEGER } -->
        <sequence>
          <element name="value">
            <type>
              <tagged number="0">
                <type ref="asn:INTEGER"
                  explicit="true"/><!-- ValueType -->
              </tagged>
            </type>
          </element>
        </sequence>
      </type>
    </tagged>
  </type>
</namedType>
```

```

        </tagged>
      </type>
    </element>
  <optional>
    <element name="left-subtree">
      <type>
        <tagged number="1">
          <type ancestor="2"/><!-- Tree { ValueType } -->
        </tagged>
      </type>
    </element>
  </optional>
  <optional>
    <element name="right-subtree">
      <type>
        <tagged number="2">
          <type ancestor="2"/><!-- Tree { ValueType } -->
        </tagged>
      </type>
    </element>
  </optional>
</sequence>
</type>
</tagged>
</type>
</namedType>

```

14. EncodingControlSections Translation

If an EncodingControlSections instance contains at least one EncodingControlSection with an encodingreference that is not RXER, then the translation of the EncodingControlSections instance is an element item with the [local name] "encodingControls". The translation of each EncodingControlSection with an encodingreference that is not RXER SHALL be appended to the [children] of the <encodingControls> element item.

Aside: This is not suggesting that RXER encoding control sections are ignored. Encoding control sections for RXER are not explicitly represented in ASN.X, but rather affect how an ASN.1 module is translated into an ASN.X module. The effect of an RXER encoding control section on the translation is addressed in other parts of this specification.

Encoding control sections for other encoding rules will have explicit representations in ASN.X.

If the encodingreference in an EncodingControlSection is GSER, then the translation of the EncodingControlSection is an element item with the [local name] "GSER". The translation of the EncodingInstructionAssignmentList in the EncodingControlSection SHALL be added to the [children] of the <GSER> element item.

The EncodingInstructionAssignmentList notation is different for each set of encoding instructions. The translation into ASN.X of an EncodingInstructionAssignmentList for GSER is specified in a separate document [GSEREIT].

Aside: The translation of an EncodingInstructionAssignmentList for GSER, as it is currently defined, is always empty.

If the encodingreference in an EncodingControlSection is XER, then the translation of the EncodingControlSection is an element item with the [local name] "XER". The translation of the EncodingInstructionAssignmentList in the EncodingControlSection SHALL be added to the [children] of the <XER> element item. The translation into ASN.X of an EncodingInstructionAssignmentList for XER is specified in a separate document [XEREIT].

15. Security Considerations

The ASN.X translation of an ASN.1 specification is semantically equivalent to the original ASN.1 specification. The security considerations that apply to an application built from the original ASN.1 specification apply equally to an application built from the ASN.X translation.

Syntax-based canonicalization for XML documents (e.g., Canonical XML [CXML]) depends on the Infoset of an XML document being preserved. However, the Infoset representation of an ASN.X module potentially changes if it is decoded and re-encoded (though its ASN.1 value is preserved), disrupting the Canonical XML representation. To avoid this problem, ASN.X modules MUST be normalized prior to the application of syntax-based canonicalization. The normalization rules can be found in Section 6.13 of the specification for RXER [RXER].

16. 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.

17. References

17.1. Normative References

- [BCP14] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [URI] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [GSER] Legg, S., "Generic String Encoding Rules (GSER) for ASN.1 Types", RFC 3641, October 2003.
- [GSEREI] Legg, S., "Encoding Instructions for the Generic String Encoding Rules (GSER)", RFC 4792, January 2007.
- [RXER] Legg, S. and D. Prager, "Robust XML Encoding Rules (RXER) for Abstract Syntax Notation One (ASN.1)", RFC 4910, July 2007.
- [RXEREI] Legg, S., "Encoding Instructions for the Robust XML Encoding Rules (RXER)", RFC 4911, July 2007.
- [GSEREIT] Legg, S., "Abstract Syntax Notation X (ASN.X) Representation of Encoding Instructions for the Generic String Encoding Rules (GSER)", RFC 4913, July 2007.
- [XEREIT] Legg, S., "Abstract Syntax Notation X (ASN.X) Representation of Encoding Instructions for the XML Encoding Rules (XER)", RFC 4914, 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.693] ITU-T Recommendation X.693 (12/01) | ISO/IEC 8825-4:2002, Information technology - ASN.1 encoding rules: XML encoding rules (XER).
- [X.693-1] Amendment 1: (to ITU-T Rec. X.693 | ISO/IEC 8825-4) XER encoding instructions and EXTENDED-XER.
- [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.
- [UNICODE] The Unicode Consortium, "The Unicode Standard, Version 4.0", Boston, MA, Addison-Wesley Developers Press, 2003. ISBN 0-321-18578-1.

17.2. Informative References

- [CXML] Boyer, J., "Canonical XML Version 1.0", W3C Recommendation, <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>, March 2001.
- [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.
- [RNG] Clark, J. and M. Makoto, "RELAX NG Tutorial", OASIS Committee Specification, <http://www.oasis-open.org/committees/relax-ng/tutorial-20011203.html>, December 2001.

Appendix A. ASN.1 for ASN.X

This appendix is normative.

AbstractSyntaxNotation-X

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

```
-- Copyright (C) The IETF Trust (2007). This version of
-- this ASN.1 module is part of RFC 4912; see the RFC itself
-- for full legal notices.
--
-- Regarding this ASN.1 module or any portion of it, the author
-- makes no guarantees and is not responsible for any damage
-- resulting from its use. The author grants 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

IMPORTS

```
Markup,
AnyURI,
NCName,
Name,
QName
FROM AdditionalBasicDefinitions
{ iso(1) identified-organization(3) dod(6)
  internet(1) private(4) enterprise(1)
  xmled(21472) asnx(1) module(0) basic(0) }
GSER-EncodingInstruction,
GSER-EncodingInstructionAssignmentList
FROM GSER-EncodingInstructionNotation
{ iso(1) identified-organization(3) dod(6)
  internet(1) private(4) enterprise(1)
  xmled(21472) asnx(1) module(0) gser-ei-notation(2) }
XER-EncodingInstruction,
XER-EncodingInstructionAssignmentList
FROM XER-EncodingInstructionNotation
{ iso(1) identified-organization(3) dod(6)
  internet(1) private(4) enterprise(1)
```

```

        xmled(21472) asnx(1) module(0) xer-ei-notation(3) }
;

ModuleDefinition ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation          Annotation OPTIONAL,
    format              [ATTRIBUTE] [VERSION-INDICATOR]
                        UTF8String ("1.0", ...) DEFAULT "1.0",
    name                [ATTRIBUTE] ModuleReference,
    identifier          [ATTRIBUTE] DefinitiveIdentifier OPTIONAL,
    schemaIdentity      [ATTRIBUTE] AnyURI OPTIONAL,
    targetNamespace     [ATTRIBUTE] AnyURI OPTIONAL,
    targetPrefix        [ATTRIBUTE] NCName OPTIONAL,
    tagDefault          [ATTRIBUTE] TagDefault DEFAULT automatic,
    extensibilityImplied [ATTRIBUTE] BOOLEAN DEFAULT FALSE,
    export              SEQUENCE { } OPTIONAL,
    -- export is not used in this version
    imports             [GROUP] ImportList OPTIONAL,
    assignments         [GROUP] AssignmentList OPTIONAL,
    encodingControls     EncodingControlSections OPTIONAL
}

ModuleReference ::= TypeReference

DefinitiveIdentifier ::= OBJECT IDENTIFIER

TagDefault ::= ENUMERATED { explicit, implicit, automatic }

Annotation ::= Markup

ImportList ::= SEQUENCE SIZE (1..MAX) OF import Import

Import ::= SEQUENCE {
    name                [ATTRIBUTE] ModuleReference OPTIONAL,
    identifier          [ATTRIBUTE] DefinitiveIdentifier OPTIONAL,
    schemaIdentity      [ATTRIBUTE] AnyURI OPTIONAL,
    namespace           [ATTRIBUTE] AnyURI OPTIONAL,
    schemaLocation      [ATTRIBUTE] AnyURI OPTIONAL
}

AssignmentList ::= SEQUENCE SIZE (1..MAX) OF
    assignment [GROUP] Assignment

Assignment ::= [NO-INSERTIONS] CHOICE {
    namedType          TypeAssignment,
    namedValue         ValueAssignment,
    namedValueSet      ValueSetTypeAssignment,
    namedClass          ObjectClassAssignment,
    namedObject         ObjectAssignment,

```

```
    namedObjectSet  ObjectSetAssignment,
    component       [GROUP] TopLevelNamedType
}

TypeAssignment ::= SEQUENCE {
    annotation  Annotation OPTIONAL,
    name        [ATTRIBUTE] TypeReference,
    type        [GROUP] Type
}

TypeReference ::= UTF8String (PATTERN "[A-Z]\\w*(-\\w+)*")
                  -- \\w is equivalent to [a-zA-Z0-9]

ValueAssignment ::= SEQUENCE {
    annotation  Annotation OPTIONAL,
    name        [ATTRIBUTE] ValueReference,
    type        [GROUP] Type,
    value       [GROUP] Value
}

ValueReference ::= Identifier

Identifier ::= UTF8String (PATTERN "[a-z]\\w*(-\\w+)*")

ValueSetTypeAssignment ::= SEQUENCE {
    annotation  Annotation OPTIONAL,
    name        [ATTRIBUTE] TypeReference,
    type        [GROUP] Type,
    valueSet    [GROUP] ValueSet
}

ObjectClassAssignment ::= SEQUENCE {
    annotation  Annotation OPTIONAL,
    name        [ATTRIBUTE] ObjectClassReference,
    objectClass [GROUP] ObjectClass
}

ObjectClassReference ::= UTF8String
                      (PATTERN "[A-Z][A-Z0-9]*(-[A-Z0-9]+)*")

ObjectAssignment ::= SEQUENCE {
    annotation  Annotation OPTIONAL,
    name        [ATTRIBUTE] ObjectReference,
    objectClass [GROUP] DefinedObjectClass,
    object      [GROUP] Object
}

ObjectReference ::= ValueReference
```

```

ObjectSetAssignment ::= SEQUENCE {
    annotation    Annotation OPTIONAL,
    name          [ATTRIBUTE] ObjectSetReference,
    objectClass   [GROUP] DefinedObjectClass,
    objectSet     [GROUP] ObjectSet
}

```

```

ObjectSetReference ::= TypeReference

```

```

TopLevelNamedType ::= NamedType
    (WITH COMPONENTS { ...,
        component (WITH COMPONENTS { ...,
            definition (WITH COMPONENTS { ..., reference ABSENT })
        }),
        element   (WITH COMPONENTS { ...,
            definition (WITH COMPONENTS { ..., reference ABSENT })
        }),
        attribute (WITH COMPONENTS { ...,
            definition (WITH COMPONENTS { ..., reference ABSENT })
        }),
        group      ABSENT,
        member     ABSENT,
        item       ABSENT,
        simpleContent ABSENT })

```

```

NamedType ::= [SINGULAR-INSERTIONS] CHOICE {
    component      Element,
    element        Element,
    attribute      Attribute,
    group          InvisibleNamedType,
    member         InvisibleNamedType,
    item           InvisibleNamedType,
    simpleContent  InvisibleNamedType
}

```

```

Attribute ::= GenericNamedType
    (WITH COMPONENTS { ...,
        definition (WITH COMPONENTS { ...,
            local (WITH COMPONENTS { ...,
                typeAsVersion ABSENT }) }) })

```

```

Element ::= GenericNamedType
    (WITH COMPONENTS { ...,
        definition (WITH COMPONENTS { ...,
            local (WITH COMPONENTS { ...,
                versionIndicator ABSENT }) }) })

```

```

InvisibleNamedType ::= GenericNamedType
    (WITH COMPONENTS { ...,
        definition (WITH COMPONENTS { ...,
            reference ABSENT,
            local (WITH COMPONENTS { ...,
                typeAsVersion ABSENT,
                versionIndicator ABSENT }) }) })

GenericNamedType ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation Annotation OPTIONAL,
    identifier [ATTRIBUTE] IdentifierOrEmpty OPTIONAL,
    definition [GROUP] CHOICE {
        reference [GROUP] DefinedComponent,
        local [GROUP] LocalComponent
    }
}

IdentifierOrEmpty ::= UTF8String (INCLUDES Identifier | "")

DefinedComponent ::= [HOLLOW-INSERTIONS] SEQUENCE {
    name [GROUP] [NO-INSERTIONS] CHOICE {
        ref [ATTRIBUTE] QName,
        elementType [ATTRIBUTE] Name
    },
    namespace [ATTRIBUTE] AnyURI OPTIONAL,
    context [ATTRIBUTE] AnyURI OPTIONAL,
    embedded [ATTRIBUTE] BOOLEAN OPTIONAL,
    prefixes [GROUP] EncodingPrefixes OPTIONAL
}
(WITH COMPONENTS { ...,
    name (WITH COMPONENTS { ref PRESENT }),
    namespace ABSENT } |
WITH COMPONENTS { ...,
    name (WITH COMPONENTS { elementType PRESENT }),
    embedded ABSENT })

LocalComponent ::= SEQUENCE {
    name [ATTRIBUTE] NCName,
    typeAsVersion [ATTRIBUTE] BOOLEAN OPTIONAL,
    versionIndicator [ATTRIBUTE] BOOLEAN OPTIONAL,
    type [GROUP] Type
}

Type ::= [NO-INSERTIONS] CHOICE {
    typeRef [NAME AS "type"] [ATTRIBUTE] QName,
    type ElementFormType
}

```

```

ElementFormType ::= [HOLLOW-INSERTIONS] SEQUENCE {
  annotation Annotation OPTIONAL,
  explicit [ATTRIBUTE] BOOLEAN OPTIONAL,
  definition [GROUP] CHOICE {
    reference [GROUP] DefinedType,
    expanded ExpandedType,
    ancestor [ATTRIBUTE] INTEGER (1..MAX),
    namedBitList NamedBitList,
    namedNumberList NamedNumberList,
    enumerated EnumeratedType,
    tagged TaggedType,
    prefixed EncodingPrefixedType,
    selection SelectionType,
    instanceOf InstanceOfType,
    fromClass ObjectClassFieldType,
    fromObjects InformationFromObjects,
    sequence SequenceType,
    set SetType,
    choice ChoiceType,
    union UnionType,
    sequenceOf SequenceOfType,
    setOf SetOfType,
    list ListType,
    constrained ConstrainedType
  }
}

```

```

DefinedType ::= SEQUENCE {
  name [GROUP] [NO-INSERTIONS] CHOICE {
    ref [ATTRIBUTE] QName,
    elementType [ATTRIBUTE] Name
  },
  context [ATTRIBUTE] AnyURI OPTIONAL,
  embedded [ATTRIBUTE] BOOLEAN OPTIONAL
}
(WITH COMPONENTS { ...,
  name (WITH COMPONENTS { ref PRESENT }) } |
WITH COMPONENTS { ...,
  name (WITH COMPONENTS { elementType PRESENT } ),
  embedded ABSENT })

```

```

ExpandedType ::= SEQUENCE {
  name [ATTRIBUTE] NCName OPTIONAL,
  module ReferencedModule OPTIONAL,
  type [GROUP] Type
}

```

```

ReferencedModule ::= SEQUENCE {

```

```
    name          [ATTRIBUTE] ModuleReference OPTIONAL,  
    identifier    [ATTRIBUTE] DefinitiveIdentifier OPTIONAL,  
    schemaIdentity [ATTRIBUTE] AnyURI OPTIONAL  
}
```

NamedBitList ::= SEQUENCE SIZE (1..MAX) OF namedBit NamedBit

```
NamedBit ::= SEQUENCE {  
    name          [ATTRIBUTE] NCName,  
    identifier    [ATTRIBUTE] Identifier OPTIONAL,  
    bit           [ATTRIBUTE] INTEGER (0..MAX)  
}
```

NamedNumberList ::= SEQUENCE SIZE (1..MAX) OF
 namedNumber NamedNumber

```
NamedNumber ::= SEQUENCE {  
    name          [ATTRIBUTE] NCName,  
    identifier    [ATTRIBUTE] Identifier OPTIONAL,  
    number        [ATTRIBUTE] INTEGER  
}
```

```
EnumeratedType ::= SEQUENCE {  
    root          [GROUP] Enumeration,  
    extension     SEQUENCE {  
        exception ExceptionSpec OPTIONAL,  
        additions [GROUP] Enumeration OPTIONAL  
    } OPTIONAL  
}
```

Enumeration ::= SEQUENCE SIZE (1..MAX) OF
 enumeration EnumerationItem

```
EnumerationItem ::= SEQUENCE {  
    name          [ATTRIBUTE] NCName,  
    identifier    [ATTRIBUTE] Identifier OPTIONAL,  
    number        [ATTRIBUTE] INTEGER OPTIONAL  
}
```

```
Tag ::= SEQUENCE {  
    tagClass      [ATTRIBUTE] TagClass OPTIONAL,  
    number        [ATTRIBUTE] INTEGER (0..MAX),  
    tagging       [ATTRIBUTE] Tagging OPTIONAL  
}
```

```
TaggedType ::= SEQUENCE {  
    COMPONENTS OF Tag,  
    type [GROUP] Type
```



```

}

TagClass ::= ENUMERATED { universal, application, private }

Tagging ::= ENUMERATED { explicit, implicit }

EncodingPrefixedType ::= [HOLLOW-INSERTIONS] SEQUENCE {
    prefixes [GROUP] EncodingPrefixes,
    type      [GROUP] Type
}

EncodingPrefixes ::= SEQUENCE SIZE (1..MAX) OF
    prefix [GROUP] EncodingPrefix

EncodingPrefix ::= [SINGULAR-INSERTIONS] CHOICE {
    tag [NAME AS "TAG"] Tag,
    gser [NAME AS "GSER"] GSER-EncodingInstruction,
    xer [NAME AS "XER"] XER-EncodingInstruction
    -- plus encoding instructions
    -- for other encoding rules in the future
}

SelectionType ::= SEQUENCE {
    alternative [GROUP] [SINGULAR-INSERTIONS] CHOICE {
        component [ATTRIBUTE] QName,
        element [ATTRIBUTE] QName,
        attribute [ATTRIBUTE] QName,
        group [ATTRIBUTE] QName,
        member [ATTRIBUTE] QName
    },
    type [GROUP] Type
}

InstanceOfType ::= DefinedObjectClass

ObjectClassFieldType ::= SEQUENCE {
    objectClass [GROUP] DefinedObjectClass,
    fieldName [GROUP] fieldName
}

fieldName ::= [SINGULAR-INSERTIONS] CHOICE {
    fieldNameAtt [NAME AS "fieldName"]
        [ATTRIBUTE] PrimitiveFieldNames,
    fieldName PrimitiveFieldNames
}

PrimitiveFieldNames ::= UTF8String

```

```

InformationFromObjects ::= [HOLLOW-INSERTIONS] SEQUENCE {
    referencedObjects [GROUP] ReferencedObjects,
    fieldName         [GROUP] FieldName
}

ReferencedObjects ::= [SINGULAR-INSERTIONS] CHOICE {
    object      [GROUP] Object,
    objectSet   [GROUP] ObjectSet
}

Insertions ::=
    ENUMERATED { none, hollow, singular, uniform, multiform }

SequenceType ::= [HOLLOW-INSERTIONS] SEQUENCE {
    insertions      [ATTRIBUTE] Insertions OPTIONAL,
    root            [GROUP] ComponentTypeList OPTIONAL,
    extensionAndFinal [GROUP] [HOLLOW-INSERTIONS] SEQUENCE {
        extension      [HOLLOW-INSERTIONS] SEQUENCE {
            exception    ExceptionSpec OPTIONAL,
            additions     [GROUP] ExtensionAdditions OPTIONAL
        },
        root            [GROUP] ComponentTypeList OPTIONAL
    } OPTIONAL
}

ComponentTypeList ::= SEQUENCE SIZE (1..MAX) OF
    componentType [GROUP] ComponentType

ComponentType ::= [NO-INSERTIONS] CHOICE {
    component      [GROUP] SequenceNamedType,
    optional       SEQUENCE {
        component    [GROUP] SequenceNamedType,
        default      Value OPTIONAL
    },
    componentsOf   Type
}

SequenceNamedType ::= NamedType
    (WITH COMPONENTS { ..., member ABSENT, item ABSENT })

ExtensionAdditions ::= SEQUENCE SIZE (1..MAX) OF
    addition [GROUP] ExtensionAddition

ExtensionAddition ::= [NO-INSERTIONS] CHOICE {
    extensionGroup ExtensionAdditionGroup,
    componentType   [GROUP] ComponentType
}

```

```

ExtensionAdditionGroup ::= [HOLLOW-INSERTIONS] SEQUENCE {
    version          [ATTRIBUTE] VersionNumber OPTIONAL,
    componentTypes   [GROUP] ComponentTypeList
}

VersionNumber ::= INTEGER (2..MAX)

SetType ::= SequenceType

ChoiceOrUnionType ::= [HOLLOW-INSERTIONS] SEQUENCE {
    insertions       [ATTRIBUTE] Insertions OPTIONAL,
    precedence       [ATTRIBUTE] PrecedenceList OPTIONAL,
    root             [GROUP] AlternativeTypeList,
    extension        [HOLLOW-INSERTIONS] SEQUENCE {
        exception    ExceptionSpec OPTIONAL,
        additions    [GROUP] ExtensionAdditionAlternatives OPTIONAL
    } OPTIONAL
}

PrecedenceList ::= [LIST] SEQUENCE SIZE (1..MAX) OF member QName

AlternativeTypeList ::= SEQUENCE SIZE (1..MAX) OF
    component [GROUP] ChoiceOrUnionNamedType

ChoiceOrUnionNamedType ::= NamedType
    (WITH COMPONENTS { ..., item ABSENT, simpleContent ABSENT })

ExtensionAdditionAlternatives ::= SEQUENCE SIZE (1..MAX) OF
    addition [GROUP] ExtensionAdditionAlternative

ExtensionAdditionAlternative ::= [NO-INSERTIONS] CHOICE {
    extensionGroup   ExtensionAdditionAlternativesGroup,
    component        [GROUP] ChoiceOrUnionNamedType
}

ExtensionAdditionAlternativesGroup ::= [HOLLOW-INSERTIONS] SEQUENCE {
    version          [ATTRIBUTE] VersionNumber OPTIONAL,
    alternatives     [GROUP] AlternativeTypeList
}

ChoiceType ::= ChoiceOrUnionType
    (WITH COMPONENTS { ...,
        precedence ABSENT,
        root (WITH COMPONENT (INCLUDES ChoiceNamedType)),
        extension (WITH COMPONENTS { ...,
            additions (WITH COMPONENT (WITH COMPONENTS { ...,
                extensionGroup (WITH COMPONENTS { ...,
                    alternatives (WITH COMPONENT

```

```

        (INCLUDES ChoiceNamedType)) })),
    component (INCLUDES ChoiceNamedType) }))) })) }))

```

```

ChoiceNamedType ::= ChoiceOrUnionNamedType
    (WITH COMPONENTS { ..., member ABSENT })

```

```

UnionType ::= ChoiceOrUnionType
    (WITH COMPONENTS { ...,
        insertions ABSENT,
        root (WITH COMPONENT (INCLUDES UnionNamedType)),
        extension (WITH COMPONENTS { ...,
            additions (WITH COMPONENT (WITH COMPONENTS { ...,
                extensionGroup (WITH COMPONENTS { ...,
                    alternatives (WITH COMPONENT
                        (INCLUDES UnionNamedType)) })),
                component (INCLUDES UnionNamedType) }))) })) }))

```

```

UnionNamedType ::= ChoiceOrUnionNamedType
    (WITH COMPONENTS { ...,
        component ABSENT,
        element ABSENT,
        attribute ABSENT,
        group ABSENT })

```

```

SequenceOfOrListType ::= SEQUENCE {
    minSize    [ATTRIBUTE] INTEGER (0..MAX) OPTIONAL,
    maxSize    [ATTRIBUTE] INTEGER (0..MAX) OPTIONAL,
    component  [GROUP] NamedType
        (WITH COMPONENTS { ...,
            attribute ABSENT,
            member ABSENT,
            simpleContent ABSENT })
}

```

```

SequenceOfType ::= SequenceOfOrListType
    (WITH COMPONENTS { ...,
        component (WITH COMPONENTS { ..., item ABSENT }) })

```

```

SetOfType ::= SequenceOfType

```

```

ListType ::= SequenceOfOrListType
    (WITH COMPONENTS { ...,
        component (WITH COMPONENTS { ...,
            component ABSENT,
            element ABSENT,
            group ABSENT }) })

```

```

ConstrainedType ::= [HOLLOW-INSERTIONS] SEQUENCE {

```

```

    type          [GROUP] Type,
    constraint     [GROUP] Constraint
}

Constraint ::= SEQUENCE {
    constraintSpec [GROUP] [NO-INSERTIONS] CHOICE {
        subtype      [GROUP] ElementSetSpecs,
        constrainedBy UserDefinedConstraint,
        table         TableConstraint,
        contents      ContentsConstraint
    },
    exception        ExceptionSpec OPTIONAL
}

UserDefinedConstraint ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation Annotation OPTIONAL,
    parameters [GROUP] ConstraintParameters OPTIONAL
}

ConstraintParameters ::= SEQUENCE SIZE (1..MAX) OF
    parameter [GROUP] UserDefinedConstraintParameter

UserDefinedConstraintParameter ::= [SINGULAR-INSERTIONS] CHOICE {
    valueParameter SEQUENCE {
        type      [GROUP] Type,
        value      [GROUP] Value
    },
    valueSetParameter SEQUENCE {
        type      [GROUP] Type,
        valueSet   [GROUP] ValueSet
    },
    objectParameter SEQUENCE {
        objectClass [GROUP] DefinedObjectClass,
        object      [GROUP] Object
    },
    objectSetParameter SEQUENCE {
        objectClass [GROUP] DefinedObjectClass,
        objectSet   [GROUP] ObjectSet
    },
    typeParameter SEQUENCE {
        type      [GROUP] Type
    },
    classParameter SEQUENCE {
        objectClass [GROUP] DefinedObjectClass
    }
}

TableConstraint ::= SEQUENCE {

```

```

    objectSet          [GROUP] ObjectSet,
    componentRelation  [GROUP] AtNotations OPTIONAL
}

AtNotations ::= SEQUENCE SIZE (1..MAX) OF
    restrictBy AtNotation

AtNotation ::= Markup

ContentsConstraint ::= SEQUENCE {
    containing  Type OPTIONAL,
    encodedBy   Value OPTIONAL
} ((WITH COMPONENTS { ..., containing PRESENT }) |
   (WITH COMPONENTS { ..., encodedBy PRESENT })))

ExceptionSpec ::= SEQUENCE {
    type   [GROUP] Type,
    value  [GROUP] Value
}

Value ::= [NO-INSERTIONS] CHOICE {
    literalValueAtt [NAME AS "literalValue"] [ATTRIBUTE] UTF8String,
    literalValue    ElementFormLiteralValue,
    valueRef        [NAME AS "value"] [ATTRIBUTE] QName,
    value           ElementFormNotationalValue
}

ElementFormLiteralValue ::= Markup
-- If asnx:literal="false" then the governing type of
-- ElementFormLiteralValue is ElementFormNotationalValue.

ElementFormNotationalValue ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation  Annotation OPTIONAL,
    definition  [GROUP] [NO-INSERTIONS] CHOICE {
        reference      [GROUP] Reference,
        expanded       ExpandedValue,
        fromObjects    InformationFromObjects,
        openTypeValue  SEQUENCE {
            type        [GROUP] Type,
            value       [GROUP] Value
        },
        components     [GROUP] ComponentValueList
    }
}

Reference ::= SEQUENCE {
    ref        [ATTRIBUTE] QName,
    context    [ATTRIBUTE] AnyURI OPTIONAL
}

```

```

}

ExpandedValue ::= SEQUENCE {
    name      [ATTRIBUTE] NCName OPTIONAL,
    module    ReferencedModule OPTIONAL,
    value     [GROUP] Value
}

ComponentValueList ::= SEQUENCE SIZE (1..MAX) OF
    component [GROUP] NamedValue

NamedValue ::= [SINGULAR-INSERTIONS] CHOICE {
    component      GenericNamedValue,
    element        GenericNamedValue,
    attribute      GenericNamedValue,
    group          GenericNamedValue,
    member         GenericNamedValue,
    item           GenericNamedValue,
    simpleContent  GenericNamedValue
}

GenericNamedValue ::= SEQUENCE {
    name  [ATTRIBUTE] QName,
    value [GROUP] Value
}

ValueSet ::= [NO-INSERTIONS] CHOICE {
    valueSetRef [NAME AS "valueSet"] [ATTRIBUTE] QName,
    -- valueSetRef is not used in this version
    valueSet    ElementFormValueSet
}

ElementFormValueSet ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation Annotation OPTIONAL,
    definition [GROUP] [NO-INSERTIONS] CHOICE {
        elementSetSpecs [GROUP] ElementSetSpecs
    }
}

ElementSetSpecs ::= [HOLLOW-INSERTIONS] SEQUENCE {
    root      [GROUP] ValueElementSetSpec,
    extension [HOLLOW-INSERTIONS] SEQUENCE {
        additions [GROUP] ValueElementSetSpec OPTIONAL
    } OPTIONAL
}

ValueElementSetSpec ::= ElementSetSpec
(WITH COMPONENTS { ...,

```

```

    object      ABSENT,
    objectSet   ABSENT,
    union       (WITH COMPONENT (INCLUDES ValueElementSetSpec)),
    intersection (WITH COMPONENT (INCLUDES ValueElementSetSpec)),
    all         (WITH COMPONENTS { ...,
        elements (INCLUDES ValueElementSetSpec),
        except   (INCLUDES ValueElementSetSpec) }) })

ElementSetSpec ::= [SINGULAR-INSERTIONS] CHOICE {
    literalValue  ElementFormLiteralValue,
    value         ElementFormNotationalValue,
    includes      Type,
    range         ValueRange,
    size          Constraint,
    typeConstraint Type,
    from          Constraint,
    withComponent Constraint,
    withComponents MultipleTypeConstraints,
    pattern       Value,
    object        ElementFormObject,
    objectSet     ElementFormObjectSet,
    union         ElementSetSpecList,
    intersection  ElementSetSpecList,
    all          SEQUENCE {
        elements [GROUP] ElementSetSpec OPTIONAL,
        except   ElementSetSpec
    }
}

ElementSetSpecList ::= SEQUENCE SIZE (2..MAX) OF
    elements [GROUP] ElementSetSpec

ValueRange ::= SEQUENCE {
    minimum [GROUP] [NO-INSERTIONS] CHOICE {
        minInclusive EndValue,
        minExclusive EndValue
    } DEFAULT minInclusive:{},
    maximum [GROUP] [NO-INSERTIONS] CHOICE {
        maxInclusive EndValue,
        maxExclusive EndValue
    } DEFAULT maxInclusive:{}
}

EndValue ::= [HOLLOW-INSERTIONS] SEQUENCE {
    value [GROUP] Value OPTIONAL
}

MultipleTypeConstraints ::= [HOLLOW-INSERTIONS] SEQUENCE {

```



```

    partial          [ATTRIBUTE] BOOLEAN DEFAULT FALSE,
    typeConstraints  [GROUP] TypeConstraints
}

TypeConstraints ::= SEQUENCE SIZE (1..MAX) OF
    namedConstraint [GROUP] NamedConstraint

NamedConstraint ::= [SINGULAR-INSERTIONS] CHOICE {
    component      GenericNamedConstraint,
    element        GenericNamedConstraint,
    attribute      GenericNamedConstraint,
    group          GenericNamedConstraint,
    member         GenericNamedConstraint,
    item           GenericNamedConstraint,
    simpleContent  GenericNamedConstraint
}

GenericNamedConstraint ::= [HOLLOW-INSERTIONS] SEQUENCE {
    name          [ATTRIBUTE] QName,
    use           [ATTRIBUTE] PresenceConstraint OPTIONAL,
    constraint     [GROUP] Constraint OPTIONAL
}

PresenceConstraint ::= ENUMERATED { present, absent, optional }

ObjectClass ::= [SINGULAR-INSERTIONS] CHOICE {
    classRef [NAME AS "class"] [ATTRIBUTE] QName,
    class    ElementFormObjectClass
}

DefinedObjectClass ::= ObjectClass
    (WITH COMPONENTS { ...,
        class (WITH COMPONENTS { ...,
            definition (WITH COMPONENTS { ...,
                objectClassDefn ABSENT }) }) })

ElementFormObjectClass ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation Annotation OPTIONAL,
    definition [GROUP] [NO-INSERTIONS] CHOICE {
        reference [GROUP] Reference,
        expanded  ExpandedObjectClass,
        objectClassDefn [GROUP] ObjectClassDefn
    }
}

ExpandedObjectClass ::= SEQUENCE {
    name [ATTRIBUTE] NCName OPTIONAL,
    module ReferencedModule OPTIONAL,

```

```

    objectClass [GROUP] ObjectClass
}

ObjectClassDefn ::= SEQUENCE SIZE (1..MAX) OF
    fieldSpec [GROUP] FieldSpec

FieldSpec ::= [SINGULAR-INSERTIONS] CHOICE {
    typeField      TypeField,
    valueField     ValueField,
    valueSetField  ValueSetField,
    objectField    ObjectField,
    objectSetField ObjectSetField,
    optional       OptionalField
}

OptionalField ::= SEQUENCE {
    field [GROUP] [SINGULAR-INSERTIONS] CHOICE {
        typeField      TypeField,
        valueField     ValueField,
        valueSetField  ValueSetField,
        objectField    ObjectField,
        objectSetField ObjectSetField
    },
    default Setting OPTIONAL
} (WITH COMPONENTS { ...,
    field (WITH COMPONENTS { typeField PRESENT })),
    default (WITH COMPONENTS { ...,
        value ABSENT,
        valueSet ABSENT,
        object ABSENT,
        objectSet ABSENT } ) } |
    WITH COMPONENTS { ...,
        field (WITH COMPONENTS { valueField PRESENT })),
        default (WITH COMPONENTS { ...,
            type ABSENT,
            valueSet ABSENT,
            object ABSENT,
            objectSet ABSENT } ) } |
    WITH COMPONENTS { ...,
        field (WITH COMPONENTS { valueSetField PRESENT })),
        default (WITH COMPONENTS { ...,
            type ABSENT,
            value ABSENT,
            object ABSENT,
            objectSet ABSENT } ) } |
    WITH COMPONENTS { ...,
        field (WITH COMPONENTS { objectField PRESENT })),
        default (WITH COMPONENTS { ...,

```

```

        type ABSENT,
        value ABSENT,
        valueSet ABSENT,
        objectSet ABSENT }) } |
WITH COMPONENTS { ...,
    field (WITH COMPONENTS { objectSetField PRESENT }),
    default (WITH COMPONENTS { ...,
        type ABSENT,
        value ABSENT,
        valueSet ABSENT,
        object ABSENT }) })

TypeField ::= SEQUENCE {
    annotation Annotation OPTIONAL,
    name       [ATTRIBUTE] TypeFieldReference
}

TypeFieldReference ::= TypeReference

ValueField ::= SEQUENCE {
    annotation Annotation OPTIONAL,
    name       [ATTRIBUTE] ValueFieldReference,
    unique     [ATTRIBUTE] BOOLEAN OPTIONAL,
    governor   [GROUP] [SINGULAR-INSERTIONS] CHOICE {
        type           [GROUP] Type,
        typeFromField  FileName
    }
}
| ((WITH COMPONENTS { ..., unique ABSENT }) |
   (WITH COMPONENTS { ...,
       governor (WITH COMPONENTS { ..., typeFromField ABSENT }) })))

ValueFieldReference ::= ValueReference

ValueSetField ::= SEQUENCE {
    annotation Annotation OPTIONAL,
    name       [ATTRIBUTE] ValueSetFieldReference,
    governor   [GROUP] [SINGULAR-INSERTIONS] CHOICE {
        type           [GROUP] Type,
        typeFromField  FileName
    }
}

ValueSetFieldReference ::= TypeReference

ObjectField ::= SEQUENCE {
    annotation Annotation OPTIONAL,
    name       [ATTRIBUTE] ObjectFieldReference,
    objectClass [GROUP] DefinedObjectClass
}

```

```

}

ObjectFieldReference ::= ObjectReference

ObjectSetField ::= SEQUENCE {
    annotation    Annotation OPTIONAL,
    name          [ATTRIBUTE] ObjectSetFieldReference,
    objectClass   [GROUP] DefinedObjectClass
}

ObjectSetFieldReference ::= ObjectSetReference

Object ::= [NO-INSERTIONS] CHOICE {
    objectRef    [NAME AS "object"] [ATTRIBUTE] QName,
    object       ElementFormObject
}

ElementFormObject ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation    Annotation OPTIONAL,
    definition    [GROUP] [SINGULAR-INSERTIONS] CHOICE {
        reference [GROUP] Reference,
        expanded   ExpandedObject,
        fromObjects InformationFromObjects,
        fields     [GROUP] ObjectDefn
    }
}

ExpandedObject ::= SEQUENCE {
    name          [ATTRIBUTE] NCName OPTIONAL,
    module        ReferencedModule OPTIONAL,
    object        [GROUP] Object
}

ObjectDefn ::= SEQUENCE OF field FieldSetting

FieldSetting ::= [HOLLOW-INSERTIONS] SEQUENCE {
    name          [ATTRIBUTE] NCName,
    setting       [GROUP] Setting
}

Setting ::= CHOICE {
    type          [GROUP] Type,
    value         [GROUP] Value,
    valueSet      [GROUP] ValueSet,
    object        [GROUP] Object,
    objectSet     [GROUP] ObjectSet
}

```

```

ObjectSet ::= [NO-INSERTIONS] CHOICE {
    objectSetRef  [NAME AS "objectSet"] [ATTRIBUTE] QName,
    objectSet      ElementFormObjectSet
}

```

```

ElementFormObjectSet ::= [HOLLOW-INSERTIONS] SEQUENCE {
    annotation  Annotation OPTIONAL,
    definition  [GROUP] [NO-INSERTIONS] CHOICE {
        reference      [GROUP] Reference,
        expanded       ExpandedObjectSet,
        objectSetSpec  [GROUP] ObjectSetSpec,
        fromObjects    InformationFromObjects
    }
}

```

```

ExpandedObjectSet ::= SEQUENCE {
    name          [ATTRIBUTE] NCName OPTIONAL,
    module        ReferencedModule OPTIONAL,
    objectSet     [GROUP] ObjectSet
}

```

```

ObjectSetSpec ::= [HOLLOW-INSERTIONS] SEQUENCE {
    root          [GROUP] ObjectElementSetSpec OPTIONAL,
    extension     [HOLLOW-INSERTIONS] SEQUENCE {
        additions  [GROUP] ObjectElementSetSpec OPTIONAL
    } OPTIONAL
} ((WITH COMPONENTS { ..., root PRESENT }) |
   (WITH COMPONENTS { ..., extension PRESENT }))

```

```

ObjectElementSetSpec ::= ElementSetSpec
(WITH COMPONENTS { ...,
    literalValue  ABSENT,
    value         ABSENT,
    includes      ABSENT,
    range         ABSENT,
    size          ABSENT,
    typeConstraint ABSENT,
    from          ABSENT,
    withComponent ABSENT,
    withComponents ABSENT,
    pattern       ABSENT,
    union         (WITH COMPONENT (INCLUDES ObjectElementSetSpec)),
    intersection  (WITH COMPONENT (INCLUDES ObjectElementSetSpec)),
    all           (WITH COMPONENTS { ...,
        elements    (INCLUDES ObjectElementSetSpec),
        except      (INCLUDES ObjectElementSetSpec) }) })

```

```

EncodingControlSections ::= SEQUENCE SIZE (1..MAX) OF

```

section [GROUP] EncodingControlSection

```
EncodingControlSection ::= [SINGULAR-INSERTIONS] CHOICE {
  gser  [NAME AS "GSER"] GSER-EncodingInstructionAssignmentList,
  xer   [NAME AS "XER"] XER-EncodingInstructionAssignmentList
  -- plus encoding control sections
  -- for other encoding rules in the future
}
```

ENCODING-CONTROL RXER

```
SCHEMA-IDENTITY  "urn:oid:1.3.6.1.4.1.21472.1.0.1"
TARGET-NAMESPACE "urn:ietf:params:xml:ns:asn1" PREFIX "asn1"
```

COMPONENT module ModuleDefinition

COMPONENT literal [ATTRIBUTE] BOOLEAN

END

Appendix B. ASN.1 for ASN.1

This appendix is non-normative.

```
<?xml version="1.0"?>
<asn1:module xmlns:asn1="urn:ietf:params:xml:ns:asn1"
  name="AbstractSyntaxNotation-1"
  identifier="1.3.6.1.4.1.21472.1.0.1"
  schemaIdentity="urn:oid:1.3.6.1.4.1.21472.1.0.1"
  targetNamespace="urn:ietf:params:xml:ns:asn1"
  targetPrefix="asn1"
  extensibilityImplied="true">
```

<annotation>

Copyright (C) The IETF Trust (2007). This version of this ASN.1 module is part of RFC 4912; see the RFC itself for full legal notices.

Regarding this ASN.1 module or any portion of it, the author makes no guarantees and is not responsible for any damage resulting from its use. The author grants 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.

</annotation>

```
<import name="GSER-EncodingInstructionNotation"
  identifier="1.3.6.1.4.1.21472.1.0.2"
  schemaIdentity="urn:oid:1.3.6.1.4.1.21472.1.0.2"
  namespace="urn:ietf:params:xml:ns:asn1"/>

<import name="XER-EncodingInstructionNotation"
  identifier="1.3.6.1.4.1.21472.1.0.3"
  schemaIdentity="urn:oid:1.3.6.1.4.1.21472.1.0.3"
  namespace="urn:ietf:params:xml:ns:asn1"/>

<namedType name="ModuleDefinition">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn1:Annotation"/>
      </optional>
      <optional>
        <attribute name="format" versionIndicator="true">
          <type>
            <constrained type="asn1:UTF8String">
              <literalValue>1.0</literalValue>
              <extension/>
            </constrained>
          </type>
        </attribute>
        <default literalValue="1.0"/>
      </optional>
      <attribute name="name" type="asn1:ModuleReference"/>
      <optional>
        <attribute name="identifier" type="asn1:DefinitiveIdentifier"/>
      </optional>
      <optional>
        <attribute name="schemaIdentity" type="asn1:AnyURI"/>
      </optional>
      <optional>
        <attribute name="targetNamespace" type="asn1:AnyURI"/>
      </optional>
      <optional>
        <attribute name="targetPrefix" type="asn1:NCName"/>
      </optional>
      <optional>
        <attribute name="tagDefault" type="asn1:TagDefault"/>
        <default literalValue="automatic"/>
      </optional>
      <optional>
        <attribute name="extensibilityImplied" type="asn1:BOOLEAN"/>
        <default literalValue="false"/>
      </optional>
    </sequence>
  </type>
</namedType>
```

```
<optional>
  <element name="export">
    <annotation> export is not used in this version </annotation>
    <type>
      <sequence/>
    </type>
  </element>
</optional>
<optional>
  <group name="imports" type="asn:ImportList"/>
</optional>
<optional>
  <group name="assignments" type="asn:AssignmentList"/>
</optional>
<optional>
  <element name="encodingControls"
    type="asn:EncodingControlSections"/>
</optional>
</sequence>
</type>
</namedType>

<namedType name="ModuleReference" type="asn:TypeReference"/>

<namedType name="DefinitiveIdentifier"
  type="asn:OBJECT-IDENTIFIER"/>

<namedType name="TagDefault">
  <type>
    <enumerated>
      <enumeration name="explicit"/>
      <enumeration name="implicit"/>
      <enumeration name="automatic"/>
    </enumerated>
  </type>
</namedType>

<namedType name="Annotation" type="asn:Markup"/>

<namedType name="ImportList">
  <type>
    <sequenceOf minSize="1">
      <element name="import" type="asn:Import"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="Import">
```



```

<type>
  <sequence>
    <optional>
      <attribute name="name" type="asn:ModuleReference"/>
    </optional>
    <optional>
      <attribute name="identifier" type="asn:DefinitiveIdentifier"/>
    </optional>
    <optional>
      <attribute name="schemaIdentity" type="asn:AnyURI"/>
    </optional>
    <optional>
      <attribute name="namespace" type="asn:AnyURI"/>
    </optional>
    <optional>
      <attribute name="schemaLocation" type="asn:AnyURI"/>
    </optional>
  </sequence>
</type>
</namedType>

<namedType name="AssignmentList">
  <type>
    <sequenceOf minSize="1">
      <group name="assignment" type="asn:Assignment"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="Assignment">
  <type>
    <choice insertions="none">
      <element name="namedType" type="asn:TypeAssignment"/>
      <element name="namedValue" type="asn:ValueAssignment"/>
      <element name="namedValueSet"
        type="asn:ValueSetTypeAssignment"/>
      <element name="namedClass" type="asn:ObjectClassAssignment"/>
      <element name="namedObject" type="asn:ObjectAssignment"/>
      <element name="namedObjectSet" type="asn:ObjectSetAssignment"/>
      <group name="component" type="asn:TopLevelNamedType"/>
    </choice>
  </type>
</namedType>

<namedType name="TypeAssignment">
  <type>
    <sequence>
      <optional>

```

```

    <element name="annotation" type="asn:Annotation"/>
  </optional>
  <attribute name="name" type="asn:TypeReference"/>
  <group name="type" type="asn:Type"/>
</sequence>
</type>
</namedType>

<namedType name="TypeReference">
  <type>
    <constrained type="asn:UTF8String">
      <pattern literalValue="[A-Z]\w*(-\w+)*"/>
      <!-- \w is equivalent to [a-zA-Z0-9] -->
    </constrained>
  </type>
</namedType>

<namedType name="ValueAssignment">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <attribute name="name" type="asn:ValueReference"/>
      <group name="type" type="asn:Type"/>
      <group name="value" type="asn:Value"/>
    </sequence>
  </type>
</namedType>

<namedType name="ValueReference" type="asn:Identifier"/>

<namedType name="Identifier">
  <type>
    <constrained type="asn:UTF8String">
      <pattern literalValue="[a-z]\w*(-\w+)*"/>
    </constrained>
  </type>
</namedType>

<namedType name="ValueSetTypeAssignment">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <attribute name="name" type="asn:TypeReference"/>
      <group name="type" type="asn:Type"/>

```

```
<group name="valueSet" type="asn1:ValueSet"/>
</sequence>
</type>
</namedType>

<namedType name="ObjectClassAssignment">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn1:Annotation"/>
      </optional>
      <attribute name="name" type="asn1:ObjectClassReference"/>
      <group name="objectClass" type="asn1:ObjectClass"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectClassReference">
  <type>
    <constrained type="asn1:UTF8String">
      <pattern literalValue="[A-Z][A-Z0-9]*(-[A-Z0-9]+)*"/>
    </constrained>
  </type>
</namedType>

<namedType name="ObjectAssignment">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn1:Annotation"/>
      </optional>
      <attribute name="name" type="asn1:ObjectReference"/>
      <group name="objectClass" type="asn1:DefinedObjectClass"/>
      <group name="object" type="asn1:Object"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectReference" type="asn1:ValueReference"/>

<namedType name="ObjectSetAssignment">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn1:Annotation"/>
      </optional>
      <attribute name="name" type="asn1:ObjectSetReference"/>
      <group name="objectClass" type="asn1:DefinedObjectClass"/>
    </sequence>
  </type>
</namedType>
```

```
<group name="objectSet" type="asn:ObjectSet"/>
</sequence>
</type>
</namedType>

<namedType name="ObjectSetReference" type="asn:TypeReference"/>

<namedType name="TopLevelNamedType">
  <type>
    <constrained type="asn:NamedType">
      <withComponents partial="true">
        <element name="component">
          <withComponents partial="true">
            <group name="definition">
              <withComponents partial="true">
                <group name="reference" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </element>
        <element name="element">
          <withComponents partial="true">
            <group name="definition">
              <withComponents partial="true">
                <group name="reference" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </element>
        <element name="attribute">
          <withComponents partial="true">
            <group name="definition">
              <withComponents partial="true">
                <group name="reference" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </element>
        <element name="group" use="absent"/>
        <element name="member" use="absent"/>
        <element name="item" use="absent"/>
        <element name="simpleContent" use="absent"/>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="NamedType">
```

```
<type>
  <choice insertions="singular">
    <element name="component" type="asn:Element"/>
    <element name="element" type="asn:Element"/>
    <element name="attribute" type="asn:Attribute"/>
    <element name="group" type="asn:InvisibleNamedType"/>
    <element name="member" type="asn:InvisibleNamedType"/>
    <element name="item" type="asn:InvisibleNamedType"/>
    <element name="simpleContent" type="asn:InvisibleNamedType"/>
  </choice>
</type>
</namedType>

<namedType name="Attribute">
  <type>
    <constrained type="asn:GenericNamedType">
      <withComponents partial="true">
        <group name="definition">
          <withComponents partial="true">
            <group name="local">
              <withComponents partial="true">
                <attribute name="typeAsVersion" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </group>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="Element">
  <type>
    <constrained type="asn:GenericNamedType">
      <withComponents partial="true">
        <group name="definition">
          <withComponents partial="true">
            <group name="local">
              <withComponents partial="true">
                <attribute name="versionIndicator" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </group>
      </withComponents>
    </constrained>
  </type>
</namedType>
```

```
<namedType name="InvisibleNamedType">
  <type>
    <constrained type="asn:GenericNamedType">
      <withComponents partial="true">
        <group name="definition">
          <withComponents partial="true">
            <group name="reference" use="absent"/>
            <group name="local">
              <withComponents partial="true">
                <attribute name="typeAsVersion" use="absent"/>
                <attribute name="versionIndicator" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </group>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="GenericNamedType">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <optional>
        <attribute name="identifier" type="asn:IdentifierOrEmpty"/>
      </optional>
      <group name="definition">
        <type>
          <choice>
            <group name="reference" type="asn:DefinedComponent"/>
            <group name="local" type="asn:LocalComponent"/>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>

<namedType name="IdentifierOrEmpty">
  <type>
    <constrained type="asn:UTF8String">
      <union>
        <includes type="asn:Identifier"/>
        <literalValue></literalValue>
      </union>
    </constrained>
  </type>
</namedType>
```

```
    </constrained>
  </type>
</namedType>

<namedType name="DefinedComponent">
  <type>
    <constrained>
      <type>
        <sequence insertions="hollow">
          <group name="name">
            <type>
              <choice insertions="none">
                <attribute name="ref" type="asn:QName"/>
                <attribute name="elementType" type="asn:Name"/>
              </choice>
            </type>
          </group>
          <optional>
            <attribute name="namespace" type="asn:AnyURI"/>
          </optional>
          <optional>
            <attribute name="context" type="asn:AnyURI"/>
          </optional>
          <optional>
            <attribute name="embedded" type="asn:BOOLEAN"/>
          </optional>
          <optional>
            <group name="prefixes" type="asn:EncodingPrefixes"/>
          </optional>
        </sequence>
      </type>
    </type>
  <union>
    <withComponents partial="true">
      <group name="name">
        <withComponents>
          <attribute name="ref" use="present"/>
        </withComponents>
      </group>
      <attribute name="namespace" use="absent"/>
    </withComponents>
    <withComponents partial="true">
      <group name="name">
        <withComponents>
          <attribute name="elementType" use="present"/>
        </withComponents>
      </group>
      <attribute name="embedded" use="absent"/>
    </withComponents>
  </union>
</namedType>
```

```
    </union>
  </constrained>
</type>
</namedType>

<namedType name="LocalComponent">
  <type>
    <sequence>
      <attribute name="name" type="asn:NCName"/>
      <optional>
        <attribute name="typeAsVersion" type="asn:BOOLEAN"/>
      </optional>
      <optional>
        <attribute name="versionIndicator" type="asn:BOOLEAN"/>
      </optional>
      <group name="type" type="asn:Type"/>
    </sequence>
  </type>
</namedType>

<namedType name="Type">
  <type>
    <choice insertions="none">
      <attribute name="type" identifier="typeRef" type="asn:QName"/>
      <element name="type" type="asn:ElementFormType"/>
    </choice>
  </type>
</namedType>

<namedType name="ElementFormType">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <optional>
        <attribute name="explicit" type="asn:BOOLEAN"/>
      </optional>
      <group name="definition">
        <type>
          <choice>
            <group name="reference" type="asn:DefinedType"/>
            <element name="expanded" type="asn:ExpandedType"/>
            <attribute name="ancestor">
              <type>
                <constrained type="asn:INTEGER">
                  <range>
                    <minInclusive literalValue="1"/>

```



```

        </range>
      </constrained>
    </type>
  </attribute>
  <element name="namedBitList" type="asn1:NamedBitList"/>
  <element name="namedNumberList" type="asn1:NamedNumberList"/>
  <element name="enumerated" type="asn1:EnumeratedType"/>
  <element name="tagged" type="asn1:TaggedType"/>
  <element name="prefixed" type="asn1:EncodingPrefixedType"/>
  <element name="selection" type="asn1:SelectionType"/>
  <element name="instanceOf" type="asn1:InstanceOfType"/>
  <element name="fromClass" type="asn1:ObjectClassFieldType"/>
  <element name="fromObjects"
    type="asn1:InformationFromObjects"/>
  <element name="sequence" type="asn1:SequenceType"/>
  <element name="set" type="asn1:SetType"/>
  <element name="choice" type="asn1:ChoiceType"/>
  <element name="union" type="asn1:UnionType"/>
  <element name="sequenceOf" type="asn1:SequenceOfType"/>
  <element name="setOf" type="asn1:SetOfType"/>
  <element name="list" type="asn1:ListType"/>
  <element name="constrained" type="asn1:ConstrainedType"/>
</choice>
</type>
</group>
</sequence>
</type>
</namedType>

<namedType name="DefinedType">
  <type>
    <constrained>
      <type>
        <sequence>
          <group name="name">
            <type>
              <choice insertions="none">
                <attribute name="ref" type="asn1:QName"/>
                <attribute name="elementType" type="asn1:Name"/>
              </choice>
            </type>
          </group>
        </sequence>
      </type>
    </constrained>
  </type>
  <optional>
    <attribute name="context" type="asn1:AnyURI"/>
  </optional>
  <optional>
    <attribute name="embedded" type="asn1:BOOLEAN"/>
  </optional>
</namedType>

```

```
    </sequence>
  </type>
  <union>
    <withComponents partial="true">
      <group name="name">
        <withComponents>
          <attribute name="ref" use="present"/>
        </withComponents>
      </group>
    </withComponents>
    <withComponents partial="true">
      <group name="name">
        <withComponents>
          <attribute name="elementType" use="present"/>
        </withComponents>
      </group>
      <attribute name="embedded" use="absent"/>
    </withComponents>
  </union>
</constrained>
</type>
</namedType>

<namedType name="ExpandedType">
  <type>
    <sequence>
      <optional>
        <attribute name="name" type="asn:NCName"/>
      </optional>
      <optional>
        <element name="module" type="asn:ReferencedModule"/>
      </optional>
      <group name="type" type="asn:Type"/>
    </sequence>
  </type>
</namedType>

<namedType name="ReferencedModule">
  <type>
    <sequence>
      <optional>
        <attribute name="name" type="asn:ModuleReference"/>
      </optional>
      <optional>
        <attribute name="identifier" type="asn:DefinitiveIdentifier"/>
      </optional>
      <optional>
        <attribute name="schemaIdentity" type="asn:AnyURI"/>
      </optional>
    </sequence>
  </type>
</namedType>
```

```
    </optional>
  </sequence>
</type>
</namedType>

<namedType name="NamedBitList">
  <type>
    <sequenceOf minSize="1">
      <element name="namedBit" type="asn:NamedBit"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="NamedBit">
  <type>
    <sequence>
      <attribute name="name" type="asn:NCName"/>
      <optional>
        <attribute name="identifier" type="asn:Identifier"/>
      </optional>
      <attribute name="bit">
        <type>
          <constrained type="asn:INTEGER">
            <range>
              <minInclusive literalValue="0"/>
            </range>
          </constrained>
        </type>
      </attribute>
    </sequence>
  </type>
</namedType>

<namedType name="NamedNumberList">
  <type>
    <sequenceOf minSize="1">
      <element name="namedNumber" type="asn:NamedNumber"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="NamedNumber">
  <type>
    <sequence>
      <attribute name="name" type="asn:NCName"/>
      <optional>
        <attribute name="identifier" type="asn:Identifier"/>
      </optional>
    </sequence>
  </type>
</namedType>
```

```
<attribute name="number" type="asn:INTEGER"/>
</sequence>
</type>
</namedType>

<namedType name="EnumeratedType">
  <type>
    <sequence>
      <group name="root" type="asn:Enumeration"/>
      <optional>
        <element name="extension">
          <type>
            <sequence>
              <optional>
                <element name="exception" type="asn:ExceptionSpec"/>
              </optional>
              <optional>
                <group name="additions" type="asn:Enumeration"/>
              </optional>
            </sequence>
          </type>
        </element>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="Enumeration">
  <type>
    <sequenceOf minSize="1">
      <element name="enumeration" type="asn:EnumerationItem"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="EnumerationItem">
  <type>
    <sequence>
      <attribute name="name" type="asn:NCName"/>
      <optional>
        <attribute name="identifier" type="asn:Identifier"/>
      </optional>
      <optional>
        <attribute name="number" type="asn:INTEGER"/>
      </optional>
    </sequence>
  </type>
</namedType>
```

```
<namedType name="Tag">
  <type>
    <sequence>
      <optional>
        <attribute name="tagClass" type="asn:TagClass"/>
      </optional>
      <attribute name="number">
        <type>
          <constrained type="asn:INTEGER">
            <range>
              <minInclusive literalValue="0"/>
            </range>
          </constrained>
        </type>
      </attribute>
    </sequence>
  </type>
</namedType>

<namedType name="TaggedType">
  <type>
    <sequence>
      <componentsOf type="asn:Tag"/>
      <group name="type" type="asn:Type"/>
    </sequence>
  </type>
</namedType>

<namedType name="TagClass">
  <type>
    <enumerated>
      <enumeration name="universal"/>
      <enumeration name="application"/>
      <enumeration name="private"/>
    </enumerated>
  </type>
</namedType>

<namedType name="Tagging">
  <type>
    <enumerated>
      <enumeration name="explicit"/>
      <enumeration name="implicit"/>
    </enumerated>
  </type>
</namedType>
```

```
</namedType>

<namedType name="EncodingPrefixedType">
  <type>
    <sequence insertions="hollow">
      <group name="prefixes" type="asn:EncodingPrefixes"/>
      <group name="type" type="asn:Type"/>
    </sequence>
  </type>
</namedType>

<namedType name="EncodingPrefixes">
  <type>
    <sequenceOf minSize="1">
      <group name="prefix" type="asn:EncodingPrefix"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="EncodingPrefix">
  <type>
    <choice insertions="singular">
      <element name="TAG" identifier="tag" type="asn:Tag"/>
      <element name="GSER" identifier="gser"
        type="asn:GSER-EncodingInstruction"/>
      <element name="XER" identifier="xer"
        type="asn:XER-EncodingInstruction"/>
      <!-- plus encoding instructions
        for other encoding rules in the future -->
    </choice>
  </type>
</namedType>

<namedType name="SelectionType">
  <type>
    <sequence>
      <group name="alternative">
        <type>
          <choice insertions="singular">
            <attribute name="component" type="asn:QName"/>
            <attribute name="element" type="asn:QName"/>
            <attribute name="attribute" type="asn:QName"/>
            <attribute name="group" type="asn:QName"/>
            <attribute name="member" type="asn:QName"/>
          </choice>
        </type>
      </group>
      <group name="type" type="asn:Type"/>
    </sequence>
  </type>
</namedType>
```

```
    </sequence>
  </type>
</namedType>

<namedType name="InstanceOfType" type="asn1:DefinedObjectClass"/>

<namedType name="ObjectClassFieldType">
  <type>
    <sequence>
      <group name="objectClass" type="asn1:DefinedObjectClass"/>
      <group name="fieldName" type="asn1:FieldName"/>
    </sequence>
  </type>
</namedType>

<namedType name="FieldName">
  <type>
    <choice insertions="singular">
      <attribute name="fieldName" identifier="fieldNameAtt"
        type="asn1:PrimitiveFieldNames"/>
      <element name="fieldName" type="asn1:PrimitiveFieldNames"/>
    </choice>
  </type>
</namedType>

<namedType name="PrimitiveFieldNames" type="asn1:UTF8String"/>

<namedType name="InformationFromObjects">
  <type>
    <sequence insertions="hollow">
      <group name="referencedObjects" type="asn1:ReferencedObjects"/>
      <group name="fieldName" type="asn1:FieldName"/>
    </sequence>
  </type>
</namedType>

<namedType name="ReferencedObjects">
  <type>
    <choice insertions="singular">
      <group name="object" type="asn1:Object"/>
      <group name="objectSet" type="asn1:ObjectSet"/>
    </choice>
  </type>
</namedType>

<namedType name="Insertions">
  <type>
    <enumerated>
```

```
<enumeration name="none"/>
<enumeration name="hollow"/>
<enumeration name="singular"/>
<enumeration name="uniform"/>
<enumeration name="multiform"/>
</enumerated>
</type>
</namedType>

<namedType name="SequenceType">
  <type>
    <sequence insertions="hollow">
      <optional>
        <attribute name="insertions" type="asn:Insertions"/>
      </optional>
      <optional>
        <group name="root" type="asn:ComponentTypeList"/>
      </optional>
      <optional>
        <group name="extensionAndFinal">
          <type>
            <sequence insertions="hollow">
              <element name="extension">
                <type>
                  <sequence insertions="hollow">
                    <optional>
                      <element name="exception" type="asn:ExceptionSpec"/>
                    </optional>
                    <optional>
                      <group name="additions" type="asn:ExtensionAdditions"/>
                    </optional>
                  </sequence>
                </type>
              </element>
            </optional>
          </group>
        </optional>
      </sequence>
    </type>
  </type>
</namedType>

<namedType name="ComponentTypeList">
  <type>
    <sequenceOf minSize="1">
```



```
<group name="componentType" type="asn:ComponentType"/>
</sequenceOf>
</type>
</namedType>

<namedType name="ComponentType">
  <type>
    <choice insertions="none">
      <group name="component" type="asn:SequenceNamedType"/>
      <element name="optional">
        <type>
          <sequence>
            <group name="component" type="asn:SequenceNamedType"/>
            <optional>
              <element name="default" type="asn:Value"/>
            </optional>
          </sequence>
        </type>
      </element>
      <element name="componentsOf" type="asn:Type"/>
    </choice>
  </type>
</namedType>

<namedType name="SequenceNamedType">
  <type>
    <constrained type="asn:NamedType">
      <withComponents partial="true">
        <element name="member" use="absent"/>
        <element name="item" use="absent"/>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="ExtensionAdditions">
  <type>
    <sequenceOf minSize="1">
      <group name="addition" type="asn:ExtensionAddition"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="ExtensionAddition">
  <type>
    <choice insertions="none">
      <element name="extensionGroup"
        type="asn:ExtensionAdditionGroup"/>
    </choice>
  </type>
</namedType>
```

```
<group name="componentType" type="asn:ComponentType"/>
</choice>
</type>
</namedType>

<namedType name="ExtensionAdditionGroup">
  <type>
    <sequence insertions="hollow">
      <optional>
        <attribute name="version" type="asn:VersionNumber"/>
      </optional>
      <group name="componentTypes" type="asn:ComponentTypeList"/>
    </sequence>
  </type>
</namedType>

<namedType name="VersionNumber">
  <type>
    <constrained type="asn:INTEGER">
      <range>
        <minInclusive literalValue="2"/>
      </range>
    </constrained>
  </type>
</namedType>

<namedType name="SetType" type="asn:SequenceType"/>

<namedType name="ChoiceOrUnionType">
  <type>
    <sequence insertions="hollow">
      <optional>
        <attribute name="insertions" type="asn:Insertions"/>
      </optional>
      <optional>
        <attribute name="precedence" type="asn:PrecedenceList"/>
      </optional>
      <group name="root" type="asn:AlternativeTypeList"/>
      <optional>
        <element name="extension">
          <type>
            <sequence insertions="hollow">
              <optional>
                <element name="exception" type="asn:ExceptionSpec"/>
              </optional>
              <optional>
                <group name="additions"
                  type="asn:ExtensionAdditionAlternatives"/>
              </optional>
            </sequence>
          </type>
        </element>
      </optional>
    </sequence>
  </type>
</namedType>
```

```
        </optional>
      </sequence>
    </type>
  </element>
</optional>
</sequence>
</type>
</namedType>

<namedType name="PrecedenceList">
  <type>
    <list minSize="1">
      <item name="member" type="asn:QName"/>
    </list>
  </type>
</namedType>

<namedType name="AlternativeTypeList">
  <type>
    <sequenceOf minSize="1">
      <group name="component" type="asn:ChoiceOrUnionNamedType"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="ChoiceOrUnionNamedType">
  <type>
    <constrained type="asn:NamedType">
      <withComponents partial="true">
        <element name="item" use="absent"/>
        <element name="simpleContent" use="absent"/>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="ExtensionAdditionAlternatives">
  <type>
    <sequenceOf minSize="1">
      <group name="addition" type="asn:ExtensionAdditionAlternative"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="ExtensionAdditionAlternative">
  <type>
    <choice insertions="none">
      <element name="extensionGroup">
```

```
        type="asn:ExtensionAdditionAlternativesGroup"/>
    <group name="component" type="asn:ChoiceOrUnionNamedType"/>
  </choice>
</type>
</namedType>

<namedType name="ExtensionAdditionAlternativesGroup">
  <type>
    <sequence insertions="hollow">
      <optional>
        <attribute name="version" type="asn:VersionNumber"/>
      </optional>
      <group name="alternatives" type="asn:AlternativeTypeList"/>
    </sequence>
  </type>
</namedType>

<namedType name="ChoiceType">
  <type>
    <constrained type="asn:ChoiceOrUnionType">
      <withComponents partial="true">
        <attribute name="precedence" use="absent"/>
        <group name="root">
          <withComponent>
            <includes type="asn:ChoiceNamedType"/>
          </withComponent>
        </group>
        <element name="extension">
          <withComponents partial="true">
            <group name="additions">
              <withComponent>
                <withComponents partial="true">
                  <element name="extensionGroup">
                    <withComponents partial="true">
                      <group name="alternatives">
                        <withComponent>
                          <includes type="asn:ChoiceNamedType"/>
                        </withComponent>
                      </group>
                    </withComponents>
                  </element>
                <group name="component">
                  <includes type="asn:ChoiceNamedType"/>
                </group>
              </withComponents>
            </withComponent>
          </group>
        </withComponents>
      </withComponent>
    </constrained>
  </type>
</namedType>
```

```
    </element>
  </withComponents>
</constrained>
</type>
</namedType>

<namedType name="ChoiceNamedType">
  <type>
    <constrained type="asn:ChoiceOrUnionNamedType">
      <withComponents partial="true">
        <element name="member" use="absent"/>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="UnionType">
  <type>
    <constrained type="asn:ChoiceOrUnionType">
      <withComponents partial="true">
        <attribute name="insertions" use="absent"/>
        <group name="root">
          <withComponent>
            <includes type="asn:UnionNamedType"/>
          </withComponent>
        </group>
        <element name="extension">
          <withComponents partial="true">
            <group name="additions">
              <withComponent>
                <withComponents partial="true">
                  <element name="extensionGroup">
                    <withComponents partial="true">
                      <group name="alternatives">
                        <withComponent>
                          <includes type="asn:UnionNamedType"/>
                        </withComponent>
                      </group>
                    </withComponents>
                  </element>
                <group name="component">
                  <includes type="asn:UnionNamedType"/>
                </group>
              </withComponents>
            </withComponent>
          </group>
        </withComponents>
      </withComponent>
    </group>
  </withComponents>
</element>
```

```
    </withComponents>
  </constrained>
</type>
</namedType>

<namedType name="UnionNamedType">
  <type>
    <constrained type="asn:ChoiceOrUnionNamedType">
      <withComponents partial="true">
        <element name="component" use="absent"/>
        <element name="element" use="absent"/>
        <element name="attribute" use="absent"/>
        <element name="group" use="absent"/>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="SequenceOfOrListType">
  <type>
    <sequence>
      <optional>
        <attribute name="minSize">
          <type>
            <constrained type="asn:INTEGER">
              <range>
                <minInclusive literalValue="0"/>
              </range>
            </constrained>
          </type>
        </attribute>
      </optional>
      <optional>
        <attribute name="maxSize">
          <type>
            <constrained type="asn:INTEGER">
              <range>
                <minInclusive literalValue="0"/>
              </range>
            </constrained>
          </type>
        </attribute>
      </optional>
      <group name="component">
        <type>
          <constrained type="asn:NamedType">
            <withComponents partial="true">
              <element name="attribute" use="absent"/>
            </withComponents>
          </constrained>
        </type>
      </group>
    </sequence>
  </type>
</namedType>
```

```
        <element name="member" use="absent"/>
        <element name="simpleContent" use="absent"/>
    </withComponents>
</constrained>
</type>
</group>
</sequence>
</type>
</namedType>

<namedType name="SequenceOfType">
    <type>
        <constrained type="asnx:SequenceOfOrListType">
            <withComponents partial="true">
                <group name="component">
                    <withComponents partial="true">
                        <element name="item" use="absent"/>
                    </withComponents>
                </group>
            </withComponents>
        </constrained>
    </type>
</namedType>

<namedType name="SetOfType" type="asnx:SequenceOfType"/>

<namedType name="ListType">
    <type>
        <constrained type="asnx:SequenceOfOrListType">
            <withComponents partial="true">
                <group name="component">
                    <withComponents partial="true">
                        <element name="component" use="absent"/>
                        <element name="element" use="absent"/>
                        <element name="group" use="absent"/>
                    </withComponents>
                </group>
            </withComponents>
        </constrained>
    </type>
</namedType>

<namedType name="ConstrainedType">
    <type>
        <sequence insertions="hollow">
            <group name="type" type="asnx:Type"/>
            <group name="constraint" type="asnx:Constraint"/>
        </sequence>
    </type>
</namedType>
```

```
</type>
</namedType>

<namedType name="Constraint">
  <type>
    <sequence>
      <group name="constraintSpec">
        <type>
          <choice insertions="none">
            <group name="subtype" type="asn:ElementSetSpecs"/>
            <element name="constrainedBy"
              type="asn:UserDefinedConstraint"/>
            <element name="table" type="asn:TableConstraint"/>
            <element name="contents" type="asn:ContentsConstraint"/>
          </choice>
        </type>
      </group>
      <optional>
        <element name="exception" type="asn:ExceptionSpec"/>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="UserDefinedConstraint">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <optional>
        <group name="parameters" type="asn:ConstraintParameters"/>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="ConstraintParameters">
  <type>
    <sequenceOf minSize="1">
      <group name="parameter"
        type="asn:UserDefinedConstraintParameter"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="UserDefinedConstraintParameter">
  <type>
```



```
<choice insertions="singular">
  <element name="valueParameter">
    <type>
      <sequence>
        <group name="type" type="asn:Type"/>
        <group name="value" type="asn:Value"/>
      </sequence>
    </type>
  </element>
  <element name="valueSetParameter">
    <type>
      <sequence>
        <group name="type" type="asn:Type"/>
        <group name="valueSet" type="asn:ValueSet"/>
      </sequence>
    </type>
  </element>
  <element name="objectParameter">
    <type>
      <sequence>
        <group name="objectClass" type="asn:DefinedObjectClass"/>
        <group name="object" type="asn:Object"/>
      </sequence>
    </type>
  </element>
  <element name="objectSetParameter">
    <type>
      <sequence>
        <group name="objectClass" type="asn:DefinedObjectClass"/>
        <group name="objectSet" type="asn:ObjectSet"/>
      </sequence>
    </type>
  </element>
  <element name="typeParameter">
    <type>
      <sequence>
        <group name="type" type="asn:Type"/>
      </sequence>
    </type>
  </element>
  <element name="classParameter">
    <type>
      <sequence>
        <group name="objectClass" type="asn:DefinedObjectClass"/>
      </sequence>
    </type>
  </element>
</choice>
```

```
</type>
</namedType>

<namedType name="TableConstraint">
  <type>
    <sequence>
      <group name="objectSet" type="asn:ObjectSet"/>
      <optional>
        <group name="componentRelation" type="asn:AtNotations"/>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="AtNotations">
  <type>
    <sequenceOf minSize="1">
      <element name="restrictBy" type="asn:AtNotation"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="AtNotation" type="asn:Markup"/>

<namedType name="ContentsConstraint">
  <type>
    <constrained>
      <type>
        <sequence>
          <optional>
            <element name="containing" type="asn:Type"/>
          </optional>
          <optional>
            <element name="encodedBy" type="asn:Value"/>
          </optional>
        </sequence>
      </type>
      <union>
        <withComponents partial="true">
          <element name="containing" use="present"/>
        </withComponents>
        <withComponents partial="true">
          <element name="encodedBy" use="present"/>
        </withComponents>
      </union>
    </constrained>
  </type>
</namedType>
```

```
<namedType name="ExceptionSpec">
  <type>
    <sequence>
      <group name="type" type="asn:Type"/>
      <group name="value" type="asn:Value"/>
    </sequence>
  </type>
</namedType>

<namedType name="Value">
  <type>
    <choice insertions="none">
      <attribute name="literalValue" identifier="literalValueAtt"
        type="asn:UTF8String"/>
      <element name="literalValue"
        type="asn:ElementFormLiteralValue"/>
      <attribute name="value" identifier="valueRef" type="asn:QName"/>
      <element name="value" type="asn:ElementFormNotationalValue"/>
    </choice>
  </type>
</namedType>

<namedType name="ElementFormLiteralValue" type="asn:Markup">
  <annotation>
    If asn:literal="false" then the governing type of
    ElementFormLiteralValue is ElementFormNotationalValue.
  </annotation>
</namedType>

<namedType name="ElementFormNotationalValue">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <group name="definition">
        <type>
          <choice insertions="none">
            <group name="reference" type="asn:Reference"/>
            <element name="expanded" type="asn:ExpandedValue"/>
            <element name="fromObjects"
              type="asn:InformationFromObjects"/>
            <element name="openTypeValue">
              <type>
                <sequence>
                  <group name="type" type="asn:Type"/>
                  <group name="value" type="asn:Value"/>
                </sequence>
              </type>
            </element>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>
```

```
        </type>
      </element>
      <group name="components" type="asn:ComponentValueList"/>
    </choice>
  </type>
</group>
</sequence>
</type>
</namedType>

<namedType name="Reference">
  <type>
    <sequence>
      <attribute name="ref" type="asn:QName"/>
      <optional>
        <attribute name="context" type="asn:AnyURI"/>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="ExpandedValue">
  <type>
    <sequence>
      <optional>
        <attribute name="name" type="asn:NCName"/>
      </optional>
      <optional>
        <element name="module" type="asn:ReferencedModule"/>
      </optional>
      <group name="value" type="asn:Value"/>
    </sequence>
  </type>
</namedType>

<namedType name="ComponentValueList">
  <type>
    <sequenceOf minSize="1">
      <group name="component" type="asn:NamedValue"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="NamedValue">
  <type>
    <choice insertions="singular">
      <element name="component" type="asn:GenericNamedValue"/>
      <element name="element" type="asn:GenericNamedValue"/>
    </choice>
  </type>
</namedType>
```

```
<element name="attribute" type="asn:GenericNamedValue"/>
<element name="group" type="asn:GenericNamedValue"/>
<element name="member" type="asn:GenericNamedValue"/>
<element name="item" type="asn:GenericNamedValue"/>
<element name="simpleContent" type="asn:GenericNamedValue"/>
</choice>
</type>
</namedType>

<namedType name="GenericNamedValue">
  <type>
    <sequence>
      <attribute name="name" type="asn:QName"/>
      <group name="value" type="asn:Value"/>
    </sequence>
  </type>
</namedType>

<namedType name="ValueSet">
  <type>
    <choice insertions="none">
      <attribute name="valueSet" identifier="valueSetRef"
        type="asn:QName">
        <annotation>
          valueSet attribute is not used in this version
        </annotation>
      </attribute>
      <element name="valueSet" type="asn:ElementFormValueSet"/>
    </choice>
  </type>
</namedType>

<namedType name="ElementFormValueSet">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <group name="definition">
        <type>
          <choice insertions="none">
            <group name="elementSetSpecs" type="asn:ElementSetSpecs"/>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>
```

```
<namedType name="ElementSetSpec">
  <type>
    <sequence insertions="hollow">
      <group name="root" type="asn:ValueElementSetSpec"/>
      <optional>
        <element name="extension">
          <type>
            <sequence insertions="hollow">
              <optional>
                <group name="additions" type="asn:ValueElementSetSpec"/>
              </optional>
            </sequence>
          </type>
        </element>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="ValueElementSetSpec">
  <type>
    <constrained type="asn:ElementSetSpec">
      <withComponents partial="true">
        <element name="object" use="absent"/>
        <element name="objectSet" use="absent"/>
        <element name="union">
          <withComponent>
            <includes type="asn:ValueElementSetSpec"/>
          </withComponent>
        </element>
        <element name="intersection">
          <withComponent>
            <includes type="asn:ValueElementSetSpec"/>
          </withComponent>
        </element>
        <element name="all">
          <withComponents partial="true">
            <group name="elements">
              <includes type="asn:ValueElementSetSpec"/>
            </group>
            <element name="except">
              <includes type="asn:ValueElementSetSpec"/>
            </element>
          </withComponents>
        </element>
      </withComponents>
    </constrained>
  </type>
```

```
</namedType>

<namedType name="ElementSetSpec">
  <type>
    <choice insertions="singular">
      <element name="literalValue"
        type="asn:ElementFormLiteralValue"/>
      <element name="value" type="asn:ElementFormNotationalValue"/>
      <element name="includes" type="asn:Type"/>
      <element name="range" type="asn:ValueRange"/>
      <element name="size" type="asn:Constraint"/>
      <element name="typeConstraint" type="asn:Type"/>
      <element name="from" type="asn:Constraint"/>
      <element name="withComponent" type="asn:Constraint"/>
      <element name="withComponents"
        type="asn:MultipleTypeConstraints"/>
      <element name="pattern" type="asn:Value"/>
      <element name="object" type="asn:ElementFormObject"/>
      <element name="objectSet" type="asn:ElementFormObjectSet"/>
      <element name="union" type="asn:ElementSetSpecList"/>
      <element name="intersection" type="asn:ElementSetSpecList"/>
      <element name="all">
        <type>
          <sequence>
            <optional>
              <group name="elements" type="asn:ElementSetSpec"/>
            </optional>
            <element name="except" type="asn:ElementSetSpec"/>
          </sequence>
        </type>
      </element>
    </choice>
  </type>
</namedType>

<namedType name="ElementSetSpecList">
  <type>
    <sequenceOf minSize="2">
      <group name="elements" type="asn:ElementSetSpec"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="ValueRange">
  <type>
    <sequence>
      <optional>
        <group name="minimum">
```

```
<type>
  <choice insertions="none">
    <element name="minInclusive" type="asn:EndValue"/>
    <element name="minExclusive" type="asn:EndValue"/>
  </choice>
</type>
</group>
<default>
  <literalValue>
    <minInclusive/>
  </literalValue>
</default>
</optional>
<optional>
  <group name="maximum">
    <type>
      <choice insertions="none">
        <element name="maxInclusive" type="asn:EndValue"/>
        <element name="maxExclusive" type="asn:EndValue"/>
      </choice>
    </type>
  </group>
  <default>
    <literalValue>
      <maxInclusive/>
    </literalValue>
  </default>
</optional>
</sequence>
</type>
</namedType>

<namedType name="EndValue">
  <type>
    <sequence insertions="hollow">
      <optional>
        <group name="value" type="asn:Value"/>
      </optional>
    </sequence>
  </type>
</namedType>

<namedType name="MultipleTypeConstraints">
  <type>
    <sequence insertions="hollow">
      <optional>
        <attribute name="partial" type="asn:BOOLEAN"/>
        <default literalValue="false"/>
      </optional>
    </sequence>
  </type>
</namedType>
```



```
        </optional>
        <group name="typeConstraints" type="asn1:TypeConstraints"/>
    </sequence>
</type>
</namedType>

<namedType name="TypeConstraints">
    <type>
        <sequenceOf minSize="1">
            <group name="namedConstraint" type="asn1:NamedConstraint"/>
        </sequenceOf>
    </type>
</namedType>

<namedType name="NamedConstraint">
    <type>
        <choice insertions="singular">
            <element name="component" type="asn1:GenericNamedConstraint"/>
            <element name="element" type="asn1:GenericNamedConstraint"/>
            <element name="attribute" type="asn1:GenericNamedConstraint"/>
            <element name="group" type="asn1:GenericNamedConstraint"/>
            <element name="member" type="asn1:GenericNamedConstraint"/>
            <element name="item" type="asn1:GenericNamedConstraint"/>
            <element name="simpleContent"
                type="asn1:GenericNamedConstraint"/>
        </choice>
    </type>
</namedType>

<namedType name="GenericNamedConstraint">
    <type>
        <sequence insertions="hollow">
            <attribute name="name" type="asn1:QName"/>
            <optional>
                <attribute name="use" type="asn1:PresenceConstraint"/>
            </optional>
            <optional>
                <group name="constraint" type="asn1:Constraint"/>
            </optional>
        </sequence>
    </type>
</namedType>

<namedType name="PresenceConstraint">
    <type>
        <enumerated>
            <enumeration name="present"/>
            <enumeration name="absent"/>
        </enumerated>
    </type>
</namedType>
```

```
<enumeration name="optional"/>
</enumerated>
</type>
</namedType>

<namedType name="ObjectClass">
  <type>
    <choice insertions="singular">
      <attribute name="class" identifier="classRef" type="asn:QName"/>
      <element name="class" type="asn:ElementFormObjectClass"/>
    </choice>
  </type>
</namedType>

<namedType name="DefinedObjectClass">
  <type>
    <constrained type="asn:ObjectClass">
      <withComponents partial="true">
        <element name="class">
          <withComponents partial="true">
            <group name="definition">
              <withComponents partial="true">
                <group name="objectClassDefn" use="absent"/>
              </withComponents>
            </group>
          </withComponents>
        </element>
      </withComponents>
    </constrained>
  </type>
</namedType>

<namedType name="ElementFormObjectClass">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <group name="definition">
        <type>
          <choice insertions="none">
            <group name="reference" type="asn:Reference"/>
            <element name="expanded" type="asn:ExpandedObjectClass"/>
            <group name="objectClassDefn" type="asn:ObjectClassDefn"/>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>
```

```
</type>
</namedType>

<namedType name="ExpandedObjectClass">
  <type>
    <sequence>
      <optional>
        <attribute name="name" type="asn:NCName"/>
      </optional>
      <optional>
        <element name="module" type="asn:ReferencedModule"/>
      </optional>
      <group name="objectClass" type="asn:ObjectClass"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectClassDefn">
  <type>
    <sequenceOf minSize="1">
      <group name="fieldSpec" type="asn:FieldSpec"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="FieldSpec">
  <type>
    <choice insertions="singular">
      <element name="typeField" type="asn:TypeField"/>
      <element name="valueField" type="asn:ValueField"/>
      <element name="valueSetField" type="asn:ValueSetField"/>
      <element name="objectField" type="asn:ObjectField"/>
      <element name="objectSetField" type="asn:ObjectSetField"/>
      <element name="optional" type="asn:OptionalField"/>
    </choice>
  </type>
</namedType>

<namedType name="OptionalField">
  <type>
    <constrained>
      <type>
        <sequence>
          <group name="field">
            <type>
              <choice insertions="singular">
                <element name="typeField" type="asn:TypeField"/>
                <element name="valueField" type="asn:ValueField"/>
              </choice>
            </type>
          </group>
        </sequence>
      </type>
    </constrained>
  </type>
</namedType>
```

```
    <element name="valueSetField" type="asn1:ValueSetField"/>
    <element name="objectField" type="asn1:ObjectField"/>
    <element name="objectSetField" type="asn1:ObjectSetField"/>
  </choice>
</type>
</group>
<optional>
  <element name="default" type="asn1:Setting"/>
</optional>
</sequence>
</type>
<union>
  <withComponents partial="true">
    <group name="field">
      <withComponents>
        <element name="typeField" use="present"/>
      </withComponents>
    </group>
    <element name="default">
      <withComponents partial="true">
        <group name="value" use="absent"/>
        <group name="valueSet" use="absent"/>
        <group name="object" use="absent"/>
        <group name="objectSet" use="absent"/>
      </withComponents>
    </element>
  </withComponents>
  <withComponents partial="true">
    <group name="field">
      <withComponents>
        <element name="valueField" use="present"/>
      </withComponents>
    </group>
    <element name="default">
      <withComponents partial="true">
        <group name="type" use="absent"/>
        <group name="valueSet" use="absent"/>
        <group name="object" use="absent"/>
        <group name="objectSet" use="absent"/>
      </withComponents>
    </element>
  </withComponents>
  <withComponents partial="true">
    <group name="field">
      <withComponents>
        <element name="valueSetField" use="present"/>
      </withComponents>
    </group>
  </withComponents>
</union>
```

```
<element name="default">
  <withComponents partial="true">
    <group name="type" use="absent"/>
    <group name="value" use="absent"/>
    <group name="object" use="absent"/>
    <group name="objectSet" use="absent"/>
  </withComponents>
</element>
</withComponents>
<withComponents partial="true">
  <group name="field">
    <withComponents>
      <element name="objectField" use="present"/>
    </withComponents>
  </group>
  <element name="default">
    <withComponents partial="true">
      <group name="type" use="absent"/>
      <group name="value" use="absent"/>
      <group name="valueSet" use="absent"/>
      <group name="objectSet" use="absent"/>
    </withComponents>
  </element>
</withComponents>
<withComponents partial="true">
  <group name="field">
    <withComponents>
      <element name="objectSetField" use="present"/>
    </withComponents>
  </group>
  <element name="default">
    <withComponents partial="true">
      <group name="type" use="absent"/>
      <group name="value" use="absent"/>
      <group name="valueSet" use="absent"/>
      <group name="object" use="absent"/>
    </withComponents>
  </element>
</withComponents>
</union>
</constrained>
</type>
</namedType>

<namedType name="TypeField">
  <type>
    <sequence>
      <optional>
```

```
    <element name="annotation" type="asn:Annotation"/>
  </optional>
  <attribute name="name" type="asn:TypeFieldReference"/>
</sequence>
</type>
</namedType>

<namedType name="TypeFieldReference" type="asn:TypeReference"/>

<namedType name="ValueField">
  <type>
    <constrained>
      <type>
        <sequence>
          <optional>
            <element name="annotation" type="asn:Annotation"/>
          </optional>
          <attribute name="name" type="asn:ValueFieldReference"/>
          <optional>
            <attribute name="unique" type="asn:BOOLEAN"/>
          </optional>
          <group name="governor">
            <type>
              <choice insertions="singular">
                <group name="type" type="asn:Type"/>
                <element name="typeFromField" type="asn:FieldName"/>
              </choice>
            </type>
          </group>
        </sequence>
      </type>
    <union>
      <withComponents partial="true">
        <attribute name="unique" use="absent"/>
      </withComponents>
      <withComponents partial="true">
        <group name="governor">
          <withComponents partial="true">
            <element name="typeFromField" use="absent"/>
          </withComponents>
        </group>
      </withComponents>
    </union>
  </constrained>
</type>
</namedType>

<namedType name="ValueFieldReference" type="asn:ValueReference"/>
```

```
<namedType name="ValueSetField">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <attribute name="name" type="asn:ValueSetFieldReference"/>
      <group name="governor">
        <type>
          <choice insertions="singular">
            <group name="type" type="asn:Type"/>
            <element name="typeFromField" type="asn:FieldName"/>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>

<namedType name="ValueSetFieldReference" type="asn:TypeReference"/>

<namedType name="ObjectField">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <attribute name="name" type="asn:ObjectFieldReference"/>
      <group name="objectClass" type="asn:DefinedObjectClass"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectFieldReference" type="asn:ObjectReference"/>

<namedType name="ObjectSetField">
  <type>
    <sequence>
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <attribute name="name" type="asn:ObjectSetFieldReference"/>
      <group name="objectClass" type="asn:DefinedObjectClass"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectSetFieldReference"
```

```
        type="asn:ObjectSetReference"/>

<namedType name="Object">
  <type>
    <choice insertions="none">
      <attribute name="object" identifier="objectRef"
        type="asn:QName"/>
      <element name="object" type="asn:ElementFormObject"/>
    </choice>
  </type>
</namedType>

<namedType name="ElementFormObject">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <group name="definition">
        <type>
          <choice insertions="singular">
            <group name="reference" type="asn:Reference"/>
            <element name="expanded" type="asn:ExpandedObject"/>
            <element name="fromObjects"
              type="asn:InformationFromObjects"/>
            <group name="fields" type="asn:ObjectDefn"/>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>

<namedType name="ExpandedObject">
  <type>
    <sequence>
      <optional>
        <attribute name="name" type="asn:NCName"/>
      </optional>
      <optional>
        <element name="module" type="asn:ReferencedModule"/>
      </optional>
      <group name="object" type="asn:Object"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectDefn">
```



```
<type>
  <sequenceOf>
    <element name="field" type="asn:FieldSetting"/>
  </sequenceOf>
</type>
</namedType>

<namedType name="FieldSetting">
  <type>
    <sequence insertions="hollow">
      <attribute name="name" type="asn:NCName"/>
      <group name="setting" type="asn:Setting"/>
    </sequence>
  </type>
</namedType>

<namedType name="Setting">
  <type>
    <choice>
      <group name="type" type="asn:Type"/>
      <group name="value" type="asn:Value"/>
      <group name="valueSet" type="asn:ValueSet"/>
      <group name="object" type="asn:Object"/>
      <group name="objectSet" type="asn:ObjectSet"/>
    </choice>
  </type>
</namedType>

<namedType name="ObjectSet">
  <type>
    <choice insertions="none">
      <attribute name="objectSet" identifier="objectSetRef"
        type="asn:QName"/>
      <element name="objectSet" type="asn:ElementFormObjectSet"/>
    </choice>
  </type>
</namedType>

<namedType name="ElementFormObjectSet">
  <type>
    <sequence insertions="hollow">
      <optional>
        <element name="annotation" type="asn:Annotation"/>
      </optional>
      <group name="definition">
        <type>
          <choice insertions="none">
            <group name="reference" type="asn:Reference"/>
          </choice>
        </type>
      </group>
    </sequence>
  </type>
</namedType>
```

```

    <element name="expanded" type="asn:ExpandedObjectSet"/>
    <group name="objectSetSpec" type="asn:ObjectSetSpec"/>
    <element name="fromObjects"
        type="asn:InformationFromObjects"/>
    </choice>
  </type>
</group>
</sequence>
</type>
</namedType>

<namedType name="ExpandedObjectSet">
  <type>
    <sequence>
      <optional>
        <attribute name="name" type="asn:NCName"/>
      </optional>
      <optional>
        <element name="module" type="asn:ReferencedModule"/>
      </optional>
      <group name="objectSet" type="asn:ObjectSet"/>
    </sequence>
  </type>
</namedType>

<namedType name="ObjectSetSpec">
  <type>
    <constrained>
      <type>
        <sequence insertions="hollow">
          <optional>
            <group name="root" type="asn:ObjectElementSetSpec"/>
          </optional>
          <optional>
            <element name="extension">
              <type>
                <sequence insertions="hollow">
                  <optional>
                    <group name="additions" type="asn:ObjectElementSetSpec"/>
                  </optional>
                </sequence>
              </type>
            </element>
          </optional>
        </sequence>
      </type>
    </constrained>
  </type>
  <union>
    <withComponents partial="true">
```

```

    <group name="root" use="present"/>
  </withComponents>
  <withComponents partial="true">
    <element name="extension" use="present"/>
  </withComponents>
</union>
</constrained>
</type>
</namedType>

<namedType name="ObjectElementSetSpec">
  <type>
    <constrained type="asn:ElementSetSpec">
      <withComponents partial="true">
        <element name="literalValue" use="absent"/>
        <element name="value" use="absent"/>
        <element name="includes" use="absent"/>
        <element name="range" use="absent"/>
        <element name="size" use="absent"/>
        <element name="typeConstraint" use="absent"/>
        <element name="from" use="absent"/>
        <element name="withComponent" use="absent"/>
        <element name="withComponents" use="absent"/>
        <element name="pattern" use="absent"/>
        <element name="union">
          <withComponent>
            <includes type="asn:ObjectElementSetSpec"/>
          </withComponent>
        </element>
        <element name="intersection">
          <withComponent>
            <includes type="asn:ObjectElementSetSpec"/>
          </withComponent>
        </element>
        <element name="all">
          <withComponents partial="true">
            <group name="elements">
              <includes type="asn:ObjectElementSetSpec"/>
            </group>
            <element name="except">
              <includes type="asn:ObjectElementSetSpec"/>
            </element>
          </withComponents>
        </element>
      </withComponents>
    </constrained>
  </type>
</namedType>

```

```
<namedType name="EncodingControlSections">
  <type>
    <sequenceOf minSize="1">
      <group name="section" type="asn:EncodingControlSection"/>
    </sequenceOf>
  </type>
</namedType>

<namedType name="EncodingControlSection">
  <type>
    <choice insertions="singular">
      <element name="GSER" identifier="gser"
        type="asn:GSER-EncodingInstructionAssignmentList"/>
      <element name="XER" identifier="xer"
        type="asn:XER-EncodingInstructionAssignmentList"/>
      <!-- plus encoding control sections
        for other encoding rules in the future -->
    </choice>
  </type>
</namedType>

<element name="module" type="asn:ModuleDefinition"/>

<attribute name="literal" type="asn:BOOLEAN"/>

</asn:module>
```

Author's Address

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

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

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

