INTERNATIONAL TELECOMMUNICATION UNION

ITU-T X.694

(01/2004)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI networking and system aspects – Abstract Syntax Notation One (ASN.1)

Information technology – ASN.1 encoding rules – mapping W3C XML schema definitions into ASN.1

CAUTION !

PREPUBLISHED RECOMMENDATION

This prepublication is an unedited version of a recently approved Recommendation. It will be replaced by the published version after editing. Therefore, there will be differences between this prepublication and the published version.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU [had/had not] received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

INTERNATIONAL STANDARD 8825-5

ITU-T RECOMMENDATION X.694

INFORMATION TECHNOLOGY – ASN.1 ENCODING RULES – MAPPING W3C XML SCHEMA DEFINITIONS INTO ASN.1

Summary

This Recommendation | International Standard defines rules for mapping an XSD Schema (a schema conforming to the W3C XML Schema specification) to an ASN.1 schema in order to use ASN.1 encoding rules such as the Basic Encoding Rules (BER), the Distinguished Encoding Rules (DER), the Packed Encoding Rules (PER) or the XML Encoding Rules (XER) for the transfer of information defined by the XSD Schema.

The use of this Recommendation | International Standard with the ASN.1 Extended XML Encoding Rules (EXTENDED-XER) provides the same XML representation of values as that defined by the original XSD Schema.

CONTENTS

Introduction

This Recommendation | International Standard specifies a mapping from a W3C XML Schema definition (an XSD Schema) into an ASN.1 schema. The mapping can be applied to any XSD Schema. It specifies the generation of one or more ASN.1 modules containing type definitions, together with ASN.1 XER encoding instructions. These are jointly described as an ASN.1 schema for XML documents.

This ASN.1 schema, when used with the ASN.1 Extended XML Encoding Rules (EXTENDED-XER), can be used to generate and to validate the same set of W3C XML 1.0 documents as the original XSD Schema. The resulting ASN.1 types and encodings support the same semantic content as the XSD Schema. Thus ASN.1 tools can be used interchangeably with XSD tools for the generation and processing of the specified XML documents.

Other standardized ASN.1 encoding rules, such as the Distinguished Encoding Rules (DER) or the Packed Encoding Rules (PER), can be used in conjunction with this standardized mapping.

The combination of this Recommendation | International Standard with ASN.1 Encoding Rules provides fully-standardized and vendor-independent compact and canonical binary encodings for data defined using an XSD Schema.

The ASN.1 schema provides a clear separation between the specification of the information content of messages (their abstract syntax) and the precise form of the XML document (for example, use of attributes instead of elements). This results in both a clearer and generally a less verbose schema than the original XSD Schema.

Annex A forms an integral part of this Recommendation | International Standard, and is an ASN.1 module containing a set of ASN.1 type assignments that correspond to each of the XSD built-in datatypes. Mappings of XSD Schemas into ASN.1 schemas either import the type reference names of those type assignments or include the type definitions in-line.

Annex B does not form an integral part of this Recommendation | International Standard, and summarizes the object identifier values assigned in this Recommendation | International Standard.

Annex C does not form an integral part of this Recommendation | International Standard, and gives examples of the mapping of XSD Schemas into ASN.1 schemas.

Annex D does not form an integral part of this Recommendation | International Standard, and describes the use of the mapping defined in this Recommendation | International Standard, in conjunction with standardized ASN.1 Encoding Rules, to provided compact and canonical encodings for data defined using an XSD Schema.

INTERNATIONAL STANDARD 8825-5

ITU-T RECOMMENDATION X.694

INFORMATION TECHNOLOGY – ASN.1 ENCODING RULES – MAPPING W3C XML SCHEMA DEFINITIONS INTO ASN.1

1 Scope

This Recommendation | International Standard specifies a mapping from any XSD Schema into an ASN.1 schema. The ASN.1 schema supports the same semantics and validates the same set of XML documents.

This Recommendation | International Standard specifies the final XER encoding instructions that are to be applied as part of the defined mapping to ASN.1 types, but does not specify which syntactic form is to be used for the specification of those final XER encoding instructions, or the order or manner of their assignment.

NOTE – Implementers of tools generating these mappings may choose any syntactic form or order of assignment that results in the specified final XER encoding instructions being applied. Examples in this Recommendation | International Standard generally use the type prefix form, but use of an XER Encoding Control Section may be preferred for the mapping of a complete XSD Schema, as a matter of style.

There are different ways (syntactically) of assigning XER encoding instructions for use in EXTENDED-XER encodings (for example, use of ASN.1 type prefix encoding instructions or use of an XER encoding control section). The choice of these syntactic forms is a matter of style and is outside the scope of this Recommendation | International Standard.

2 Normative references

The following Recommendations | International Standards and W3C specifications contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations, International Standards and W3C specifications are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations, International Standards and W3C specifications listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations. The W3C maintains a list of currently valid W3C specifications.

2.1 Identical Recommendations | International Standards

NOTE – The complete set of ASN.1 Recommendations | International Standards are listed below, as they can all be applicable in particular uses of this Recommendation | International Standard. Where these are not directly referenced in the body of this Recommendation | International Standard, a † symbol is added to the reference.

- ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*. †
- ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*.
- ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*. †

– ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*. †

– ITU-T Recommendation X.691 (2002) | ISO/IEC 8825-2:2002, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*. †

– ITU-T Recommendation X.692 (2002) | ISO/IEC 8825-3:2002, *Information technology – ASN.1 encoding rules: Encoding Control Notation (ECN) for ASN.1*. †

– ITU-T Recommendation X.693 (2002) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: Specification of XML Encoding Rules (XER)*.

– ITU-T Recommendation X.693 (2002) / Amd.1 (2003) | ISO/IEC 8825-4:2002/Amd.1: 2003, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) – Amendment 1: XER Encoding Instructions and EXTENDED-XER.*

2.2 Additional references

– ISO 8601:2000, *Data elements and interchange formats – Information interchange – Representation of dates and times.*

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

– W3C XML 1.0:2000, *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, Copyright © [6 October 2000] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2000/REC-xml-20001006.*

- W3C XML Namespaces:1999, *Namespaces in XML, W3C Recommendation, Copyright © [14 January 1999] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/1999/REC-xml-names-19990114.*

– W3C XML Information Set:2001, XML *Information Set, W3C Recommendation, Copyright © [24 October 2001] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2001/REC-xml-infoset-20011024/.*

– W3C XML Schema:2001, *XML Schema Part 1: Structures, W3C Recommendation, Copyright © [2 May 2001] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/*

– W3C XML Schema:2001, *XML Schema Part 2: Datatypes, W3C Recommendation, Copyright © [2 May 2001] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/*

NOTE – When the reference "W3C XML Schema" is used in this Recommendation | International Standard, it refers to W3C XML Schema Part 1 and W3C XML Schema Part 2.

– IETF RFC 2396 (1998), *Uniform Resource Identifiers (URI): Generic Syntax*.

– IETF RFC 1766 (1995) *Tags for the Identification of Languages*.

NOTE – The reference to a document within this Recommendation | International Standard does not give it, as a stand-alone document, the status of a Recommendation or International Standard.

3 Definitions

3.1 Imported definitions

3.1.1 This Recommendation | International Standard uses the terms defined in ITU-T Rec. X.680 | ISO/IEC 8824-1 and in ITU-T Rec. X.693 | ISO/IEC 8825-4.

NOTE – In particular, the terms "final XER encoding instructions", "type prefix" and "XER encoding control section" are defined in these Recommendations | International Standards.

3.1.2 This Recommendation | International Standard also uses the terms defined in W3C XML Schema and W3C XML Information Set.

NOTE 1 – It is believed that these terms do not conflict with the terms referenced in 3.1.1. If such a conflict occurs, the definition of the term in 3.1.1 applies.

NOTE 2 – In particular, the terms "schema component" and "property (of a schema component)" are defined in W3C XML Schema, and the term "element information item" is defined in W3C XML Information Set.

3.2 Additional definitions

For the purposes of this Recommendation | International Standard, the following additional definitions apply.

3.2.1 XSD namespace: A namespace with a URI of "http://www.w3.org/2001/XMLSchema"

3.2.2 XSI namespace: A namespace with a URI of "http://www.w3.org/2001/XMLSchema-instance"

3.2.3 XML namespace: A namespace with a URI of "http://www.w3.org/XML/1998/namespace"

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
BER	ASN.1 Basic Encoding Rules
DER	ASN.1 Distinguished Encoding Rules
PER	ASN.1 Packed Encoding Rules
URI	IETF Uniform Resource Identifier
XER	ASN.1 XML Encoding Rules
XML	W3C Extensible Markup Language
XSD	W3C XML Schema

5 Notation

5.1 This Recommendation | International Standard references the notation defined by ITU-T Rec. X.680 | ISO/IEC 8824-1, ITU-T Rec. X.682 | ISO/IEC 8824-3, W3C XML 1.0 and W3C XML Schema.

5.2 When it is necessary in the body of this Recommendation | International Standard to specify, either formally or in examples, the assignment of XER encoding instructions, the type prefix notation is generally used (but see 6.3 and 6.4). In Annex A, an XER encoding control section is used.

5.3 In this Recommendation | International Standard, **`bold Courier`** is used for ASN.1 notation and **bold Arial** is used for XSD notation and for XSD terms and concepts.

5.4 The XSD Schemas used in the examples in this Recommendation | International Standard use the prefix **xsd:** to identify the XSD namespace.

6 Purpose and extent of standardization

6.1 The mapping to ASN.1 that is specified in this Recommendation | International Standard ensures that:

a) any resulting ASN.1 modules generated by tools conforming to this Recommendation | International Standard (from the same XSD Schema) define the same (structured) abstract values;

b) all BASIC-XER, CXER, EXTENDED-XER, and binary encodings of that resulting ASN.1 specification will produce the same encodings (subject to encoder's options); and

c) all XML documents that conform to the source XSD Schema are valid EXTENDED-XER encodings of abstract values of that ASN.1 specification.

6.2 There are many aspects of an ASN.1 definition (such as the use of white-space, or of encoding control sections or type prefixes) that affect neither the abstract values being defined nor the XER or binary encodings of those values. Such aspects of the ASN.1 definition are generally not standardized in this Recommendation | International Standard.

6.3 There are many different ways in ASN.1 of assigning an XER encoding instruction to a type, including:

a) use of a type prefix for every encoding instruction to be assigned; or

b) use of an encoding control section, with a separate encoding instruction for each required assignment; or

c) use of an encoding control section, with a single encoding instruction making a global assignment, possibly supplemented by use of a negating encoding instruction for specific types.

6.4 This Recommendation | International Standard specifies when a final XER encoding instruction shall be present, and uses the syntax of 6.3 a) in most of its examples. However, the use of the different options in 6.3 is not standardized, and conforming implementations of the mapping may choose any syntactic form (or a mixture of syntactic forms) for the assignment of final XER encoding instructions.

NOTE – The choice among these options does not affect the final binary or XML encodings.

6.5 A formal specification of the required mapping is not provided.

6.6 This Recommendation | International Standard is concerned only with the mapping of XSD Schemas that conform to W3C XML Schema.

NOTE – Such conformance can be either by the provision of one or more W3C XSD schema documents or by other means as specified in W3C XML Schema.

7 Mapping XSD Schemas

7.1 A mapping is based on a source XSD Schema, which is a set of schema components (see W3C XML Schema Part 1, 2.2). No particular representation of schema components or sets of schema components is required or assumed for the mapping, although it is expected that the source XSD Schema will usually be provided as one or more XML schema documents (see W3C XML Schema Part 1, 3.15.2).

NOTE 1 – Since the mapping is defined in terms of schema components (and not in terms of their XML representation), it is not affected by details of the XML representation, such as the use of multiple schema documents linked by **xsd:include** and **xsd:redefine** element information items, the placement of element information items in one or another schema documents, the order of **xsd:attribute** element information items within a **xsd:complexType** element information item, and so on.

NOTE 2 – Two sets of schema documents that differ in many aspects but represent the same set of schema components generate the same set of ASN.1 type assignments, with the same final encoding instructions assigned to them and to their components to any depth.

7.2 The source XSD Schema shall meet all the constraints imposed by the XSD specification. If the source XSD Schema is represented (in part or all) as a set of XML schema documents, each schema document shall be valid according to the XSD Schema for Schemas (see W3C XML Schema Part 1, Appendix A).

7.3 At least one ASN.1 module (see 7.4) shall be generated for each different **target namespace** (whether a namespace name or the keyword **absent**) that is the **target namespace** of one or more schema components in the source XSD Schema. Each ASN.1 module shall contain one or more type assignments corresponding to top-level schema components (see 7.9) that have the same **target namespace**. Each ASN.1 module can also contain one or more special ASN.1 type assignments whose associated ASN.1 type assignments are in the same ASN.1 module (see 7.6).

NOTE –The schema components represented in the multiple schema documents become part of the same XSD Schema through the use of the **xsd:include, xsd:redefine,** and **xsd:import** element information items.

7.4 The number of ASN.1 modules generated for each **target namespace** (including the keyword **absent**) may be more than one, but each ASN.1 module shall not contain type assignments corresponding to top-level schema components with different **target namespace**s (including the keyword **absent**).

7.5 When multiple ASN.1 modules are generated for a given **target namespace** (including the keyword **absent**), all the type assignments present in them shall be generated as if they were being added to a single ASN.1 module for the purpose of generating distinct type reference names (see 10.3). The type reference names generated from the **name**s of the top-level schema components with a given **target namespace** shall be the same type reference names regardless of the number of ASN.1 modules generated for that **target namespace** and regardless of the way type assignments are divided among the various ASN.1 modules.

NOTE – This is designed to provide flexibility without compromising interoperability.

7.6 Each special ASN.1 type assignment (see clauses 29, 30, and 31) shall be inserted in the same ASN.1 module as its associated ASN.1 type assignment (see 29.4, 31.4, and 30.4, respectively).

7.7 All ASN.1 modules generated by the mapping shall contain (in the XER encoding control section) a `GLOBAL-DEFAULTS MODIFIED-ENCODINGS` encoding instruction and a `GLOBAL-DEFAULTS CONTROL-NAMESPACE` encoding instruction specifying the XSI namespace.

7.8 A source XSD Schema shall be processed as follows:

a) for each top-level **element declaration**, an ASN.1 type assignment shall be generated by applying clause 14 to the **element declaration**;

b) for each top-level **attribute declaration**, an ASN.1 type assignment shall be generated by applying clause 15 to the **attribute declaration**;

c) for each user-defined top-level **simple type definition**, an ASN.1 type assignment shall be generated by applying clause 13 to the **simple type definition**;

d) for each top-level **complex type definition**, an ASN.1 type assignment shall be generated by applying clause 20 to the **complex type definition**;

e) for each **model group definition** whose **model group** has a **compositor** of **sequence** or **choice**, an ASN.1 type assignment shall be generated by applying clause 17 to the **model group definition**.

NOTE 1 - The remaining schema components of the source XSD schema will be processed as a result of mapping these schema components.

NOTE 2 - The order in which schema components are to be mapped is specified in 10.4. The order of the items of the list above has no significance for the mapping.

7.9 Column 1 of Table 1 lists schema components. Column 2 gives the reference to the clause in W3C XML Schema that defines the schema component. Column 3 lists the clause that defines the mapping of those schema components into ASN.1.

Table 1: Mapping of XSD schema components

XSD schema component	W3C XML Schema reference	Mapping defined by
attribute declaration	Part 1, 3.2	Clause 15
element declaration	Part 1, 3.3	Clause 14
complex type definition	Part 1, 3.4	Clause 20
attribute use	Part 1, 3.5	Clause 15
attribute group definition	Part 1, 3.6	*not mapped as such*
model group definition	Part 1, 3.7	Clause 17
model group	Part 1, 3.8	Clause 18
particle	Part 1, 3.9	Clause 19
wildcard	Part 1, 3.10	Clause 21

identity-constraint definition	Part 1, 3.11	*ignored by the mapping*
notation declaration	Part 1, 3.12	*ignored by the mapping*
annotation	Part 1, 3.13	*ignored by the mapping*
simple type definition	Part 1, 3.14	Clauses 11, 13
schema	Part 1, 3.15	Clause 9
ordered	Part 2, 4.2.2.1	*ignored by the mapping*
bounded	Part 2, 4.2.3.1	*ignored by the mapping*
cardinality	Part 2, 4.2.4.1	*ignored by the mapping*
numeric	Part 2, 4.2.5.1	*ignored by the mapping*
length	Part 2, 4.3.1.1	Clause 12
minLength	Part 2, 4.3.2.1	Clause 12
maxLength	Part 2, 4.3.3.1	Clause 12
pattern	Part 2, 4.3.4.1	Clause 12
enumeration	Part 2, 4.3.5.1	Clause 12
whiteSpace	Part 2, 4.3.6.1	Clause 12
maxInclusive	Part 2, 4.3.7.1	Clause 12
maxExclusive	Part 2, 4.3.8.1	Clause 12
minExclusive	Part 2, 4.3.9.1	Clause 12
minInclusive	Part 2, 4.3.10.1	Clause 12
totalDigits	Part 2, 4.3.11.1	Clause 12
fractionDigits	Part 2, 4.3.12.1	Clause 12

8 Ignored schema components and properties

8.1 The mapping shall ignore the schema components and properties that are listed in this clause.

8.2 All **annotation**s (see W3C XML Schema Part 1, 3.13) shall be ignored.

NOTE – All attribute information items in a schema document with names qualified with namespaces other than the XSD namespace (see W3C XML Schema Part 1, 3.13.1) are a property of **annotation**s, and are ignored.

8.3 All **identity-constraint definition**s (see W3C XML Schema Part 1, 3.11) shall be ignored .

NOTE – The **identity-constraint definition** provides mechanisms for specifying referential constraints that can be required in a valid instance. ASN.1 currently has no concept of such constraints, and such constraints cannot be mapped into a formal ASN.1 specification, but they may be included as normative comments that are binding on an application implementation.

8.4 All **notation declaration**s (see W3C XML Schema Part 1, 3.12) shall be ignored.

8.5 All schema components that are the **fundamental facet**s (**ordered**, **bounded**, **cardinality**, **numeric**) of **simple type definition**s (see W3C XML Schema Part 2, 4.2) shall be ignored.

8.6 The properties **identity-constraint definitions**, **substitution group exclusions** and **disallowed substitutions** of **element declaration**s shall be ignored.

8.7 The properties **final**, **abstract**, and **prohibited substitutions** of **complex type definitions** shall be ignored.

8.8 The property **process contents** of **wildcard**s shall be ignored.

NOTE – There is no support in ASN.1 for any action other than **skip**.

8.9 The properties **fundamental facets** and **final** of **simple type definition**s shall be ignored.

8.10 All **value constraint**s that are present on any **element declaration**s or **attribute declaration**s whose **type definition** is either **xsd:QName** or a **simple type definition** derived from **xsd:QName** or **xsd:NOTATION** shall be ignored.

8.11 All **attribute group definition**s shall be ignored.

NOTE – The **attribute use**s in an **attribute group definition** become part of the **attribute uses** of the **complex type definition**s whose XML representation contains a reference to the **attribute group definition**.

9 The ASN.1 module and namespaces

NOTE – A full description of the relationship between the namespace concept of W3C XML Namespaces and naming in ASN.1 is provided in ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 16. Type reference names and identifiers defined in an ASN.1 module are assigned a namespace by means of a **NAMESPACE** encoding instruction, and otherwise do not have a namespace. The mapping generates **NAMESPACE** encoding instructions as appropriate.

9.1 The mapping generates one or more ASN.1 modules corresponding to all schema components in the Schema that have the same **target namespace**.

9.2 The ASN.1 "ModuleIdentifier" (see ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 12) to be generated by the mapping is not standardized. Where **IMPORTS** statements are used, the ASN.1 module names and module identifiers in the **IMPORTS** statements shall be those generated for the ASN.1 modules generated by the mapping.

NOTE – The choice of "ModuleIdentifier" does not affect the encodings in any of the standard encoding rules.

9.3 The ASN.1 modules shall have a "TagDefault" of **AUTOMATIC TAGS**.

9.4 In each ASN.1 module, there shall be an ASN.1 **IMPORTS** statement importing the ASN.1 type reference names in the module named **XSD** specified in Annex A that are referenced in the ASN.1 module.

9.5 The **IMPORTS** statement shall also import the ASN.1 type reference names of type assignments that have been placed (as a result of the mapping) in other ASN.1 modules but are referenced in this ASN.1 module.

9.6 There shall be no **EXPORTS** statement.

NOTE – This means that all ASN.1 type reference names in the ASN.1 module can be imported into other modules.

10 Name conversion

10.1 General

10.1.1 This Recommendation | International Standard specifies the generation of:

a) ASN.1 type reference names corresponding to the **name**s of **model group definition**s, top-level **element declaration**s, top-level **attribute declaration**s, top-level **complex type definition**s, and user-defined top-level **simple type definition**s;

b) ASN.1 identifiers corresponding to the **name**s of top-level **element declaration**s, top-level **attribute declaration**s, local **element declaration**s, and local **attribute declaration**s;

c) ASN.1 identifiers for the mapping of certain **simple type definition**s with an **enumeration** facet (see 12.4.1 and 12.4.2);

d) ASN.1 type reference names of special type assignments (see clauses 29, 30, and 31); and

e) ASN.1 identifiers of certain sequence components introduced by the mapping (see clause 20).

10.1.2 All of these ASN.1 names are generated by applying 10.3 either to the **name** of the corresponding schema component, or to a member of the **value** of an **enumeration** facet, or to a specified character string, as specified in the relevant clauses of this Recommendation | International Standard.

10.2 Generating ASN.1 type definitions that are references to ASN.1 type assignments

10.2.1 This subclause applies as explicitly invoked by other clauses of this Recommendation | International standard to generate an ASN.1 type definition that is a reference (a "DefinedType") to an ASN.1 type assignment.

10.2.2 If an ASN.1 type definition (R, say) that is a "DefinedType" is to be inserted in an ASN.1 module (M, say) other than the ASN.1 module where the referenced ASN.1 type assignment (TA, say) is being inserted, and the type reference name of TA is identical to either the type reference name of another ASN.1 type assignment being inserted in module M or to another type reference name being imported into module M, then R shall be an "ExternalTypeReference" (constructed as appropriate for module M) for TA, otherwise it shall be a "typereference" for TA.

10.3 Generating identifiers and type reference names

10.3.1 This subclause applies as explicitly invoked by other clauses of this Recommendation | International standard to generate an ASN.1 type reference name or identifier.

10.3.2 **Name**s of **attribute declaration**s, **element declaration**s, **model group definition**s, user-defined top-level **simple type definition**s, and top-level **complex type definition**s can be identical to ASN.1 reserved words or can contain characters not allowed in ASN.1 identifiers or in ASN.1 type reference names. In addition, there are cases in which ASN.1 names are required to be distinct where the **name**s of the corresponding XSD schema components (from which the ASN.1 names are mapped) are allowed to be identical.

10.3.3 The following transformations shall be applied, in order, to each character string being mapped to an ASN.1 name, where each transformation (except the first) is applied to the result of the previous transformation:

the characters " " (SPACE), "." (FULL STOP), and "_" (LOW LINE) shall all be replaced by a "-" (HYPHEN-MINUS); and

any characters except "A" to "Z" (LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z), "a" to "z" (LATIN SMALL LETTER A to LATIN SMALL LETTER Z), "0" to "9" (DIGIT ZERO to DIGIT NINE), and "-" (HYPHEN-MINUS) shall be removed; and

a sequence of two or more HYPHEN-MINUS characters shall be replaced with a single HYPHEN-MINUS; and

HYPHEN-MINUS characters occurring at the beginning or at the end of the name shall be removed; and

if a character string that is to be used as a type reference name starts with a lower-case letter, the first letter shall be capitalized (converted to upper-case); if it starts with a digit (DIGIT ZERO to DIGIT NINE), it shall be prefixed with an "**x**" (LATIN CAPITAL LETTER X) character; and

if a character string that is to be used as an identifier starts with an upper-case letter, the first letter shall be uncapitalized (converted to lower-case); if it starts with a digit (DIGIT ZERO to DIGIT NINE), it shall be prefixed with an "**x**" (LATIN SMALL LETTER X) character; and

if a character string that is to be used as a type reference name is empty, it shall be replaced by "**x**" (LATIN CAPITAL LETTER X); and

if a character string that is to be used as an identifier is empty, it shall be replaced by "**x**" (LATIN SMALL LETTER X).

10.3.4 Depending on the kind of name being generated, one of the three following subclauses applies.

10.3.4.1 If the name being generated is the type reference name of an ASN.1 type assignment and the character string generated by 10.3.3 is identical to the type reference name of another ASN.1

type assignment previously generated in the same ASN.1 module or in another ASN.1 module with the same namespace (including absence of a namespace), or is one of the reserved words specified in ITU-T Rec. X.680 | ISO/IEC 8824-1, 11.27, then a suffix shall be appended to the character string generated by 10.3.3. The suffix shall consist of a HYPHEN-MINUS followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of an integer. This integer shall be the least positive integer such that the new name is different from the type reference name of any other ASN.1 type assignment previously generated in any of those ASN.1 modules.

10.3.4.2 If the name being generated is the identifier of a component of a sequence, set, or choice type, and the character string generated by 10.3.3 is identical to the identifier of a previously generated component of the same sequence, set, or choice type, then a suffix shall be appended to the character string generated by 10.3.3. The suffix shall consist of a HYPHEN-MINUS followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of an integer. This integer shall be the least positive integer such that the new identifier is different from the identifier of any previously generated component of that sequence, set, or choice type.

10.3.4.3 If the name being generated is the "identifier" in an "EnumerationItem" of an enumerated type, and the character string generated by 10.3.3 is identical to the "identifier" in another "EnumerationItem" previously generated in the same enumerated type, then a suffix shall be appended to the character string generated by 10.3.3. The suffix shall consist of a HYPHEN-MINUS followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of an integer. This integer shall be the least positive integer such that the new identifier is different from the "identifier" in any other "EnumerationItem" already present in that ASN.1 enumerated type.

10.3.5 For an ASN.1 type reference name (or identifier) that is generated by applying this subclause 10.3 to the **name** of an **element declaration, attribute declaration,** top-level **complex type definition** or user-defined top-level **simple type definition**, if the type reference name (or identifier) generated is different from the **name**, a final **NAME** encoding instruction shall be assigned to the ASN.1 type assignment with that type reference name (or to the component with that identifier) as specified in the three following subclauses.

10.3.5.1 If the only difference is the case of the first letter (which is upper-case in the type reference name and lower-case in the **name**), then the "Keyword" in the **NAME** encoding instruction shall be **UNCAPITALIZED**.

10.3.5.2 If the only difference is the case of the first letter (which is lower-case in the identifier and upper-case in the **name**), then the "Keyword" in the **NAME** encoding instruction shall be **CAPITALIZED**.

10.3.5.3 Otherwise, the "NewName" in the **NAME** encoding instruction shall be the **name**.

EXAMPLE – The top-level **complex type definition**:

```
<xsd:complexType name="COMPONENTS">
        <xsd:sequence>
                <xsd:element name="Elem" type="xsd:boolean"/>
                <xsd:element name="elem" type="xsd:integer"/>
                <xsd:element name="Elem-1" type="xsd:boolean"/>
                <xsd:element name="elem-1" type="xsd:integer"/>
        </xsd:sequence>
</xsd:complexType>
```

is mapped to the ASN.1 type assignment:

```
COMPONENTS-1 ::= [NAME AS "COMPONENTS"] SEQUENCE {
      elem     [NAME AS CAPITALIZED] BOOLEAN,
      elem-1   [NAME AS "elem"] INTEGER,
      elem-1-1 [NAME AS "Elem-1"] BOOLEAN,
      elem-1-2 [NAME AS "elem-1"] INTEGER }
```

10.3.6 For an ASN.1 type reference name (or identifier) that is generated by applying this subclause 10.3 to the **name** of an **element declaration, attribute declaration,** top-level **complex type definition** or

user-defined top-level **simple type definition**, if the **target namespace** of the schema component is not **absent**, then a final **NAMESPACE** encoding instruction shall be assigned to the ASN.1 type assignment with that type reference name (or to the named type with that identifier) and shall specify the **target namespace** of the schema component.

10.3.7 For an ASN.1 identifier that is generated by this subclause 10.3 for the mapping of a **simple type definition** with an **enumeration** facet where the identifier generated is different from the corresponding member of the **value** of the **enumeration** facet, a final **TEXT** encoding instruction shall be assigned to the ASN.1 enumerated type, with qualifying information specifying the "identifier" in the "EnumerationItem" of the enumerated type. One of the two following subclauses applies.

10.3.7.1. If the only difference is the case of the first letter (which is lower case in the identifier and upper case in the member of the **value** of the **enumeration** facet), then the "Keyword" in the **TEXT** encoding instruction shall be **CAPITALIZED.**

10.3.7.2 Otherwise, the "NewName" in the **TEXT** encoding instruction shall be the member of the **value** of the **enumeration** facet.

10.4 Order of the mapping

10.4.1 An order is imposed on the top-level schema components of the source XSD Schema on which the mapping is performed. This applies to **model group definition**s, top-level **complex type definition**s, user-defined top-level **simple type definition**s, top-level **attribute declaration**s, and top-level **element declaration**s.

NOTE - Other top-level schema components are not mapped to ASN.1, and XSD built-in datatypes are mapped in a special way.

10.4.2 The order is specified in the three following subclauses.

10.4.2.1 Top-level schema components shall first be ordered by their **target namespace,** with the **absent** namespace preceding all namespace names in ascending lexicographical order.

10.4.2.2 Within each target namespace, top-level schema components shall be divided into four sets ordered as follows:

a) **element declaration**s;

b) **attribute declaration**s;

c) **complex type definition**s and **simple type definition**s;

d) **model group definition**s.

10.4.2.3 Within each set (see 10.4.2.2), schema components shall be ordered by **name** in ascending lexicographical order.

10.4.3 The mapping generates some ASN.1 type assignments that do not correspond directly to any XSD schema component. These are:

a) choice types (with a final **USE-TYPE** encoding instruction) corresponding to a type derivation hierarchy; the type reference names of these types have a "**-derivations**" suffix (see clause 29);

b) choice types (with a final **USE-TYPE** encoding instruction on the type and a final **USE-NIL** encoding instruction on each alternative) corresponding to a type derivation hierarchy where the user-defined top-level **simple type definition** or **complex type definition** that is the root of the derivation hierarchy is used as the **type definition** of one or more **element declaration**s that are **nillable**; the type reference names of these types have a "**-deriv-nillable**" suffix (see clause 29);

c) choice types (with a final **USE-TYPE** encoding instruction on the type and a final **DEFAULT-FOR-EMPTY** encoding instructions on each alternative) corresponding to a type derivation hierarchy where the user-defined top-level **simple type definition** or **complex type definition** that is the root of the derivation hierarchy is used as the **type definition** of one or more **element declaration**s that are not **nillable** and have a **value constraint** that is a **default** value; the type reference names of these types have a "**-deriv-default-**" suffix (see clause 29);

d) choice types (with a final **USE-TYPE** encoding instruction on the type and a final **DEFAULT-FOR-EMPTY** encoding instructions on each alternative) corresponding to a type derivation hierarchy where the user-defined top-level **simple type definition** or **complex type definition** that is the root of the derivation hierarchy

is used as the **type definition** of one or more **element declaration**s that are not **nillable** and have a **value constraint** that is a **fixed** value; the type reference names of these types have a "**-deriv-fixed-**" suffix (see clause 29);

e) choice types (with a final **USE-TYPE** encoding instruction on the type and final **USE-NIL** and **DEFAULT-FOR-EMPTY** encoding instructions on each alternative) corresponding to a type derivation hierarchy where the user-defined top-level **simple type definition** or **complex type definition** that is the root of the derivation hierarchy is used as the **type definition** of one or more **element declaration**s that are **nillable** and have a **value constraint** that is a **default** value; the type reference names of these types have a "**-deriv-nillable-default-**" suffix (see clause 29);

f) choice types (with a final **USE-TYPE** encoding instruction on the type and final **USE-NIL** and **DEFAULT-FOR-EMPTY** encoding instructions on each alternative) corresponding to a type derivation hierarchy where the user-defined top-level **simple type definition** or **complex type definition** that is the root of the derivation hierarchy is used as the **type definition** of one or more **element declaration**s that are **nillable** and have a **value constraint** that is a **fixed** value; the type reference names of these types have a "**-deriv-nillable-fixed-**" suffix (see clause 29);

g) choice types (with a final **UNTAGGED** encoding instruction) corresponding to an element substitution group; the type reference names of these types have a "**-group**" suffix (see clause 31);

h) sequence types (with a final **USE-NIL** encoding instruction) corresponding to the use of a user-defined top-level **simple type definition** or **complex type definition** as the **type definition** of one or more **element declaration**s that are **nillable**; the type reference names of these types have a "**-nillable**" suffix (see clause 30).

10.4.4 All ASN.1 type assignments that correspond directly to the XSD schema components in the source XSD Schema shall be generated before all ASN.1 type assignments listed in 10.4.3 (if any).

10.4.5 ASN.1 type assignments that correspond directly to the XSD schema components shall be generated in the order of the corresponding XSD schema components (see 10.4.1). ASN.1 type assignments listed in 10.4.3 (if any) shall be generated in the order of the XSD schema components (see 10.4.1) corresponding to the "associated type assignment" (see clauses 29, 30, and 31).

10.4.6 For 10.4.3 c) to f), if the **simple type definition** or **complex type definition** that is the root of the derivation hierarchy is used as the **type definition** of multiple **element declaration**s that have different values in the **value constraint**, the ASN.1 type assignments shall be generated in ascending lexicographical order of the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the value in the **value constraint.**

11 Mapping uses of XSD built-in datatypes

11.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International standard to generate an ASN.1 type definition corresponding to the use of an XSD built-in datatype.

11.2 A use of an XSD built-in datatype shall be mapped to an ASN.1 type definition in accordance with Table 2. The table gives the ASN.1 type definition to be used. The notation "XSD.Name" indicates that the ASN.1 type definition shall be the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the corresponding ASN.1 type assignment present in the **XSD** module.

Table 2 –ASN.1 type definitions corresponding to uses of XSD built-in datatypes

XSD built-in datatype	ASN.1 type definition
anyURI	XSD.AnyURI
anySimpleType	XSD.AnySimpleType
anyType	XSD.AnyType

XSD built-in datatype	ASN.1 type definition
base64Binary	[BASE64] OCTET STRING
boolean	BOOLEAN
byte	INTEGER (-128..127)
date	XSD.Date
dateTime	XSD.DateTime
decimal	XSD.Decimal
double	XSD.Double
duration	XSD.Duration
ENTITIES	XSD.ENTITIES
ENTITY	XSD.ENTITY
float	XSD.Float
gDay	XSD.GDay
gMonth	XSD.GMonth
gMonthDay	XSD.GMonthDay
gYear	XSD.GYear
gYearMonth	XSD.GYearMonth
hexBinary	OCTET STRING
ID	XSD.ID
IDREF	XSD.IDREF
IDREFS	XSD.IDREFS
int	XSD.Int
integer	INTEGER
language	XSD.Language
long	XSD.Long
Name	XSD.Name
NCName	XSD.NCName
negativeInteger	INTEGER (MIN..-1)
NMTOKEN	XSD.NMTOKEN
NMTOKENS	XSD.NMTOKENS
nonNegativeInteger	INTEGER (0..MAX)

XSD built-in datatype	ASN.1 type definition
nonPositiveInteger	INTEGER (MIN..0)
normalizedString	XSD.NormalizedString
NOTATION	XSD.NOTATION
positiveInteger	INTEGER (1..MAX)
QName	XSD.QName
short	XSD.Short
string	XSD.String
time	XSD.Time
token	XSD.Token
unsignedByte	INTEGER (0..255)
unsignedInt	XSD.UnsignedInt
unsignedLong	XSD.UnsignedLong
unsignedShort	XSD.UnsignedShort

12 Mapping facets

This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to map a **facet** of a **simple type definition**. A **facet** of a **simple type definition** STD is mapped to an ASN.1 constraint applied to the ASN.1 type definition corresponding to STD, unless STD has an **enumeration** facet that is being mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2). In this case, no ASN.1 constraint is generated from the facet (see 12.1.2, 12.2.1, 12.3.1, and 12.5.1).

12.1 The length, minLength, and maxLength facets

12.1.1 The **length**, **minLength**, and **maxLength** facets shall be ignored for the XSD built-in datatypes **xsd:QName** and **xsd:NOTATION** and for any **simple type definition**s derived from these by restriction.

12.1.2 If a **length**, **minLength**, or **maxLength** facet belongs to a **simple type definition** that has also an **enumeration** facet being mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2), then no "EnumerationItem"s shall be included in the "Enumeration" for the members (if any) of the **value** of the **enumeration** facet that do not satisfy the **length**, **minLength**, or **maxLength** facet..

12.1.3 Otherwise, the **length**, **minLength**, and **maxLength** facets of the **simple type definition** shall be mapped to an ASN.1 size constraint according to Table 3.

Table 3 – ASN.1 size constraints corresponding to the length, minLength, and maxLength facets

XSD facet	ASN.1 size constraint
length=*value*	(SIZE(*value*))
minLength=*min*	(SIZE(*min* .. MAX))
maxLength=*max*	(SIZE(0 .. *max*))

minLength=*min* **maxLength=***max*	**(SIZE(***min* **..** *max***))**

12.2 The pattern facet

12.2.1 If a **pattern** facet belongs to a **simple type definition** that has also an **enumeration** facet being mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2), then no "EnumerationItem"s shall be included in the "Enumeration" for the members (if any) of the **value** of the **enumeration** facet that do not satisfy the **pattern** facet.

12.2.2 Otherwise, the **pattern** facet shall be mapped to a user-defined constraint. One of the two following subclauses applies.

12.2.2.1 If the **value** of the **pattern** facet is a single regular expression, the user-defined constraint shall be:

```
(CONSTRAINED BY {/* XML representation of the XSD pattern "xyz" */})
```

where "xyz" is the XML representation of the **value** of the **pattern** facet, except that if the substring "*/" appears in the **value** of the **pattern** facet, it shall be replaced by the character string "*/".

12.2.2.2 If the **value** of the **pattern** facet is a conjunction of unions of regular expressions (the general case), the user-defined constraint is not specified (but see 12.5.5).

12.3 The whiteSpace facet

12.3.1 If a **whiteSpace** facet with a **value** of **replace** or **collapse** belongs to a **simple type definition** that has also an **enumeration** facet being mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2), then no "EnumerationItem"s shall be included in the "Enumeration" for the members (if any) of the **value** of the **enumeration** facet that contain any of the characters HORIZONTAL TABULATION, NEWLINE or CARRIAGE RETURN, or (in the case of **collapse**) contain leading, trailing, or multiple consecutive SPACE characters.

12.3.2 Otherwise, at most one of the three following subclauses applies:

12.3.2.1 If the **value** of the **whiteSpace** facet is **preserve**, then the **whiteSpace** facet shall be ignored.

12.3.2.2 If the **value** of the **whiteSpace** facet is **replace** and the ASN.1 type definition corresponding to the **simple type definition** is an ASN.1 restricted character string type, then a permitted alphabet constraint shall be added to the ASN.1 type definition to remove HORIZONTAL TABULATION, NEWLINE, and CARRIAGE RETURN characters. A final **WHITESPACE REPLACE** encoding instruction shall be assigned to the ASN.1 type definition. The following or an equivalent permitted alphabet constraint shall be used:

```
(FROM ({0, 0, 0, 32} .. {0, 16, 255, 255}))
```

12.3.2.3 If the **value** of the **whiteSpace** facet is **collapse** and the ASN.1 type definition corresponding to the **simple type definition** is an ASN.1 restricted character string type, then both a permitted alphabet constraint as specified in 12.3.2.2 and a pattern constraint that forbids leading, trailing, and multiple consecutive SPACE characters shall be added to the ASN.1 type definition. A final **WHITESPACE COLLAPSE** encoding instruction shall be assigned to the ASN.1 type definition. The following or an equivalent pattern constraint shall be used:

```
(PATTERN "([^ ]([^ ]| [^ ])*)?").
```

12.4 The enumeration facet

12.4.1 An **enumeration** facet belonging to a **simple type definition** with a **variety** of **atomic** that is derived by restriction (directly or indirectly) from **xsd:string** shall not be mapped to an ASN.1 constraint. Instead, the facet shall be mapped to the "Enumeration" of the ASN.1 enumerated type corresponding to the **simple type definition** (see 13.5) as specified in the three following subclauses.

12.4.1.1 For each member of the **value** of the **enumeration** facet, an "EnumerationItem" that is an "identifier" shall be added to the "Enumeration" (subject to 12.1.2, 12.2.1, 12.3.1, and 12.5.1).

12.4.1.2 Each "identifier" shall be generated by applying 10.3 to the corresponding member of the **value** of the **enumeration** facet.

12.4.1.3 The members of the **value** of the **enumeration** facet shall be mapped in ascending lexicographical order and any duplicate members shall be discarded.

12.4.2 An **enumeration** facet belonging to a **simple type definition** with a **variety** of **atomic** that is derived by restriction (directly or indirectly) from **xsd:integer** shall not be mapped to an ASN.1 constraint. Instead, the facet shall be mapped to the "Enumeration" of the ASN.1 enumerated type corresponding to the **simple type definition** (see 13.6) as specified in the three following subclauses.

12.4.2.1 For each member of the **value** of the **enumeration** facet, an "EnumerationItem" that is a "NamedNumber" shall be added to the "Enumeration" (subject to 12.1.2, 12.2.1, 12.3.1, and 12.5.1).

12.4.2.2 The "identifier" in each "NamedNumber" shall be generated by concatenating the character string "`int`" with the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the corresponding member of the **value** of the **enumeration** facet. The "SignedNumber" in the "NamedNumber" shall be the ASN.1 value notation for the member (an integer number).

12.4.2.3 The members of the **value** of the **enumeration** facet shall be mapped in ascending numerical order and any duplicate members shall be discarded.

12.4.3 Any other **enumeration** facet shall be mapped to an ASN.1 constraint that is either a single value or a union of single values corresponding to the members of the **value** of the **enumeration**.

NOTE – The **enumeration** facet applies to the value space of the **base type definition**. Therefore, for an **enumeration** of the XSD built-in datatypes **xsd:QName** or **xsd:NOTATION**, the value of the `uri` component of the `[USE-QNAME] SEQUENCE` produced as a single value ASN.1 constraint is determined, in the XML representation of an XSD Schema, by the namespace declarations whose scope includes the **xsd:`QName`** or **xsd:`NOTATION`**, and by the prefix (if any) of the **xsd:`QName`** or **xsd:`NOTATION`**.

EXAMPLE 1 – The following represents a user-defined top-level **simple type definition** that is a restriction of **xsd:string** with an **enumeration** facet:

```
<xsd:simpleType name="state">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="off"/>
                <xsd:enumeration value="on"/>
        </xsd:restriction>
</xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
State ::= [NAME AS UNCAPITALIZED] ENUMERATED {off, on}
```

EXAMPLE 2 – The following represents a user-defined top-level **simple type definition** that is a restriction of **xsd:integer** with an **enumeration** facet:

```
<xsd:simpleType name="integer-0-5-10">
        <xsd:restriction base="xsd:integer">
                <xsd:enumeration value="0"/>
                <xsd:enumeration value="5"/>
                <xsd:enumeration value="10"/>
        </xsd:restriction>
</xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
Integer-0-5-10 ::= [NAME AS UNCAPITALIZED] ENUMERATED {int0(0), int5(5), int10(10)}
```

EXAMPLE 3 – The following represents a user-defined top-level **simple type definition** that is a restriction of **xsd:integer** with a **minInclusive** and a **maxInclusive** facet:

```
<xsd:simpleType name="integer-1-10">
        <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="1"/>
                <xsd:maxInclusive value="10"/>
        </xsd:restriction>
</xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
Integer-1-10 ::= [NAME AS UNCAPITALIZED] INTEGER(1..10)
```

EXAMPLE 4 – The following represents a user-defined top-level **simple type definition** that is a restriction (with a **minExclusive** facet) of another **simple type definition**, derived by restriction from **xsd:integer** with the addition of a **minInclusive** and a **maxInclusive** facet:

```
<xsd:simpleType name="multiple-of-4">
        <xsd:restriction>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:integer">
                                <xsd:minInclusive value="1"/>
                                <xsd:maxInclusive value="10"/>
                        </xsd:restriction>
                </xsd:simpleType>
                <xsd:minExclusive value="5"/>
        </xsd:restriction>
</xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
Multiple-of-4 ::= [NAME AS UNCAPITALIZED] INTEGER(5<..10)
```

EXAMPLE 5 – The following represents a user-defined top-level **simple type definition** that is a restriction (with a **minLength** and a **maxLength** facet) of another **simple type definition**, derived by restriction from **xsd:string** with the addition of an **enumeration** facet:

```
<xsd:simpleType name="color">
        <xsd:restriction>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                                <xsd:enumeration value="white"/>
                                <xsd:enumeration value="black"/>
                                <xsd:enumeration value="red"/>
                        </xsd:restriction>
                </xsd:simpleType>
                <xsd:minLength value="2"/>
                <xsd:maxLength value="4"/>
        </xsd:restriction>
</xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
Color ::= [NAME AS UNCAPITALIZED] ENUMERATED {red}
```

12.5 Other facets

12.5.1 If a **totalDigits**, **fractionDigits**, **maxInclusive, maxExclusive**, **minExclusive,** or **minInclusive** facet belongs to a **simple type definition** that has also an **enumeration** facet being mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2), then no "EnumerationItem"s shall be included in the "Enumeration" for the members (if any) of the **value** of the **enumeration** facet that do not satisfy the **totalDigits**, **fractionDigits**, **maxInclusive, maxExclusive**, **minExclusive,** or **minInclusive** facet..

12.5.2 If a **maxInclusive, maxExclusive**, **minExclusive,** or **minInclusive** facet belongs to a **simple type definition** without an **enumeration** facet or with an **enumeration** facet which is not being mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2), then one of the two following subclauses applies:

12.5.2.1 If the **simple type definition** is derived by restriction (directly or indirectly) from an XSD built-in date or time datatype (**xsd:date**, **xsd:dateTime**, **xsd:duration**, **xsd:gDay**, **xsd:gMonth**, **xsd:gYear**, **xsd:gYearMonth**, **xsd:gMonthDay**, or **xsd:time**), then the **maxInclusive**, **maxExclusive**, **minExclusive,** and **minInclusive** facets of the **simple type definition** shall be mapped to an ASN.1 user-defined constraint (see 12.5.5).

12.5.2.2 Otherwise, the **maxInclusive**, **maxExclusive**, **minExclusive** and **minInclusive** facets of the **simple type definition** shall be mapped to an ASN.1 value range or single value constraint in accordance with Table 4.

Table 4 – ASN.1 constraints corresponding to the maxInclusive, maxExclusive, minExclusive, and minInclusive facets

XSD facet	ASN.1 constraint
maxInclusive=*ub*	(**MIN** .. *ub*)
maxExclusive=*ub*	(**MIN** .. < *ub*)
minExclusive=*lb*	(*lb* <.. **MAX**)
minInclusive=*lb*	(*lb* .. **MAX**)
minInclusive=*ub* **maxInclusive=***lb*	(*lb* .. *ub*)
minInclusive=*v* **maxInclusive=***v*	(*v*)
minInclusive=*ub* **maxExclusive=***lb*	(*lb* ..< *ub*)
minExclusive=*ub* **maxInclusive=***lb*	(*lb* <.. *ub*)
minExclusive=*ub* **maxExclusive=***lb*	(*lb* <..< *ub*)

12.5.4 If a **totalDigits** or **fractionDigits** facet belongs to a **simple type definition** without an **enumeration** facet or with an **enumeration** facet which is not mapped to an ASN.1 "Enumeration" (see 12.4.1 and 12.4.2), then the **totalDigits** and **fractionDigits** facets of the **simple type definition** shall be mapped to a user-defined constraint (see 12.5.5).

12.5.5 When a facet is mapped to an ASN.1 user-defined constraint, it is recommended that the facet and its **value** appear in an ASN.1 comment in the user-defined constraint. The precise form of the user-defined constraint is not specified.

13 Mapping simple type definitions

13.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International standard to generate an ASN.1 type assignment or ASN.1 type definition corresponding to a. **simple type definition.**

13.2 This clause specifies the mapping of **simple type definition**s that are not XSD built-in datatypes. The set of XSD built-in datatypes are mapped to the predefined ASN.1 module specified in Annex A (the **XSD** module), which shall be included in the ASN.1 specifications generated by the mapping.

13.3 A user-defined top-level **simple type definition** shall be mapped to an ASN.1 type assignment. The "typereference" in the "TypeAssignment" shall be generated by applying 10.3 to the **name** of the **simple type definition** and the "Type" in the "TypeAssignment" shall be an ASN.1 type definition as specified in subclauses 13.5 to 13.10.

13.4 An anonymous **simple type definition** shall be mapped to an ASN.1 type definition as specified in subclauses 13.5 to 13.10.

13.5 For a **simple type definition** with a **variety** of **atomic** with an **enumeration** facet that is derived by restriction (directly or indirectly) from **xsd:string**, the ASN.1 type definition shall be an ASN.1 enumerated type whose "Enumeration" shall be generated as specified in 12.4.1.

13.6 For a **simple type definition** with a **variety** of **atomic** with an **enumeration** facet that is derived by restriction (directly or indirectly) from **xsd:integer**, the ASN.1 type definition shall be an ASN.1 enumerated type whose "Enumeration" shall be generated as specified in 12.4.2. A final **USE-NUMBER** encoding instruction shall be assigned to the ASN.1 enumerated type.

13.7 For any other **simple type definition** (D, say) with any **variety** that is derived by restriction (directly or indirectly) from a user-defined top-level **simple type definition**, the ASN.1 type definition shall be generated by applying clause 23 to the user-defined top-level **simple type definition** (B, say) such that:

a) D is derived by restriction (directly or indirectly) from B; and

b) either B is the **base type definition** of D, or all intermediate derivation steps from B to D are anonymous **simple type definition**s.

Then, for each of the **facets** of D (if any), an ASN.1 constraint generated by applying clause 12 to the facet shall be added to the ASN.1 type definition.

13.8 For any other **simple type definition** (D, say) with a **variety** of **atomic**, the ASN.1 type definition shall be generated by applying clause 23 to the XSD built-in datatype (B, say) such that:

a) D is derived by restriction (directly or indirectly) from B; and

b) either B is the **base type definition** of D, or all intermediate derivation steps from B to D are anonymous **simple type definition**s.

Then, for each of the **facets** of D, an ASN.1 constraint generated by applying clause 12 to the facet shall be added to the ASN.1 type definition.

13.9 For any other **simple type definition** (D, say) with a **variety** of **list**, the three following subclauses apply.

13.9.1 The ASN.1 type definition shall be an ASN.1 sequence-of type whose component shall be a "Type" generated by applying clause 23 to the **item type definition**.

13.9.2 For each of the **facets** of D, an ASN.1 constraint generated by applying clause 12 to the facet shall be added to the ASN.1 sequence-of type.

13.9.3 A final `LIST` encoding instruction shall be assigned to the ASN.1 sequence-of type.

EXAMPLE – The following represents a user-defined top-level **simple type definition** that is a **list** of **xsd:float**:

```
<xsd:simpleType name="list-of-float">
        <xsd:list itemType="xsd:float"/>
</xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
List-of-float ::= [LIST] [NAME AS UNCAPITALIZED] SEQUENCE OF XSD.Float
```

13.10 For any other **simple type definition** (D, say) with a **variety** of **union**, the five following subclauses apply.

13.10.1 The ASN.1 type definition shall be an ASN.1 choice type with one alternative for each member of the **member type definitions**.

13.10.2 For each member of the **member type definitions**, the "identifier" in the "NamedType" of the corresponding alternative shall be generated by applying 10.3 either to the **name** of the member (if the member is an XSD built-in datatype or a user-defined top-level **simple type definition**) or to the character string "`alt`" (if the member is an anonymous **simple type definition**), and the "Type" in the "NamedType" shall be the ASN.1 type definition generated by applying clause 23 to the member of the **member type definitions**.

13.10.3 For each member of the **member type definitions** that is an anonymous **simple type definition**, the corresponding "NamedType" shall have a final `NAME AS ""` encoding instruction.

13.10.4 For each of the **facets** of D, an ASN.1 constraint generated by applying clause 12 to the facet shall be added to the ASN.1 choice type.

13.10.5 A final `USE-UNION` encoding instruction shall be assigned to the ASN.1 choice type.

EXAMPLE – The following represents a user-defined top-level **simple type definition** that is a **union** of two anonymous **simple type definition**s:

```
<xsd:simpleType name="decimalOrBinary">
        <xsd:union>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:decimal"/>
                </xsd:simpleType>
                <xsd:simpleType>
```

```
                <xsd:restriction base="xsd:float"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
```

It is mapped to the ASN.1 type assignment:

```
DecimalOrBinary ::= [NAME AS UNCAPITALIZED] [USE-UNION] CHOICE {
            alt         [NAME AS ""] XSD.Decimal,
            alt-1       [NAME AS ""] XSD.Float }
```

14 Mapping element declarations

14.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International standard to generate an ASN.1 type assignment or ASN.1 type definition corresponding to an **element declaration.**

NOTE - The presence of a **value constraint** on an **element declaration** normally affects the mapping. However, subclause **8.10** implies that an **element declaration** that has a **value constraint** and whose **type definition** is **xsd:QName** or **xsd:NOTATION** or a restriction of these XSD built-in datatypes is mapped as if it had no **value constraint**.

14.2 A top-level **element declaration** that is **abstract** shall be ignored.

14.3 A top-level **element declaration** that is not **abstract** shall be mapped to an ASN.1 type assignment. The "typereference" in the "TypeAssignment" shall be generated by applying 10.3 to the **name** of the **element declaration** and the "Type" in the "TypeAssignment" shall be an ASN.1 type definition as specified in 14.5.

14.4 A local **element declaration** shall be mapped to an ASN.1 type definition as specified in 14.5.

14.5 One of the two following subclauses (14.5.1 and 14.5.2) applies.

14.5.1 If the **type definition** of the **element declaration** is an anonymous **simple type definition** or **complex type definition** or an XSD built-in datatype (A, say), then one of the two following subclauses applies.

14.5.1.1 If the **element declaration** is not **nillable**, then the ASN.1 type definition shall be generated by applying clause 23 to A.

14.5.1.2 If the **element declaration** is **nillable** then the ASN.1 type definition shall be generated by applying either clause 26 (if A is a **simple type definition**) or clause 27 (if A is a **complex type definition**) to A.

14.5.2 If the **type definition** of the **element declaration** is a user-defined top-level **simple type definition** or **complex type definition** (T, say), then one of the four following subclauses applies.

14.5.2.1 If the **element declaration** is not **nillable** and does not have a substitutable **type definition** (see 14.6), then the ASN.1 type definition shall be generated by applying clause 23 to T.

14.5.2.2 If the **element declaration** is **nillable** and does not have a substitutable **type definition** (see 14.6), then the ASN.1 type definition shall be generated by applying either clause 26 (if T is a **simple type definition**) or clause 27 (if T is a **complex type definition**) to T.

14.5.2.3 If the **element declaration** is not **nillable** and has a substitutable **type definition** (see 14.6), then the ASN.1 type definition shall be generated by applying clause 24 to T.

14.5.2.4 If the **element declaration** is **nillable** and has a substitutable **type definition** (see 14.6), then the ASN.1 type definition shall be generated by applying clause 25 to T.

14.6 The phrase "has a substitutable **type definition**", applied to an **element declaration**, means that the **type definition** of the **element declaration** is a user-defined top-level **simple type definition** or **complex type definition** that occurs as the **base type definition** of another top-level **simple type definition** or **complex type definition**.

15 Mapping attribute declarations

15.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment or ASN.1 type definition corresponding to an **attribute declaration.**

15.2 A top-level **attribute declaration** shall be mapped to an ASN.1 type assignment. The "typereference" in the "TypeAssignment" shall be generated by applying 10.3 to the **name** of the **attribute declaration,** and the "Type" in the "TypeAssignment" shall be an ASN.1 type definition as specified in 15.4. A final **ATTRIBUTE** encoding instruction shall be assigned to the ASN.1 type assignment.

15.3 A local **attribute declaration** shall be mapped to an ASN.1 type definition as specified in 15.4.

15.4 The ASN.1 type definition shall be generated by applying clause 23 to the **type definition** of the **attribute declaration.**

16 Mapping values of simple type definitions

16.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 "Value" corresponding to a value in the value space of a **simple type definition.**

16.2 Given a value V in the value space of a **simple type definition**, and:

a) the ASN.1 type definition mapped from this **simple type definition**; and

b) the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of V,

V shall be mapped to an ASN.1 basic value notation for the abstract value of the ASN.1 type definition for which, in EXTENDED-XER, the canonical lexical representation is a valid "ExtendedXMLValue" encoding.

17 Mapping model group definitions

17.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment corresponding to a **model group definition.**

17.2 A **model group definition** whose **model group** has a **compositor** of **sequence** or **choice** shall be mapped to an ASN.1 type assignment. The "typereference" in the "TypeAssignment" shall be generated by applying 10.3 to the **name** of the **model group definition** and the "Type" in the "TypeAssignment" shall be generated by applying clause 18 to the **model group** of the **model group definition.**

NOTE - **Model group definition**s whose **model group** has a **compositor** of **all** are not mapped to ASN.1.

18 Mapping model groups

18.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a **model group.**

NOTE - This clause is not invoked for every **model group**. For example, a **model group** with a compositor of **all** is not mapped to ASN.1, but its **particles** are mapped as specified in 20.9.

18.2 A **model group** with a **compositor** of **sequence** shall be mapped to an ASN.1 sequence type. For each **particle** in the **model group** in order, a "NamedType" shall be generated by applying clause 19 to the **particle**, and that "NamedType" shall be added to the sequence type as one of its components. A final **UNTAGGED** encoding instruction shall be assigned to the sequence type.

18.3 A **model group** with a **compositor** of **choice** shall be mapped to an ASN.1 choice type. For each **particle** in the **model group** in order, a "NamedType" shall be generated by applying clause 19 to the **particle,** and that "NamedType" shall be added to the choice type as one of its alternatives. A final **UNTAGGED** encoding instruction shall be assigned to the choice type.

19 Mapping particles

19.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 "NamedType" corresponding to a **particle**.

NOTE - This clause is not invoked for all **particle**s. For example, the (topmost) **particle** of the **content type** of a **complex type definition** is mapped in a special way if its **term** is a **model group** with a **compositor** of **sequence** or **all** (see 20.8).

19.2 The three following subclauses define terms that are used in the remainder of this clause 19.

19.2.1 If both **min occurs** and **max occurs** of a **particle** are one, the **particle** is called a "mandatory presence particle".

19.2.2 If **min occurs** is zero and **max occurs** is one, then:

a) if the mapping of the **particle** is to generate a component of an ASN.1 sequence type, the **particle** is called an "optional presence particle";

b) otherwise, the **particle** is called an "optional single-occurrence particle".

19.2.3 If **max occurs** is two or more, the **particle** is called a "multiple-occurrence particle".

19.3 A "mandatory presence particle" or "optional presence particle" shall be mapped to a "NamedType" as specified in the two following subclauses.

19.3.1 The "identifier" in the "NamedType" shall be generated by applying 10.3 to the character string specified in 19.5 and the "Type" in the "NamedType" shall be generated by applying 19.6 to the **term** of the **particle**.

19.3.2 If the **particle** is an "optional presence particle", the "NamedType" shall be followed by the `OPTIONAL` keyword.

19.4 An "optional single-occurrence particle" or a "multiple-occurrence particle" shall be mapped to a "NamedType" as specified in the six following subclauses.

19.4.1 The "identifier" in the "NamedType" shall be generated by applying 10.3 to the character string obtained by appending the suffix `"-list"` to the character string specified in 19.5. The "Type" in the "NamedType" shall be a sequence-of type.

19.4.2 If the **particle** is an "optional single-occurrence particle" or "multiple-occurrence particle", a size constraint shall be added to the sequence-of type in accordance with Table 5.

19.4.3 If the **term** of the **particle** is an **element declaration**, then the component of the sequence-of type shall be a "NamedType". The "identifier" in this "NamedType" shall be generated by applying 10.3 to the **name** of the **element declaration** and the "Type" in this "NamedType" shall be generated by applying 19.6 to the **term** of the **particle**.

19.4.4 If the **term** of the **particle** is a **wildcard**, then the component of the sequence-of type shall be a "NamedType". The "identifier" in this "NamedType" shall be `elem` and the "Type" in this "NamedType" shall be generated by applying 19.6 to the **term** of the **particle**.

19.4.5 If the **term** of the **particle** is a **model group**, then the component of the sequence-of type shall be a "Type" and shall be generated by applying 19.6 to the term of the particle.

19.4.6 A final `UNTAGGED` encoding instruction shall be assigned to the sequence-of type.

19.5 The character string used in the generation of the "identifier" in the "NamedType" corresponding to the **particle** shall be:

a) if the **particle** is the **content type** of a **complex type definition**, the character string "content";

b) if the **term** of the **particle** is an **element declaration**, the **name** of the **element declaration**;

c) if the **term** of the **particle** is the **model group** of a **model group definition**, the **name** of the **model group definition**;

d) if the **term** of the **particle** is a **model group** with a **compositor** of **sequence** unrelated to a **model group definition**, the character string "`sequence`";

e) if the **term** of the **particle** is a **model group** with a **compositor** of **choice** unrelated to a **model group definition**, the character string "`choice`";

f) if the **term** of the **particle** is a **wildcard**, the character string "`elem`".

19.6 The "Type" in the "NamedType" corresponding to the **particle** (see 19.3) or the "Type" in the "NamedType" in the "SequenceOfType" corresponding to the **particle** (see 19.4) shall be:

a) if the **term** of the **particle** is a top-level **element declaration** which is not the head of an element substitution group, the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the ASN.1 type assignment generated by applying clause 14 to the **element declaration**;

b) if the **term** of the **particle** is a top-level **element declaration** which is the head of an element substitution group, the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the ASN.1 type assignment generated by applying clause 31.to the **element declaration**;

c) if the **term** of the **particle** is a local **element declaration,** the ASN.1 type definition generated by applying clause 14 to the **element declaration**;

d) if the **term** of the **particle** is the **model group** of a **model group definition**, the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the ASN.1 type assignment generated by applying clause 17 to the **model group definition**;

e) if the **term** of the **particle** is a **model group** unrelated to a **model group definition**, the ASN.1 type definition generated by applying clause 18 to the **model group;**

f) if the **term** of the **particle** is a **wildcard,** the ASN.1 type definition generated by applying clause 21 to the **wildcard**.

Table 5 – ASN.1 size constraint corresponding to min occurs and max occurs

min occurs and max occurs	ASN.1 size constraint
min occurs = n **max occurs** = n $n \geq 2$	`SIZE (`*n*`)`
min occurs = *min* **max occurs** = *max* *max* > *min* and *max* ≥ 2	`SIZE (`*min* `..` *max*`)`
min occurs = 0 **max occurs** = 1	`SIZE (0 .. 1)`
min occurs = *min* **max occurs** = **unbounded** *min* ≥ 1	`SIZE (`*min* `.. MAX)`
min occurs = 0 **max occurs = unbounded**	*no size constraint*

20 Mapping complex type definitions

20.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment or ASN.1 type definition corresponding to a **complex type definition.**

20.2 A top-level **complex type definition** shall be mapped to an ASN.1 type assignment. The "typereference" in the "TypeAssignment" shall be generated by applying 10.3 to the **name** of the **complex type definition** and the "Type" in the "TypeAssignment" shall be an ASN.1 type definition as specified in subclauses 20.4 to 20.11.

20.3 An anonymous **complex type definition** shall be mapped to an ASN.1 type definition as specified in subclauses 20.4 to 20.11.

20.4 The ASN.1 type definition shall be an ASN.1 sequence type. Zero or more components shall be added to the ASN.1 sequence type as specified by the following subclauses, in the specified order.

20.5 If the **content type** of the **complex type definition** is a **mixed** content model, then a component shall be added to the ASN.1 sequence type. The "identifier" in the "NamedType" of this component shall be `embed-values` and the "Type" in the "NamedType" shall be a sequence-of type whose component shall be a "Type" generated by applying clause 23 to the XSD built-in datatype **xsd:string**. A final `EMBED-VALUES` encoding instruction shall be assigned to the ASN.1 sequence type.

20.6 If the **content type** of the **complex type definition** is a **particle** whose **term** is a **model group** with a **compositor** of **all**, then a component shall be added to the ASN.1 sequence type. The "identifier" in the "NamedType" of the component shall be `order` and the "Type" in the "NamedType" shall be a sequence-of type whose component shall be an "EnumeratedType". For each **particle** of the **model group** (whose **term** is always an **element declaration**), an "EnumerationItem" that is an "identifier" identical to the "identifier" in the "NamedType" corresponding to each **particle** shall be added to the "Enumeration" in order. A final `USE-ORDER` encoding instruction shall be assigned to the ASN.1 sequence type.

NOTE - The "identifier"s in the "NamedType"s being mapped from the **particles** are generated (applying 10.3) as each component is added to the sequence type. Therefore, even though the `order` component is placed in a position that textually precedes the positions of those components within the ASN.1 sequence type, the generation of the `order` component can only be completed after all the **particles** have been mapped to sequence components.

20.7 If the **complex type definition** has **attribute uses**, then components generated by applying clause 22 to the **attribute uses** shall be added to the ASN.1 sequence type in an order based on the **target namespace** and **name** of the **attribute declaration** of each **attribute use**. The **attribute use**s shall first be ordered by **target namespace** of the **attribute declaration** (with the keyword **absent** preceding all namespace names sorted in ascending lexicographical order) and then by **name** of the **attribute declaration** within each **target namespace** (also in ascending lexicographical order).

20.8 If the **complex type definition** has an **attribute wildcard**, then a component generated from the **attribute wildcard** (see 21.3) shall be added to the ASN.1 sequence type.

20.9 If the **content type** of the **complex type definition** is a **particle**, then one of the five following subclauses applies.

20.9.1 If the **term** of the **particle** is a **model group** with a **compositor** of **sequence** whose **min occurs** and **max occurs** are both one, then, for each **particle** of the **model group** in order, then a component generated by applying clause 19 to the **particle** in the **model group** shall be added to the ASN.1 sequence type.

20.9.2 If the **term** of the **particle** is a **model group** with a **compositor** of **sequence** whose **min occurs** and **max occurs** are not both one, then a component generated by applying clause 19 to the **particle** in the **content type** shall be added to the ASN.1 sequence type.

20.9.3 If the **term** of the **particle** is a **model group** with a **compositor** of **all**, then, for each **particle** of the **model group** in order, a component generated by applying clause 19 to the **particle** of the **model group** shall be added to the ASN.1 sequence type. If the **particle** in the **content type** of the **complex type definition** has **min occurs** zero, each of the **particles** of the **model group** with **min occurs** one shall be mapped as if it had **min occurs** zero.

20.9.4 If the **term** of the **particle** is a **model group** with a **compositor** of **choice**, then a component generated by applying clause 19 to the **particle** in the **content type** shall be added to the ASN.1 sequence type.

20.10 If the **content type** of the **complex type definition** is a **simple type definition**, then a component shall be added to the ASN.1 sequence type. The "identifier" in the "NamedType" of the component shall be generated by applying 10.3 to the character string "`base`" and the "Type" in the "NamedType" shall be the ASN.1 type definition generated by applying clause 23 to the **content type**. A final `UNTAGGED` encoding instruction shall be assigned to the component.

20.11 If the **content type** of the **complex type definition** is **empty**, then no further components shall be added to the ASN.1 sequence type.

21 Mapping wildcards

21.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment or ASN.1 type definition corresponding to a **simple type definition.**

21.2 A **wildcard** that is the **term** of a **particle** shall be mapped to the ASN.1 type definition generated by applying clause 23 to the XSD built-in datatype **xsd:string**. A final **ANY-ELEMENT** encoding instruction shall be assigned to the ASN.1 type definition.

21.3 A **wildcard** that is the **attribute wildcard** of a **complex type** shall be mapped to a "NamedType". The "identifier" in the "NamedType" shall be generated by applying 10.3 to the character string "**attr**" and the "Type" in the "NamedType" shall be a sequence-of type. The component of the sequence-of type shall be a "Type" generated by applying clause 23 to the XSD built-in datatype **xsd:string**. The following user-defined constraint shall be applied to the sequence-of type:

```
(CONSTRAINED BY
     {/* Each item shall conform to the "AnyAttributeFormat" specified in
         ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
```

A final **ANY-ATTRIBUTES** encoding instruction shall be assigned to the sequence-of type.

21.4 If the **wildcard** has a **namespace constraint**, this shall be mapped to a "NameSpaceRestriction" in the **ANY-ELEMENT** or **ANY-ATTRIBUTES** encoding instruction.

22 Mapping attribute uses

22.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 "NamedType" corresponding to an **attribute use.**

22.2 An **attribute use** shall be mapped to a "NamedType".

22.3 The "identifier" in the "NamedType" shall be generated by applying 10.3 to the **name** of the **attribute declaration** of the **attribute use**, and the "Type" in the "NamedType" shall be:

a) if the **attribute use** has a top-level **attribute declaration**, the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the ASN.1 type assignment generated by applying clause 15 to the **attribute declaration**;

b) if the **attribute use** has a local **attribute declaration**, the ASN.1 type definition generated by applying clause 15 to the **attribute declaration**.

22.4 If either the **attribute use** or its **attribute declaration** has a **value constraint**, the "NamedType" shall be followed by the keyword **DEFAULT** and by a "Value" generated by applying clause 16 either to the value in the **value constraint** of the **attribute use** (if the **attribute use** has a **value constraint**), or to the value in the **value constraint** of its **attribute declaration** (otherwise).

22.5 If either the **attribute use** or its **attribute declaration** has a **value constraint** that is a **fixed** value, then an ASN.1 single value constraint with a "Value" identical to the "Value" following the **DEFAULT** keyword shall be added to the "NamedType".

22.6 If the **attribute use** is not **required** and neither the **attribute use** nor its **attribute declaration** has a **value constraint**, the "NamedType" shall be followed by the keyword **OPTIONAL**.

22.7 A final **ATTRIBUTE** encoding instruction shall be assigned to the "Type" in the "NamedType".

23 Mapping uses of simple and complex type definitions (general case)

23.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a use of a **simple type definition** or **complex type definition**. This includes their use as the **type definition** of **element declaration**s that do

not have a substitutable **type definition** (see 14.6), are not **nillable**, and may or may not have a **value constraint.**

23.2 A use of a top-level **simple type definition** that is an XSD built-in datatype shall be mapped as specified in clause 11.

23.3 A use of a user-defined top-level **simple type definition** shall be mapped to the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the ASN.1 type assignment generated by applying clause 13 to the **simple type definition**.

23.4 A use of a top-level **complex type definition** shall be mapped to the ASN.1 type definition (a "DefinedType") generated by applying 10.2 to the ASN.1 type assignment generated by applying clause 20 to the **complex type definition**.

23.5 A use of an anonymous **simple type definition** is not distinguished from the **simple type definition** itself, and shall be mapped as specified in clause 13 for the **simple type definition**.

23.6 A use of an anonymous **complex type definition** is not distinguished from the **complex type definition** itself, and shall be mapped as specified in clause 20 for the **complex type definition**.

23.7 If a **simple type definition** or **complex type definition** is used as the **type definition** of an **element declaration** with a **value constraint**, then a final `DEFAULT-FOR-EMPTY` encoding instruction shall be assigned to the ASN.1 type definition, and one of the three following subclauses applies.

23.7.1 For a **simple type definition**, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

23.7.2 For a **complex type definition** whose **content type** is a **simple type definition**, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

23.7.3 For a **complex type definition** with a **mixed** content type, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of **xsd:string** with **whiteSpace preserve**.

23.8 If a **simple type definition** or **complex type definition** is used as the **type definition** of an **element declaration** with a **value constraint** that is a **fixed** value, then one of the three following subclauses applies.

23.8.1 For a **simple type definition**, an ASN.1 single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be added to the ASN.1 definition.

23.8.2 For a **complex type definition** whose **content type** is a **simple type definition**, an ASN.1 inner subtype constraint shall be added to the the ASN.1 definition and shall apply to the `base` component a single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction.

23.8.3 For a **complex type definition** with a **mixed** content type, an ASN.1 inner subtype constraint shall be added to the the ASN.1 definition and shall apply:

a) to the `embed-values` component, an ASN.1 single value constraint with a "Value" consisting in a single occurrence of a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction;

b) to each component that is `OPTIONAL` and does not have a final `ATTRIBUTE` encoding instruction, the keyword `ABSENT`; and

c) to each component whose type is a sequence-of type, a `SIZE(0)` constraint.

24 Mapping special uses of simple and complex type definitions (substitutable)

24.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a **simple type definition** or

complex type definition used as the **type definition** of **element declaration**s that have a substitutable **type definition** (see 14.6), are not **nillable**, and may or may not have a **value constraint**.

24.2 A use of a **simple type definition** (STD, say) or **complex type definition** (CTD, say) shall be mapped to an ASN.1 choice type.

24.3 One alternative shall be added to the ASN.1 choice type for STD or CTD itself and one alternative shall be added for each user-defined top-level **simple type definition** and **complex type definition** in the source XSD schema that is derived by restriction or extension (directly or indirectly) from STD or CTD.

24.4 For each alternative, the "identifier" in the "NamedType" shall be generated by applying 10.3 to the **name** of the **simple type definition** or **complex type definition** corresponding to the alternative, and the "Type" in the "NamedType" shall be the ASN.1 type definition generated by applying clause 23 to the **simple type definition** or **complex type definition**.

24.5 The first alternative added to the choice type shall be the one corresponding to STD or CTD itself. The subsequent alternatives shall be added to the choice type in an order based on the **target namespace** and **name** of the **simple type definition**s and **complex type definition**s. Type definitions shall first be ordered by **target namespace** (with the **absent** namespace preceding all namespace names sorted in ascending lexicographical order) and then by **name** (also in ascending lexicographical order) within each **target namespace**.

24.6 A final `USE-TYPE` encoding instruction shall be assigned to the ASN.1 choice type.

24.7 If there is a **value constraint**, then a final `DEFAULT-FOR-EMPTY` encoding instruction shall be assigned to each alternative of the ASN.1 choice type. One of the three following subclauses applies.

24.7.1 If the alternative corresponds to a **simple type definition**, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

24.7.2 If the alternative corresponds to a **complex type definition** whose **content type** is a **simple type definition**, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

24.7.3 If the alternative corresponds to a **complex type definition** with a **mixed** content type, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of **xsd:string** with **whiteSpace preserve**.

24.8 If there is a **value constraint** that is a **fixed** value, then an ASN.1 inner subtype constraint shall be added to the ASN.1 choice type. One of the three following subclauses applies.

24.8.1 If the alternative corresponds to a **simple type definition**, the inner subtype constraint shall apply to the alternative an ASN.1 single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction.

24.8.2 If the alternative corresponds to a **complex type definition** whose **content type** is a **simple type definition**, the inner subtype constraint shall apply to the alternative another ASN.1 inner subtype constraint that applies to the `base` component a single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction.

24.8.3 If the alternative corresponds to a **complex type definition** with a **mixed** content type, the inner subtype constraint shall apply to the alternative another ASN.1 inner subtype constraint that applies:

a) to the `embed-values` component, an ASN.1 single value constraint with a "Value" consisting in a single occurrence of a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction;

b) to each component that is **OPTIONAL** and does not have a final **ATTRIBUTE** encoding instruction, the keyword **ABSENT**; and

c) to each component whose type is a sequence-of type, a **SIZE(0)** constraint.

25 Mapping special uses of simple and complex type definitions (substitutable, nillable)

25.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a **simple type definition** or **complex type definition** used as the **type definition** of **element declaration**s that have a substitutable **type definition** (see 14.6), are **nillable**, and may or may not have a **value constraint**.

25.2 A use of a **simple type definition** (STD, say) or **complex type definition** (CTD, say) shall be mapped to an ASN.1 choice type.

25.3 One alternative shall be added to the ASN.1 choice type for STD or CTD itself and one alternative shall be added for each user-defined top-level **simple type definition** and **complex type definition** in the source XSD schema that is derived by restriction or extension (directly or indirectly) from STD or CTD.

25.4 For each alternative, the "identifier" in the "NamedType" shall be generated by applying 10.3 to the **name** of the **simple type definition** or **complex type definition** corresponding to the alternative, and the "Type" in the "NamedType" shall be the ASN.1 type definition (a "DefinedType") generated by applying clause 10.2 to the ASN.1 type assignment generated by applying clause 30 to the **simple type definition** or **complex type definition**.

25.5 The first alternative added to the choice type shall be the one corresponding to STD or CTD itself. The subsequent alternatives shall be added to the choice type in an order based on the **target namespace** and **name** of the **simple type definition**s and **complex type definition**s. Type definitions shall first be ordered by **target namespace** (with the **absent** namespace preceding all namespace names sorted in ascending lexicographical order) and then by **name** (also in ascending lexicographical order) within each **target namespace**.

25.6 A final **USE-TYPE** encoding instruction shall be assigned to the ASN.1 choice type.

25.7 If there is a **value constraint**, then a final **DEFAULT-FOR-EMPTY** encoding instruction shall be assigned to each alternative of the ASN.1 choice type. One of the three following subclauses applies.

25.7.1 If the alternative corresponds to a **simple type definition**, the "Value" in the final **DEFAULT-FOR-EMPTY** encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

25.7.2 If the alternative corresponds to a **complex type definition** whose **content type** is a **simple type definition**, the "Value" in the final **DEFAULT-FOR-EMPTY** encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

25.7.3 If the alternative corresponds to a **complex type definition** with a **mixed** content type, the "Value" in the final **DEFAULT-FOR-EMPTY** encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of **xsd:string** with **whiteSpace preserve**.

25.8 If there is a **value constraint** that is a **fixed** value, then an ASN.1 inner subtype constraint shall be added to the ASN.1 choice type. One of the three following subclauses applies.

25.8.1 If the alternative corresponds to a **simple type definition**, the inner subtype constraint shall apply to the alternative (which is an ASN.1 sequence type with a final **USE-NIL** encoding instruction) another ASN.1 inner subtype constraint which in turn shall apply to the **content** component the keyword **PRESENT** and an ASN.1 single value constraint with a "Value" identical to the "Value" in the final **DEFAULT-FOR-EMPTY** encoding instruction.

25.8.2 If the alternative corresponds to a **complex type definition** whose **content type** is a **simple type definition**, the inner subtype constraint shall apply to the alternative (which is an ASN.1 sequence type with a final `USE-NIL` encoding instruction) another ASN.1 inner subtype constraint that applies to the `content` component the keyword `PRESENT` and an ASN.1 single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction.

25.8.3 If the alternative corresponds to a **complex type definition** with a **mixed** content type, the inner subtype constraint shall apply to the alternative (which is an ASN.1 sequence type with a final `USE-NIL` encoding instruction) another ASN.1 inner subtype constraint that applies:

a) to the `embed-values` component, an ASN.1 single value constraint with a "Value" consisting in a single occurrence of a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction;

b) to the `content` component (which is an ASN.1 sequence type), the keyword `PRESENT` and another inner subtype constraint that applies the keyword `ABSENT` to each of its components that is `OPTIONAL` and a `SIZE(0)` constraint to each of its components whose type is a sequence-of type;

c) to each component that is `OPTIONAL` and does not have a final `ATTRIBUTE` encoding instruction, the keyword `ABSENT`; and

d) to each component whose type is a sequence-of type, a `SIZE(0)` constraint.

26 Mapping special uses of simple type definitions (nillable)

26.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a **simple type definition** used as the **type definition** of **element declaration**s that do not have a substitutable **type definition** (see 14.6), are **nillable**, and may or may not have a **value constraint**.

26.2 A use of a **simple type definition** shall be mapped to an ASN.1 sequence type with one `OPTIONAL` component.

26.3 The "identifier" in the "NamedType" of the component shall be `content` and the "Type" in the "NamedType" shall be the ASN.1 type definition generated by applying clause 23 to the **simple type definition**.

26.4 A final `USE-NIL` encoding instruction shall be assigned to the ASN.1 sequence type.

26.5 If there is a **value constraint**, then a final `DEFAULT-FOR-EMPTY` encoding instruction shall be assigned to the ASN.1 sequence type. The "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint**.

26.6 If there is a **value constraint** that is a **fixed** value, then an ASN.1 inner subtype constraint shall be added to the ASN.1 sequence type. The inner subtype constraint shall apply to the `content` component an ASN.1 single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction. The inner subtype constraint shall also apply the keyword `PRESENT` to the `content` component.

27 Mapping special uses of complex type definitions (nillable)

27.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a **complex type definition** used as the **type definition** of **element declaration**s that do not have a substitutable **type definition** (see 14.6), are **nillable**, and may or may not have a **value constraint**.

27.2 A use of a **complex type definition** shall be mapped to an ASN.1 sequence type. One or more components shall be added to the ASN.1 sequence type as specified by the following subclauses, in the specified order.

27.3 If the **content type** of the **complex type definition** is a **mixed** content model, then an `embed-values` component shall be added to the ASN.1 sequence type as specified in 20.5.

27.4 If the **content type** of the **complex type definition** is a **particle** whose **term** is a **model group** with a **compositor** of **all**, then an `order` component shall be added to the ASN.1 sequence type as specified in 20.6.

27.5 If the **complex type definition** has **attribute uses**, components mapped from the **attribute uses** shall be added to the ASN.1 sequence type as specified in 20.7.

27.6 If the **complex type definition** has an **attribute wildcard**, then a component generated from the **attribute wildcard** shall be added to the ASN.1 sequence type. as specified in 20.8.

27.7 If the **content type** of the **complex type definition** is a **particle**, then one of the two following subclauses applies.

27.7.1 If the **term** of the **particle** is a **model group** with a **compositor** of **sequence** or **choice**, then an `OPTIONAL` component shall be added to the ASN.1 sequence type. The "identifier" in the "NamedType" of the component shall be generated by applying 10.3 to the character string "`content`" and the "Type" in the "NamedType" shall be an ASN.1 sequence type with a single component, which shall be generated by applying clause 19 to the **particle** in the **content type**.

27.7.2 If the **term** of the **particle** is a **model group** with a **compositor** of **all**, then an `OPTIONAL` component shall be added to the ASN.1 sequence type. The "identifier" in the "NamedType" of the component shall be generated by applying 10.3 to the character string "`content`" and the "Type" in the "NamedType" shall be an ASN.1 sequence type. For each **particle** of the **model group** in order, a component generated by applying clause 19 to the **particle** of the **model group** shall be added to the inner ASN.1 sequence type. If the **particle** in the **content type** of the **complex type definition** has **min occurs** zero, each of the **particles** of the **model group** with **min occurs** one shall be mapped as if it had **min occurs** zero.

27.8 If the **content type** of the **complex type definition** is a **simple type definition**, then an `OPTIONAL` component shall be added to the ASN.1 sequence type. The "identifier" in the "NamedType" of the component shall be generated by applying 10.3 to the character string "`content`" and the "Type" in the "NamedType" shall be the ASN.1 type definition generated by applying clause 23 to the **content type.**

27.9 If the **content type** of the **complex type definition** is **empty**, then no further components shall be added to the ASN.1 sequence type.

27.10 A final `USE-NIL` encoding instruction shall be assigned to the ASN.1 sequence type.

27.11 If there is a **value constraint**, then a final `DEFAULT-FOR-EMPTY` encoding instruction shall be assigned to the ASN.1 sequence type. One of the two following subclauses applies.

27.11.1 If the **content type** of the **complex type definition** is a **simple type definition**, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of the **simple type definition**.

27.11.2 If the **content type** of the **complex type definition** is a **mixed** content type, the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction shall be generated by applying clause 16 to the value in the **value constraint** considered as a value in the value space of **xsd:string** with **whiteSpace preserve**.

27.12 If there is a **value constraint** that is a **fixed** value, then an ASN.1 inner subtype constraint shall be added to the ASN.1 sequence type. The inner subtype constraint shall apply the keyword `PRESENT` to the `content` component. One of the two following subclauses applies.

27.12.1 If the **content type** of the **complex type definition** is a **simple type definition**, the inner subtype constraint shall apply to the `content` component an ASN.1 single value constraint with a "Value" identical to the "Value" in the final `DEFAULT-FOR-EMPTY` encoding instruction.

27.12.2 If the **content type** of the **complex type definition** is a **mixed** content type, the inner subtype constraint shall apply:

a) to the **embed-values** component, an ASN.1 single value constraint with a "Value" consisting in a single occurrence of a "Value" identical to the "Value" in the final **DEFAULT-FOR-EMPTY** encoding instruction;

b) to each component of the **content** component (an ASN.1 sequence type) that is **OPTIONAL**, the keyword **ABSENT**;

c) to each component of the **content** component (an ASN.1 sequence type) whose type is a sequence-of type, a **SIZE(0)** constraint.

28 Mapping special uses of element declarations (head of element substitution group)

28.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type definition corresponding to a top-level **element declaration** that is the head of an element substitution group and is used as the **term** of **particle**s.

28.2 A use of a top-level **element declaration** shall be mapped to an ASN.1 choice type.

28.3 One alternative shall be added to the ASN.1 choice type for the top-level **element declaration** itself (H, say) and one alternative shall be added for each top-level **element declaration** in the source XSD Schema that is not **abstract** and whose **substitution group affiliation** is H.

28.4 For each alternative, the "identifier" in the "NamedType" shall be generated by applying 10.3 to the **name** of the top-level **element declaration** corresponding to the alternative, and the "Type" in the "NamedType" shall be the ASN.1 type definition (a "DefinedType") generated by applying clause 10.2 to the ASN.1 type assignment generated by applying clause 14 to the top-level **element declaration**.

NOTE - In XSD, substitution group membership is transitive, i.e., the members of a substitution group ESG1 whose head is a member of another substitution group ESG2 are all also members of ESG2.

28.5 Alternatives shall be added to the choice type in an order based on the **target namespace** and **name** of the top-level **element declaration**s. The **element declaration**s shall first be ordered by **target namespace** (with the **absent** namespace preceding all namespace names sorted in ascending lexicographical order) and then by **name** (also in ascending lexicographical order) within each **target namespace**.

NOTE - The **element declaration** that is the head of the element substitution group is ordered together with the other **element declarations** that belong to the element substitution group.

28.6 A final **UNTAGGED** encoding instruction shall be assigned to the choice type.

29 Generating special ASN.1 type assignments for element declarations

29.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment corresponding to a user-defined top-level **simple type definition** or **complex type definition** used as the **type definition** of **element declaration**s that have a substitutable **type definition** (see 14.6) or are **nillable**.

29.2 This clause 29 is invoked by other clauses for a given combination of:

a) a **simple type definition** or **complex type definition**;

b) whether the **element declaration**s has a substitutable **type definition** (see 14.6);

c) whether the **element declaration** is **nillable**; and

d) whether the **element declaration** has a **value constraint** and the kind and value of the **value constraint;**

and generates an ASN.1 type assignment (called a "special ASN.1 type assignment (for element declarations)") for a combination of the above items.

29.3 One and only one special ASN.1 type assignment shall be generated for each different combination of the above items that actually occurs in one or more invocations of this clause 29 over the mapping of a source XSD Schema.

NOTE - For example, if two or more **element declarations** in a large XSD Schema have identical **type definitions**, are both **nillable**, and both have a **value constraint** that is a **default** value and is the same value, then a single special ASN.1 type assignment is generated. The type reference name of this type assignment will occur in the "Type" in the "TypeAssignment"s corresponding to both **element declaration**s.

29.4 The term "associated ASN.1 type assignment" designates the ASN.1 type assignment being mapped from the **simple type definition** or **complex type definition** that is the **type definition** of the **element declaration** for which a special ASN.1 type assignment is generated, by applying clause 13 or clause 20, respectively.

NOTE - Any special ASN.1 type assignment has an associated ASN.1 type assignment, as this clause 29 applies only when the **type definition** of an **element declaration** is a user-defined top-level **simple type definition** or **complex type definition**. All such **simple type definitions** and **complex type definitions** are mapped to ASN.1 type assignments.

29.5 For a given **element declaration**, the "typereference" in the "TypeAssignment" for a special ASN.1 type assignment shall be constructed by appending a suffix to the type reference name of the associated ASN.1 type assignment and applying 10.3 to the resulting character string, and the "Type" in the "TypeAssignment" shall be the ASN.1 type definition generated by applying either clause 24 or clause 25 to the **simple type definition** or **complex type definition** that is the **type definition** of the **element declaration**. One of the following applies:

a) if the **element declaration** is not **nillable**, has a substitutable **type definition**, and does not have a **value constraint**, the suffix shall be "`-derivations`" and clause 24 shall be applied;

b) if the **element declaration** is **nillable**, has a substitutable **type definition**, does not have a **value constraint**, the suffix shall be "`-deriv-nillable`" and clause 25 shall be applied;

c) if the **element declaration** is not **nillable**, has a substitutable **type definition**, and has a **value constraint** that is a **default** value, the suffix shall be "`-deriv-default-`" followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the value in the **value constraint,** and clause 24 shall be applied;

d) if the **element declaration** is not **nillable**, has a substitutable **type definition**, and has a **value constraint** that is a **fixed** value, the suffix shall be "`-deriv-fixed-`" followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the value in the **value constraint**, and clause 24 shall be applied;

e) if the **element declaration** is **nillable**, and has a substitutable **type definition**, and has a **value constraint** that is a **default** value, the suffix shall be "`-deriv-nillable-default-`" followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the value in the **value constraint,** and clause 25 shall be applied;

f) if the **element declaration** is **nillable**, has a substitutable **type definition**, and has a **value constraint** that is a **fixed** value, the suffix shall be "`-deriv-nillable-fixed-`" followed by the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the value in the **value constraint,** and clause 25 shall be applied.

30 Generating special ASN.1 type assignments for type definitions

30.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment corresponding to a user-defined top-level **simple type definition** or **complex type definition** that belongs to the derivation hierarchy of the **type definition** of **element declaration**s that have a substitutable **type definition** (see 14.6) and are **nillable**.

30.2 This clause 30 is invoked by other clauses for a given **simple type definition** or **complex type definition** and generates an ASN.1 type assignment (called a "special ASN.1 type assignment (for a type definition)").

30.3 One and only one special ASN.1 type assignment shall be generated for each **simple type definition** or **complex type definition** that actually occurs in one or more invocations of this clause 30 over the mapping of a source XSD Schema.

30.4 The term "associated ASN.1 type assignment" designates the ASN.1 type assignment being mapped from the **simple type definition** or **complex type definition** by applying clause 13 or clause 20, respectively.

30.5 The "typereference" in the "TypeAssignment" for a special ASN.1 type assignment shall be constructed by appending the suffix "`-nillable`" to the type reference name of the associated ASN.1 type assignment and applying 10.3 to the resulting character string, and the "Type" in the "TypeAssignment" shall be the ASN.1 type definition generated by applying either clause 26 or clause 27 to the **simple type definition** or **complex type definition,** respectively.

31 Generating special ASN.1 type assignments for element substitution groups

31.1 This clause applies as explicitly invoked by other clauses of this Recommendation | International Standard to generate an ASN.1 type assignment corresponding to a **particle** whose **term** is a top-level **element declaration** that is the head of an element substitution group.

31.2 This clause 31 is invoked by other clauses for a top-level **element declaration** that is the head of an element substitution group and generates an ASN.1 type assignment (called a "special ASN.1 type assignment (for an element substitution group)").

31.3 One and only one special ASN.1 type assignment shall be generated for each top-level **element declaration** that actually occurs in one or more invocations of this clause 31 over the mapping of a source XSD Schema.

31.4 The term "associated ASN.1 type assignment" designates the ASN.1 type assignment being mapped from the top-level **element declaration** by applying clause 14.

31.5 The "typereference" in the "TypeAssignment" for a special ASN.1 type assignment shall be constructed by appending the suffix "`-group`" to the type reference name of the associated ASN.1 type assignment and applying 10.3 to the resulting character string, and the "Type" in the "TypeAssignment" shall be the ASN.1 type definition generated by applying clause 28 to the top-level **element declaration**.

Annex A

ASN.1 type definitions corresponding to XSD built-in datatypes

(This annex forms an integral part of this Recommendation | International Standard)

A.1 This annex specifies a module that defines the ASN.1 types that correspond to the XSD built-in datatypes and that are used for the mapping from W3C XML Schema to ASN.1.

A.2 W3C XML Schema defines many built-in date and time datatypes to represent durations, instants or recurring instants. Although they are all derived from ISO 8601, there are some extensions and restrictions. The XSD built-in date and time datatypes are mapped to **visibleString** with a user-defined constraint referencing the applicable XSD clause. A permitted alphabet constraint is added to provide a more efficient encoding with the Packed Encoding Rules (PER), since user-defined constraints are not PER-visible (and hence are not used in optimizing encodings).

A.3 The **XSD** module is:

```
XSD {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(1)}
DEFINITIONS
AUTOMATIC TAGS ::=
BEGIN

/* xsd:anySimpleType */

AnySimpleType ::= XMLCompatibleString

/* xsd:anyType */

AnyType ::= SEQUENCE {
      embed-values SEQUENCE OF String,
      attr SEQUENCE
        (CONSTRAINED BY {
          /* Each item shall conform to the "AnyAttributeFormat" specified
           in ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */ } ) OF String,
      elem-list SEQUENCE OF elem String
        (CONSTRAINED BY {
           /* Shall conform to the "AnyElementFormat" specified
              in ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 19 */ } ) }
      (CONSTRAINED BY {
      /* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */ } )

/* xsd:anyUri */

AnyURI ::= XMLStringWithNoCRLFHT
      (CONSTRAINED BY {
          /* The XMLStringWithNoCRLFHT shall be a valid URI as defined in IETF RFC
             2396 */ } )

/* xsd:base64Binary */

Base64Binary ::= OCTET STRING

/* xsd:boolean */

Boolean ::= BOOLEAN

/* xsd:byte */

Byte ::= INTEGER (-128..127)

/* xsd:date */

Date ::= DATE-TIME (DateOnly)

/* xsd:dateTime */

DateTime ::= DATE-TIME

/* xsd:decimal */

Decimal ::= REAL (WITH COMPONENTS {..., base(10)})
```

```
                (ALL EXCEPT(-0 | MINUS-INFINITY | PLUS-INFINITY | NOT-A-NUMBER))

/* xsd:double */

Double ::=  REAL (WITH COMPONENTS {
                      mantissa(-9007199254740991..9007199254740991),
                      base(2),
                      exponent(-1075..970)})

/* xsd:duration */

Duration ::= DURATION

/* xsd:ENTITIES */

ENTITIES ::= SEQUENCE (SIZE(1..MAX)) OF ENTITY

/* xsd:ENIITY */

ENTITY ::= NCName

/* xsd:float */

Float ::=  REAL (WITH COMPONENTS {
                      mantissa(-16777215..16777215),
                      base(2),
                      exponent(-149..104)})

/* xsd:gDay */

GDay ::= DATE-TIME (Day)

/* xsd:gMonth */

GMonth ::= DATE-TIME (Month)

/* xsd:gMonthDay */

GMonthDay ::= DATE-TIME (MonthDay)

/* xsd:gYear */

GYear ::= DATE-TIME (Year)

/* xsd:gYearMonth */

GYearMonth ::= DATE-TIME (YearMonth)

/* xsd:hexBinary */

HexBinary ::= OCTET STRING

/* xsd:ID */

ID ::= NCName

/* xsd:IDREF */

IDREF ::= NCName

/* xsd:IDREFS */

IDREFS ::= SEQUENCE (SIZE(1..MAX)) OF IDREF

/* xsd:int */

Int ::= INTEGER (-2147483648..2147483647)

/* xsd:integer */

Integer ::= INTEGER

/* xsd:language */

Language ::= VisibleString (FROM ("a".."z" | "A".."Z" | "-" | "0".."9"))
      (PATTERN
       "[a-zA-Z]#(1,8)(-[a-zA-Z0-9]#(1,8))*")

       /* The semantics of Language is specified in IETF RFC 3066 */

/* xsd:long */

Long ::= INTEGER (-9223372036854775808..9223372036854775807)
```

```
        /* xsd:name */

        Name ::= Token (XMLStringWithNoWhitespace)
              (CONSTRAINED BY {
                  /* The Token shall be a Name as defined in W3C XML 1.0, 2.3 */ } )

        /* xsd:NCName */

        NCName ::= Name
              (CONSTRAINED BY {
                    /* The Name shall be an NCName as defined in W3C XML Namespaces, 2 */ }
)

        /* xsd:negativeInteger */

        NegativeInteger ::= INTEGER (MIN..-1)

        /* xsd:NMTOKEN */

        NMTOKEN ::= Token (XMLStringWithNoWhitespace)
              (CONSTRAINED BY {
                   /* The Token shall be an NMTOKEN as defined in W3C XML 1.0, 2.3 */ } )

        /* xsd:NMTOKENS */

        NMTOKENS ::= SEQUENCE (SIZE(1..MAX)) OF NMTOKEN

        /* xsd:nonNegativeInteger */

        NonNegativeInteger ::= INTEGER (0..MAX)

        /* xsd:nonPositiveInteger */

        NonPositiveInteger ::= INTEGER (MIN..0)

        /* xsd:normalizedString */

        NormalizedString ::= String (XMLStringWithNoCRLFHT)
              (CONSTRAINED BY {
                  /* The String shall be a normalizedString as defined in W3C XML Schema
              Part 2, 3.3.1 */})

        /* xsd:NOTATION */

        NOTATION ::= QName

        /* xsd:positiveInteger */

        PositiveInteger ::= INTEGER (1..MAX)

        /* xsd:QName */

        QName ::= SEQUENCE {
              uri   AnyURI OPTIONAL,
              name  NCName }

        /* xsd:short */

        Short ::= INTEGER (-32768..32767)

        /* xsd:string */

        String ::=  XMLCompatibleString

        /* xsd:time */

        Time ::= DATE-TIME (TimeOnly)

        /* xsd:token */

        Token ::= NormalizedString (CONSTRAINED BY {
              /* The NormalizedString shall be a token as defined in W3C XML Schema Part 2,
              3.3.2 */})

        /* xsd:unsignedByte */

        UnsignedByte ::=  INTEGER (0..255)

        /* xsd:unsignedInt */

        UnsignedInt ::=  INTEGER (0..4294967295)
```

```
/* xsd:unsignedLong */

UnsignedLong ::=  INTEGER (0..18446744073709551615)

/* xsd:unsignedShort */

UnsignedShort ::= INTEGER (0..65535)


/* ASN.1 type definitions supporting the mapping of W3C XML Schema built-in
datatypes */

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

XMLStringWithNoWhitespace ::= UTF8String (FROM(
      {0, 0, 0, 33} .. {0, 0, 215, 255} |
      {0, 0, 224, 0} .. {0, 0, 255, 253} |
      {0, 1, 0, 0} .. {0, 16, 255, 253}))

XMLStringWithNoCRLFHT ::= UTF8String (FROM(
      {0, 0, 0, 32} .. {0, 0, 215, 255} |
      {0, 0, 224, 0} .. {0, 0, 255, 253} |
      {0, 1, 0, 0} .. {0, 16, 255, 253}))


/* ASN.1 type definitions supporting the mapping of W3C XML Schema built-in date
and time datatypes */

DURATION ::= VisibleString   (FROM ("0".."9" | "DHMPSTY:.+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.6 */})

DATE-TIME ::= VisibleString  (FROM ("0".."9" | "TZ:.+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.7 */})

DateOnly ::= DATE-TIME (FROM ("0".."9" | "Z:+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.9 */})

Day ::= DATE-TIME      (FROM ("0".."9" | "Z:+-"))
                       (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.13 */})

Month ::= DATE-TIME    (FROM ("0".."9" | "Z:+-"))

                 (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.14 */})

MonthDay ::= DATE-TIME(FROM ("0".."9" | "Z:+-"))
                       (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.12 */})

Year ::= DATE-TIME     (FROM ("0".."9" | "Z:+-"))
                       (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.11 */})

YearMonth ::= DATE-TIME      (FROM ("0".."9" | "Z:+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.10 */})

TimeOnly ::= DATE-TIME       (FROM ("0".."9" | "Z:.+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.8 */})

ENCODING-CONTROL XER
           GLOBAL-DEFAULTS MODIFIED-ENCODINGS
           GLOBAL-DEFAULTS CONTROL-NAMESPACE
                 "http://www.w3.org/2001/XMLSchema-instance"
                 PREFIX "xsi"
           NAMESPACE ALL, ALL IN ALL AS
                 "http://www.w3.org/2001/XMLSchema"
                 PREFIX "xsd"
           USE-QNAME QName
           BASE64 Base64Binary
           DECIMAL Decimal
           LIST ENTITIES, IDREFS, NMTOKENS
           EMBED-VALUES AnyType
           ANY-ATTRIBUTES AnyType.any-attributes
```

```
            ANY-ELEMENT AnyType.any-elements.*
            UNTAGGED AnyType.any-elements
            NAME AnySimpleType, AnyURI, Base64Binary, Boolean,
                  Byte, Date, DateTime, Decimal, Double, Duration,
                  Float, GDay, GMonth, GMonthDay, GYear, GYearMonth,
                  HexBinary, Int, Integer, Language, Long,
                  NegativeInteger, NonNegativeInteger, NonPositiveInteger,
                  NormalizedString, PositiveInteger, Short,
                  String, Time, Token,
                  UnsignedByte, UnsignedInt, UnsignedLong, UnsignedShort
                  AS UNCAPITALIZED
            WHITESPACE AnyURI, Language, Token, DURATION, DATE-TIME COLLAPSE
            WHITESPACE NormalizedString REPLACE

END
```

Annex B

Assignment of object identifier values

(This annex does not form an integral part of this Recommendation | International Standard)

The following object identifier and object descriptor value is assigned in this Recommendation | International Standard:

For the module defining ASN.1 types corresponding to the XSD built-in datatypes:

```
{ joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(1) }
```

"ASN.1 XSD Module"

Annex C

Examples of mappings

(This annex does not form an integral part of this Recommendation | International Standard)

This annex illustrates the mapping specified in this Recommendation | International Standard by giving an ASN.1 module corresponding to an XSD Schema.

C.1 A Schema using simple type definitions

The following Schema contains examples of XSD built-in datatypes (**xsd:string**, **xsd:decimal**, **xsd:integer**, **xsd:int**, **xsd:date**), other **simple type definition**s and **complex type definition**s.

```
<?xml version="1.0" encoding="UTF-8"?>
	<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
		<xsd:element name="EXAMPLES">
			<xsd:complexType>
				<xsd:sequence>
					<xsd:element ref="personnelRecord"/>
					<xsd:element name="decimal" type="xsd:decimal"/>
					<xsd:element name="daysOfTheWeek" type="ListOfDays"/>
					<xsd:element ref="namesOfMemberNations"/>
					<xsd:element ref="fileIdentifier" maxOccurs="unbounded"/>
				</xsd:sequence>
			</xsd:complexType>
		</xsd:element>
		<xsd:element name="personnelRecord">
			<xsd:complexType>
				<xsd:sequence>
					<xsd:element name="name" type="name"/>
					<xsd:element name="title" type="xsd:string"/>
					<xsd:element name="decimal" type="xsd:integer"/>
					<xsd:element name="dateOfHire" type="xsd:date"/>
					<xsd:element ref="nameOfSpouse"/>
					<xsd:element ref="children"/>
				</xsd:sequence>
			</xsd:complexType>
		</xsd:element>
		<xsd:element name="nameOfSpouse" type="name"/>
		<xsd:complexType name="name">
			<xsd:sequence>
				<xsd:element name="givenName" type="xsd:string"/>
				<xsd:element name="initial" type="xsd:string"/>
				<xsd:element name="familyName" type="xsd:string"/>
			</xsd:sequence>
		</xsd:complexType>
		<xsd:element name="children">
			<xsd:complexType>
				<xsd:sequence>
					<xsd:element ref="ChildInformation" minOccurs="0"
	maxOccurs="unbounded"/>
				</xsd:sequence>
			</xsd:complexType>
		</xsd:element>
		<xsd:element name="ChildInformation">
			<xsd:complexType>
				<xsd:sequence>
					<xsd:element name="name" type="name"/>
					<xsd:element name="dateOfBirth" type="xsd:date"/>
				</xsd:sequence>
			</xsd:complexType>
		</xsd:element>
		<xsd:simpleType name="ListOfDays">
			<xsd:list itemType="Day"/>
		</xsd:simpleType>
		<xsd:simpleType name="Day">
			<xsd:restriction base="xsd:string">
				<xsd:enumeration value="monday"/>
				<xsd:enumeration value="tuesday"/>
				<xsd:enumeration value="wednesday"/>
				<xsd:enumeration value="thursday"/>
				<xsd:enumeration value="friday"/>
```

```
                        <xsd:enumeration value="saturday"/>
                        <xsd:enumeration value="sunday"/>
                    </xsd:restriction>
                </xsd:simpleType>
                <xsd:element name="namesOfMemberNations">
                    <xsd:simpleType>
                        <xsd:list itemType="xsd:string"/>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="fileIdentifier">
                    <xsd:complexType>
                        <xsd:choice>
                            <xsd:element name="serialNumber" type="xsd:int"/>
                            <xsd:element name="relativeName" type="xsd:string"/>
                            <xsd:element ref="unidentified"/>
                        </xsd:choice>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="unidentified">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:restriction base="xsd:anyType"/>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:schema>
```

C.2 The corresponding ASN.1 definitions

The following is the corresponding ASN.1 specification and validates the same XML documents as the XSD Schema:

```
EXAMPLES{joint-iso-itu-t asn1(1) examples(999) xml-defined-types(3)}
DEFINITIONS AUTOMATIC TAGS

XER INSTRUCTIONS ::=
BEGIN

IMPORTS String, Decimal, Int, Date, AnyType
        FROM XSD
        {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(1)};

EXAMPLES ::=     SEQUENCE {
      personnelRecord      PersonnelRecord,
      number               Decimal,
      daysOfTheWeek        ListOfDays,
      namesOfMemberNations NamesOfMemberNations,
      fileIdentifier-list  [UNTAGGED]
             SEQUENCE (SIZE(1..MAX)) OF fileidentifier FileIdentifier }

PersonnelRecord  ::=   [NAME AS UNCAPITALIZED] SEQUENCE {
      name             Name,
      title            XSD.String,
      number           INTEGER,
      dateOfHire       Date,
      nameOfSpouse     NameOfSpouse,
      children         Children }

NameOfSpouse ::= [NAME AS UNCAPITALIZED] Name

Name ::= [NAME AS UNCAPITALIZED] SEQUENCE {
      givenName        XSD.String,
      initial          XSD.String,
      familyName       XSD.String }

Children ::= [NAME AS UNCAPITALIZED] SEQUENCE {
       childInformation-list [UNTAGGED]
            SEQUENCE OF ChildInformation }

ChildInformation ::=   SEQUENCE {
      name             Name,
      dateOfBirth      Date }

ListOfDays ::=   [LIST] SEQUENCE OF Day
```

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

NamesOfMemberNations ::= [NAME AS UNCAPITALIZED] [LIST] SEQUENCE OF XSD.String

FileIdentifier ::= [NAME AS UNCAPITALIZED] SEQUENCE {
      choice [UNTAGGED] CHOICE {
            serialNumber      Int,
            relativeName      XSD.String,
            unidentified      UNIDENTIFIED   } }

UNIDENTIFIED ::=  [NAME AS LOWERCASED] XSD.AnyType

ENCODING-CONTROL XER
      GLOBAL-DEFAULTS MODIFIED-ENCODINGS

END
```

C.3 Further examples

In this subclause, all the partial examples (the examples that do not contain the **schema** element) assume that the XML elements representing the XSD syntax are in the scope of a **default namespace** declaration whose **namespace name** is the **target namespace** of the schema.

C.3.1 Schema documents with **import** and **include** element information items

The following XSD Schema is composed of two namespaces that are composed from four schema files:

```
<!-- file "http://example.com/xyz/schema.xsd" -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xyz="http://example.com/xyz"
        targetNamespace="http://example.com/xyz">

        <xsd:element name="xyz-elem" type="xsd:string"/>

        <xsd:complexType name="Xyz-type">
                <xsd:attribute name="xyz-attr" type="xsd:boolean"/>
        </xsd:complexType>
</xsd:schema>

<!-- file "http://example.com/abc/main.xsd" -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:abc="http://example.com/abc"
        targetNamespace="http://example.com/xyz">

        <xsd:include schemaLocation="http://example.com/abc/sub1.xsd"/>

        <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
                schemaLocation="http://example.com/xyz/schema.xsd"/>

        <xsd:redefine schemaLocation="http://example.com/abc/sub2.xsd">
                <xsd:attribute name="sub2-attr" type="xsd:token"/>
        </xsd:redefine>

        <xsd:element name="abc-elem" type="Xyz-type"/>
</xsd:schema>

<!-- file "http://example.com/abc/sub1.xsd" -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:abc="http://example.com/abc"
            targetNamespace="http://example.com/xyz">

        <xsd:element name="sub1-elem" type="xsd:string"/>
</xsd:schema>

<!-- file "http://example.com/abc/sub2.xsd" -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="sub2-elem " type="xsd:string"/>
        <xsd:attribute name="sub2-attr" type="xsd:string"/>
</xsd:schema>
```

Those four schema documents are mapped to the two following ASN.1 modules:

```
      XYZ   -- The module reference is not standardized
      DEFINITIONS AUTOMATIC TAGS ::=
```

```
BEGIN

Xyz-elem ::= [NAME AS UNCAPITALIZED] XSD.String
Xyz-type ::= SEQUENCE {
      xyz-attr [ATTRIBUTE] BOOLEAN OPTIONAL }

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

ABC   -- The module reference is not standardized
DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
IMPORTS
Xyz-type FROM XYZ
Token, String FROM XSD;

Abc-elem ::= [NAME AS UNCAPITALIZED] Xyz-type
Sub1-elem ::= [NAME AS UNCAPITALIZED] XSD.String
Sub2-elem ::= [NAME AS UNCAPITALIZED] XSD.String
Sub2-attr ::= [NAME AS UNCAPITALIZED] [ATTRIBUTE] XSD.Token

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

C.3.2 Mapping simple type definitions

C.3.2.1 simple type definition derived by restriction

For a complete set of examples of simple type restrictions, see the examples of facets in C.3.3.

C.3.2.2 simple type definition derived by list

```
<xsd:simpleType name="Int-list">
      <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="Int-10-to-100-list">
      <xsd:list>
            <xsd:simpleType>
                  <xsd:restriction base="xsd:integer">
                        <xsd:minInclusive value="10"/>
                        <xsd:minInclusive value="100"/>
                  </xsd:restriction>
            </xsd:simpleType>
      </xsd:list
</xsd:simpleType>
```

These simple type definitions are mapped to the following ASN.1 type assignments:

```
Int-list ::= [LIST] SEQUENCE OF INTEGER

Int-10-to-100-list ::= [LIST] SEQUENCE OF INTEGER (10..100)
```

C.3.2.3 simple type definition derived by union

```
<xsd:simpleType name="Int-or-boolean">
      <xsd:union itemType="xsd:integer xsd:boolean"/>
</xsd:simpleType>

<xsd:simpleType name="Time-or-int-or-boolean--or-dateRestriction">
      <xsd:union itemType=" xsd:time Int-or-boolean">
            <xsd:simpleType>
                  <xsd:restriction base="xsd:date">
                        <xsd:minInclusive value="2003-01-01"/>
                  </xsd:restriction>
            </xsd:simpleType>
      </xsd:union>
</xsd:simpleType>
```

These simple type definitions are mapped to the following ASN.1 type assignments:

```
Int-or-boolean ::= [USE-UNION] CHOICE {
      integer [NAMESPACE "http://www.w3.org/2001/XMLSchema"] INTEGER,
      boolean [NAMESPACE "http://www.w3.org/2001/XMLSchema"] BOOLEAN }

Time-or-int-or-boolean-or-dateRestriction ::= [USE-UNION] CHOICE {
      time [NAMESPACE "http://www.w3.org/2001/XMLSchema"] XSD.Time,
      integer [NAMESPACE "http://www.w3.org/2001/XMLSchema"] INTEGER,
      boolean [NAMESPACE "http://www.w3.org/2001/XMLSchema"] BOOLEAN,
      alt [NAME AS ""] XSD.Date (CONSTRAINED BY
            { /* minInclusive="2003-01-01" */ }) }
```

C.3.2.4 Mapping type derivation hierarchies for simple type definitions

```
<xsd:simpleType name="Int-10-to-50">
      <xsd:restriction base="xsd:integer">
            <xsd:minExclusive value="10"/>
            <xsd:maxExclusive value="50"/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Ten-multiples">
      <xsd:restriction base="Int-10-to-50">
            <xsd:enumeration value="20"/>
            <xsd:enumeration value="30"/>
            <xsd:enumeration value="40"/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Twenty-multiples">
      <xsd:restriction base="Ten-multiples">
            <xsd:pattern value=".*[02468]0|0"/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="Stock-level">
      <xsd:extension base="Int-10-to-50">
            <xsd:attribute name="procurement" type="Int-10-to-50"/>
      </xsd:extension>
</xsd:complexType>
```

These simple type definitions are mapped to the following ASN.1 type assignments:

```
Int-10-to-50 ::= INTEGER (10<..<50)

Ten-multiples ::= ENUMERATED {int20(20), int30(30), int40(40)}

Twenty-multiples ::= ENUMERATED {int20(20), int40(40)}

Stock-level ::= SEQUENCE {
      procurement [ATTRIBUTE] Int-10-to-50 OPTIONAL,
      base        Int-10-to-50-derivations }

Ten-multiples-derivations ::= [USE-TYPE] CHOICE {
      ten-multiples    [NAME AS CAPITALIZED] Ten-multiples,
      twenty-multiples [NAME AS CAPITALIZED] Twenty-multiples }

Int-10-to-50-derivations ::= [USE-TYPE] CHOICE {
      int-10-to-50     [NAME AS CAPITALIZED] Int-10-to-50,
      ten-multiples    [NAME AS CAPITALIZED] Ten-multiples,
      twenty-multiples [NAME AS CAPITALIZED] Twenty-multiples,
      stock-level      [NAME AS CAPITALIZED] Stock-level }
```

C.3.3 Mapping facets

C.3.3.1 length, minLength, and maxLength

```
<xsd:simpleType name="String-10">
      <xsd:restriction base="xsd:string">
            <xsd:length value="10"/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="String-5-to-10">
      <xsd:restriction base="xsd:string">
            <xsd:minLength value="5"/>
            <xsd:maxLength value="10"/>
```

```
        </xsd:restriction>
</xsd:simpleType>
```

These two **simple type definition**s are mapped to the following ASN.1 type assignments:

```
String-10 ::= XSD.String (SIZE(10))

String-5-to-10 ::= XSD.String (SIZE(5..10))
```

C.3.3.2 pattern

```
<xsd:simpleType name="My-filename">
        <xsd:restriction base="xsd:string">
                <xsd:pattern value="[&#00;-&#FF;]*"/>
                <xsd:pattern value="/?([^/]*/)*[^/]*/*" />
        </xsd:restriction>
</xsd:simpleType>
```

This **simple type definition** is mapped to the following ASN.1 type assignment:

```
My-filename ::= XSD.String
      (CONSTRAINED BY
            {/* XML representation of the XSD pattern
                "&#00;-&#FF;" | "/?([^/]*&#x2F;)*[^/]*&#x2F;*" */})
```

C.3.3.3 whiteSpace

```
<xsd:simpleType name="My-String">
        <xsd:restriction base="xsd:string">
                <xsd:whitespace value="preserve"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="My-NormalizedString">
        <xsd:restriction base="xsd:string">
                <xsd:whitespace value="replace"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="My-TokenString">
        <xsd:restriction base="xsd:string">
                <xsd:whitespace value="collapse"/>
        </xsd:restriction>
</xsd:simpleType>
```

These **simple type definition**s are mapped to the following ASN.1 type assignments:

```
My-String ::= XSD.String

My-NormalizedString ::= [WHITESPACE REPLACE] XSD.String
            (FROM ({0, 0, 0, 32} .. {0, 16, 255, 255}))

My-TokenString ::= [WHITESPACE REPLACE] XSD.String
            (FROM ({0, 0, 0, 32} .. {0, 16, 255, 255}))
            (PATTERN "([^ ]([^ ]| [^ ])*)?")
```

C.3.3.4 minInclusive, minExclusive, maxInclusive, and maxExclusive

```
<xsd:simpleType name="Int-10-to-100">
        <xsd:restriction base="xsd:integer">
                <xsd:minExclusive value="10"/>
                <xsd:maxInclusive value="100"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Pi-approximation">
        <xsd:restriction base="xsd:double">
                <xsd:minExclusive value="3.14159"/>
                <xsd:maxExclusive value="3.1416"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Morning">
        <xsd:restriction base="xsd:time">
                <xsd:minInclusive value="00:00:00"/>
                <xsd:maxExclusive value="12:00:00"/>
        </xsd:restriction>
</xsd:simpleType>
```

These **simple type definition**s are mapped to the following ASN.1 type assignments:

```
Int-10-to-100 ::= INTEGER (10<..100)

Pi-approximation ::= XSD.Double (3.14159<..<3.1416)

Morning ::= XSD.Time (CONSTRAINED BY
      {/* minInclusive="00:00:00" maxExclusive="12:00:00" */ })
```

C.3.3.5 totalDigits and fractionDigits

```
<xsd:simpleType name="RefundableExpenses">
      <xsd:restriction base="xsd:decimal">
            <xsd:totalDigits="5"/>
            <xsd:fractionDigits value="2"/>
      </xsd:restriction>
</xsd:simpleType>
```

This **simple type definition** is mapped to the following ASN.1 type assignment:

```
RefundableExpenses ::= XSD.Decimal (CONSTRAINED BY
      {/* totalDigits="5" fractionDigits="2" */ })
```

C.3.3.6 enumeration

```
<xsd:simpleType name="FarmAnimals">
      <xsd:restriction base="xsd:normalizedString">
            <xsd:enumeration value="Horse"/>
            <xsd:enumeration value="Bull"/>
            <xsd:enumeration value="Cow"/>
            <xsd:enumeration value="Pig"/>
            <xsd:enumeration value="Duck"/>
            <xsd:enumeration value="Goose"/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PrimeNumbersBelow30">
      <xsd:restriction base="xsd:integer">
            <xsd:enumeration value="2"/>
            <xsd:enumeration value="3"/>
            <xsd:enumeration value="5"/>
            <xsd:enumeration value="7"/>
            <xsd:enumeration value="11"/>
            <xsd:enumeration value="13"/>
            <xsd:enumeration value="17"/>
            <xsd:enumeration value="19"/>
            <xsd:enumeration value="23"/>
            <xsd:enumeration value="29"/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="X680-release">
      <xsd:restriction base="xsd:gYearMonth">
            <xsd:enumeration value="2002-07"/>
            <xsd:enumeration value="1997-12"/>
            <xsd:enumeration value="1994-07"/>
      </xsd:restriction>
</xsd:simpleType>
```

These **simple type definition**s are mapped to the following ASN.1 type assignments:

```
FarmAnimals ::= ENUMERATED {horse, bull, cow, pig, duck, goose}

PrimeNumbersBelow30 ::= [USE-NUMBER] ENUMERATED {int2(2), int3(3), int5(5),
      int7(7), int11(11), int13(13), int17(17), int19(19), int23(23), int29(29)}

X680-release ::= XSD.GYearMonth ("2002-07" | "1997-12" | "1994-07")
```

The following encoding instruction is included in the XER encoding control section:

```
TEXT FarmAnimals:ALL AS CAPITALIZED
```

C.3.3.7 enumeration in conjunction with other facets

The following examples are based on the inheritance of facets using the restriction of some of the types defined in C.3.3.6.

```
<xsd:simpleType name="FarmAnimals-subset">
```

```
        <xsd:restriction base="FarmAnimals">
                <xsd:minLength value="4"/>
                <xsd:pattern value="[^oe]*"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PrimeNumbersBelow30-subset">
        <xsd:restriction base="PrimeNumbersBelow30">
                <xsd:minExclusive value="5"/>
                <xsd:pattern value=".*[23].*"/>
        </xsd:restriction>
</xsd:simpleType>
```

These **simple type definition**s are mapped to the following ASN.1 type assignments:

```
/* Horse and Goose do not satisfy the pattern facet

   Cow and Pig do not satisfy the minLength facet */

FarmAnimals-subset ::= ENUMERATED {bull, duck}

/* 2, 3 and 5 do not satisfy the minExclusive facet

   2, 5, 7, 11, 17 and 19 do not satisfy  the pattern facet */

PrimeNumbersBelow30-subset ::= [USE-NUMBER] ENUMERATED {int13(13), int23(23),
                                                        int29(29)}
```

The following encoding instruction is included in the XER encoding control section:

```
TEXT FarmAnimals-subset:ALL AS CAPITALIZED
```

C.3.4 Mapping element declarations

C.3.4.1 element declarations whose type definition is a user-defined top-level simple type definition or complex type definition

```
<xsd:element name="Forename" type="xsd:token"/>

<xsd:element name="File" type="My-filename"/>

<xsd:element name="Value" type="Int-10-to-50"/>
```

These **element declaration**s are mapped to the following ASN.1 type assignments:

```
Forename ::= XSD.Token

File ::= My-filename

Value ::= Int-10-to-50-derivations
```

NOTE – The type "**My-filename**" and its mapping to ASN.1 is defined in C.3.3.2; the type "**Int-10-to-50**" and its mapping to ASN.1 is defined in C.3.2.4.

C.3.4.2 element declarations whose type definition is an anonymous simple type definition or complex type definition

```
<xsd:element name="maxOccurs">
        <xsd:simpleType>
                <xsd:union memberTypes="xsd:nonNegativeInteger">
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:token">
                                        <xsd:enumeration name="unbounded"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:union>
        </xsd:simpleType>
</xsd:element>

<xsd:element name="address">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="line-1" type="xsd:token"/>
                        <xsd:element name="line-2" type="xsd:token"/>
                        <xsd:element name="city" type="xsd:token"/>
                        <xsd:element name="state" type="xsd:token" minOccurs="0"/>
                        <xsd:element name="zip" type="xsd:token"/>
                </xsd:sequence>
                <xsd:attribute name="country" type="xsd:token"/>
        </xsd:complexType>
</xsd:element>
```

These **element declaration**s are mapped to the following ASN.1 type assignments:

```
MaxOccurs ::= [NAME AS UNCAPITALIZED] [USE-UNION] CHOICE {
      nonNegativeInteger [NAMESPACE AS "http://www.w3.org/2001/XMLSchema"]
                                          XSD.NonNegativeInteger,
      alt                [NAME AS ""]    ENUMERATED {unbounded} }

Address ::= [NAME AS UNCAPITALIZED] SEQUENCE {
      Country      [ATTRIBUTE] XSD.Token OPTIONAL,
      line-1       XSD.Token,
      line-2       XSD.Token,
      city         XSD.Token,
      state        XSD.Token OPTIONAL,
      zip          XSD.Token }
```

C.3.4.3 element declarations which are the head of an element substitution group

```
<xsd:element name="Tic" type="xsd:integer"  abstract="true"/>

<xsd:element name="Tac" type="xsd:byte" substitutionGroup="Tic"/>

<xsd:element name="Toe" substitutionGroup="Tic"/>

<xsd:element name="Foo" type="xsd:date"/>

<xsd:element name="Bar" substitutionGroup="Foo"/>
```

These **element declaration**s are mapped to:

```
Tac ::=  INTEGER (-128..127)

Toe ::= INTEGER

Tic-group ::= [UNTAGGED] CHOICE {
      tac [NAME AS CAPITALIZED] Tac,
      toe [NAME AS CAPITALIZED] Toe }

Foo ::= XSD.Date

Bar ::= XSD.Date

Foo-group ::= [UNTAGGED] CHOICE {
      foo [NAME AS CAPITALIZED] Foo,
      bar [NAME AS CAPITALIZED] Bar }
```

C.3.4.4 element declarations with a value constraint that is a default value

C.3.4.4.1 The following is an **element declaration** with an anonymous **simple type definition**, not used as the **base type definition** of any type.

```
<xsd:element name="Telephone" type="xsd:token" default="undefined"/>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
Telephone ::= [DEFAULT-FOR-EMPTY "undefined"] XSD.Token
```

C.3.4.4.2 The following is an **element declaration** with an anonymous **complex type definition** with simple content, not used as the **base type definition** of any type.

```
<xsd:element name="InternationalTelephone" default="undefined">
      <xsd:simpleContent>
            <xsd:extension base="xsd:token">
                  <xsd:attribute name="country-code" type="xsd:integer"/>
            </xsd:extension>
      </xsd:simpleContent>
</xsd:element>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
InternationalTelephone ::= [DEFAULT-FOR-EMPTY "undefined"] SEQUENCE {
      country-code  [ATTRIBUTE] INTEGER OPTIONAL,
      base          [UNTAGGED]  XSD.Token }
```

C.3.4.4.3 The following is an **element declaration** with an anonymous **complex type definition.** The **complex type definition** has complex content that is **mixed** and emptiable, and is not used as the **base type definition** of any type.

```
<xsd:element name="Description" default="absent" mixed="true">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="bold" type="xsd:string"/>
            <xsd:element name="italic" type="xsd:string"/>
      </xsd:choice>
</xsd:element>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
Description ::= [EMBED-VALUES] [DEFAULT-FOR-EMPTY "absent"] SEQUENCE {
      embed-values  SEQUENCE OF XSD.String,
      choice-list   [UNTAGGED] SEQUENCE OF [UNTAGGED] CHOICE {
            bold        XSD.String,
            italic      XSD.String } } (CONSTRAINED BY
      {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */})
```

C.3.4.4.4 The **type definition** of the **element declaration** in the following example is used as the **base type definition** of another type.

This example uses the XSD and ASN.1 types of the example in C.3.2.4.

```
<xsd:element name="Quantity" type="Int-10-to-50" default="20"/>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
Quantity ::= Int-10-to-50-deriv-default-20
```

If no ASN.1 type corresponding to `Int-10-to-50`, with a default value of "20" has already been generated, the following type is also generated:

```
Int-10-to-50-deriv-default-20 ::= [USE-TYPE] CHOICE {
      int-10-to-50     [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY 20]
                                    Int-10-to-50,
      ten-multiples    [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY int20]
                                    Ten-multiples,
      twenty-multiples [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY int20]
                                    Twenty-multiples,
      stock-level      [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY 20]
                                    Stock-level }
```

C.3.4.5 element declaration with a value constraint that is a fixed value

C.3.4.5.1 The following is an **element declaration** with an anonymous **simple type definition**, which is not used as the **base type definition** of any type.

```
<xsd:element name="UnknownTelephone" type="xsd:token" fixed="undefined"/>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
UnknownTelephone ::= [DEFAULT-FOR-EMPTY "undefined"] XSD.Token ("undefined")
```

C.3.4.5.2 The following is an **element declaration** with an anonymous **complex type definition.** The **complex type definition** has simple content and is not used as the **base type definition** of any type.

```
<xsd:element name="UnknownInternationalTelephone" fixed="undefined">
      <xsd:simpleContent>
            <xsd:extension base="xsd:token">
                  <xsd:attribute name="country-code" type="xsd:integer"/>
            </xsd:extension>
      </xsd:simpleContent>
</xsd:element>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
UnknownInternationalTelephone ::= [DEFAULT-FOR-EMPTY "undefined"] SEQUENCE {
      country-code    [ATTRIBUTE] INTEGER OPTIONAL,
      base            [UNTAGGED] XSD.Token }
            (WITH COMPONENTS {..., base ("undefined")})
```

C.3.4.5.3 The following is an **element declaration** with an anonymous **complex type definition.** The **complex type definition** has complex content that is **mixed** and emptiable, and is not used as the **base type definition** of any type.

```
<xsd:element name="UnknownDescription" fixed="absent" mixed="true">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="bold" type="xsd:string"/>
            <xsd:element name="italic" type="xsd:string"/>
      </xsd:choice>
</xsd:element>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
UnknownDescription ::= [EMBED-VALUES] [DEFAULT-FOR-EMPTY "absent"] SEQUENCE {
      embed-values      SEQUENCE OF XSD.String,
      choice-list       [UNTAGGED] SEQUENCE OF [UNTAGGED] CHOICE {
            bold              XSD.String,
            italic            XSD.String } }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */})
      (WITH COMPONENTS {embed-values ({"absent"}),
                        choice-list (SIZE(0))})
```

C.3.4.5.4 The **type definition** of the following **element declaration** is a **simple type definition** used as the **base type definition** of another type.

This example uses the XSD and ASN.1 types of the example C.3.2.4.

```
<xsd:element name="Quantity" type="Int-10-to-50" fixed="20"/>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
Quantity ::= Int-10-to-50-deriv-fixed-20
```

If no ASN.1 type corresponding to `Int-10-to-50` with a fixed value of "20" has already been generated, the following type is also generated:

```
Int-10-to-50-deriv-fixed-20 ::= [USE-TYPE] CHOICE {
      int-10-to-50      [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY 20]
                              Int-10-to-50,
      ten-multiples     [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY int20]
                              Ten-multiples,
      twenty-multiples  [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY int20]
                              Twenty-multiples,
      stock-level       [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY 20]
                              Stock-level }
(WITH COMPONENTS {
      int-10-to-50 (20),
      ten-multiples (int20),
      twenty-multiples (int20),
      stock-level (WITH COMPONENTS {..., base (20)}) })
```

C.3.4.6 element declarations that are nillable

C.3.4.6.1 The following example shows a **element declaration** that is **nillable** and whose **type definition** is an XSD built-in datatype.

```
<xsd:element name="Nillable-1" type="xsd:string" nillable="true"/>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
Nillable-1 ::= [USE-NIL] SEQUENCE {
      content XSD.String OPTIONAL }
```

C.3.4.6.2 The following example shows an **element declaration** that is **nillable** and whose **type definition** is an anonymous **complex type definition**.

```
<xsd:element name="Nillable-2" nillable="true">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="a" type="xsd:string"/>
                  <xsd:element name="b" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="b" type="xsd:boolean"/>
        </xsd:complexType>
</xsd:element>
```

This **element declaration** is mapped to the following ASN.1 type assignment:

```
Nillable-2 ::= [USE-NIL] SEQUENCE {
      b           [ATTRIBUTE] BOOLEAN OPTIONAL,
      content     SEQUENCE {
            a           XSD.String,
            b           XSD.String } OPTIONAL }
```

C.3.4.6.3 The following example shows an **element declaration** that is **nillable**, and whose **type definition** is a top-level **complex type definition**.

```
<xsd:complexType name="Foo">
      <xsd:sequence>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="b" type="xsd:boolean"/>
</xsd:complexType>

<xsd:element name="Nillable-3" type="Foo" nillable="true"/>
```

These schema components are mapped to the following ASN.1 type assignments:

```
Foo ::= SEQUENCE {
      b     [ATTRIBUTE] BOOLEAN OPTIONAL,
      a     XSD.String,
      b-1   [NAME AS "b"] XSD.String }

Foo-nillable ::= [USE-NIL] SEQUENCE {
      b           [ATTRIBUTE] BOOLEAN OPTIONAL,
      content     SEQUENCE {
            a           XSD.String,
            b           XSD.String } OPTIONAL }

Nillable-3 ::= Foo-nillable
```

C.3.4.6.4 The following example shows an **element declaration** that is **nillable**, whose **type definition** is a top-level **complex type definition**, and which is used as the **base type definition** of another **complex type definition**.

The following schema components are defined in addition to the schema components of C.3.4.6.3:

```
<xsd:complexType name="Bar">
      <xsd:complexContent>
            <xsd:extension base="Foo">
                  <xsd:sequence>
                        <xsd:element name="z" type="xsd:string"/>
                  </xsd:sequence>
                  <xsd:attribute name="c" type="xsd:boolean"/>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Nillable-4" type="Foo" nillable="true"/>
```

In addition to the type Foo of C.3.4.6.3, the following ASN.1 types are generated:

```
Bar ::= SEQUENCE {
      b     [ATTRIBUTE] BOOLEAN OPTIONAL,
      c     [ATTRIBUTE] BOOLEAN OPTIONAL,
      a     XSD.String,
      b-1   [NAME AS "b"] XSD.String,
      z     XSD.String }

Foo-derivations ::= [USE-TYPE] CHOICE {
      foo   [NAME AS CAPITALIZED] Foo,
      bar   [NAME AS CAPITALIZED] Bar }

Foo-nillable ::= [USE-NIL] SEQUENCE {
      b           [ATTRIBUTE] BOOLEAN OPTIONAL,
      content     SEQUENCE {
            a           XSD.String,
            b           XSD.String } OPTIONAL }

Bar-nillable ::= [USE-NIL] SEQUENCE {
      b           [ATTRIBUTE] BOOLEAN OPTIONAL,
      c           [ATTRIBUTE] BOOLEAN OPTIONAL,
      content     SEQUENCE {
            a           XSD.String,
```

```
            b           XSD.String,
            z           XSD.String } OPTIONAL }

Foo-deriv-nillable ::= [USE-TYPE] CHOICE {
      foo         [NAME AS CAPITALIZED] Foo-nillable,
      bar         [NAME AS CAPITALIZED] Bar-nillable }

Nillable-4 ::= Foo-deriv-nillable
```

C.3.5 Mapping attribute uses and attribute declarations

C.3.5.1 The following is an example of a top-level **attribute declaration** whose **type definition** is a user-defined top-level **simple type definition**.

```
<xsd:attribute name="name" type="NCName"/>
```

This **attribute declaration** is mapped to the following ASN.1 type assignment:

```
Name ::= [NAME AS UNCAPITALIZED] [ATTRIBUTE] XSD.NCName
```

C.3.5.2 The following is an example of a top-level **attribute declaration** whose **type definition** is an anonymous **simple type definition**.

```
<xsd:attribute name="form">
      <xsd:simpleType>
            <xsd:restriction base="xsd:token">
                  <xsd:enumeration value="qualified"/>
                  <xsd:enumeration value="unqualified"/>
            </xsd:restriction>
      </xsd:simpleType>
</xsd:attribute>
```

This **attribute declaration** is mapped to the following ASN.1 type assignment:

```
Form ::= [NAME AS UNCAPITALIZED] [ATTRIBUTE] ENUMERATED {qualified, unqualified}
```

C.3.5.3 The following example is an **attribute use** with a **value constraint** that is a **default** value.

The **attribute declaration** whose **name** is "form" and that is referenced in this example is defined in C.3.5.2.

```
<xsd:complexType name="element">
      <xsd:attribute name="name" type="xsd:NCName" default="NAME"/>
      <xsd:attribute ref="form" default="qualified"/>
</xsd:complexType>
```

This **complex type definition** is mapped to the following ASN.1 type assignment:

```
Element ::= [NAME AS UNCAPITALIZED] SEQUENCE {
      name [ATTRIBUTE] XSD.NCName DEFAULT "NAME",
      form [ATTRIBUTE] Form DEFAULT qualified }
```

C.3.5.4 This example shows a top-level **attribute declaration** with a **value constraint** that is a **default** value and an **attribute use** with this **attribute declaration**.

```
<xsd:attribute name="minOccurs" type="xsd:nonNegativeInteger" default="1"/>

<xsd:attribute name="maxOccurs" default="1"/>
      <xsd:simpleType>
            <xsd:union memberTypes="xsd:nonNegativeInteger" >
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:NMTOKEN">
                              <xsd:enumeration value="unbounded"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:union>
      </xsd:simpleType>
</xsd:attribute>

<xsd:complexType name="Particle">
      <xsd:sequence>
            <xsd:element name="particle"/>
      </xsd:sequence>
      <xsd:attribute ref="minOccurs"/>
      <xsd:attribute ref="maxOccurs" default="unbounded"/>
</xsd:complexType>
```

These schema components are mapped to the following ASN.1 type assignments:

```
MinOccurs ::= [ATTRIBUTE] [NAME AS UNCAPITALIZED] XSD.NonNegativeInteger

MaxOccurs ::= [ATTRIBUTE] [NAME AS UNCAPITALIZED] [USE-UNION] CHOICE {
      nonNegativeInteger     [NAMESPACE AS "http://www.w3.org/2001/XMLSchema"]
                                    XSD.NonNegativeInteger,
      alt                    [NAME AS ""] ENUMERATED {unbounded} }

Particle ::= SEQUENCE {
      minOccurs         [ATTRIBUTE] MinOccurs DEFAULT 1,
      maxOccurs         [ATTRIBUTE] MaxOccurs DEFAULT alt : unbounded,
      particle          XSD.AnyType }
```

C.3.5.5 This example shows an **attribute use** whose **attribute declaration** has a **target namespace** that is not **absent**.

```
<xsd:complexType name="Ack">
      <xsd:attribute name="number" type="xsd:integer" form="qualified" />
</xsd:complexType>
```

This **complex type definition** is mapped to the following ASN.1 type assignment:

```
Ack ::= SEQUENCE {
      number [NAMESPACE AS "http://targetnamespaceForExample"] [ATTRIBUTE]
                        INTEGER OPTIONAL }
```

C.3.6 Mapping model group definitions

C.3.6.1 The following is a **model group definition** whose **model group** has a **compositor** of **sequence.**

```
<xsd:group name="mySequence">
      <xsd:sequence>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:sequence>
</xsd:group>
```

This **model group definition** is mapped to the following ASN.1 type assignment:

```
MySequence ::= [UNTAGGED] SEQUENCE {
      a XSD.String,
      b BOOLEAN }
```

C.3.6.2 The following is a **model group definition** whose **model group** has a **compositor** of **all:**

```
<xsd:group name="myAll ">
      <xsd:all>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:all>
</xsd: group>
```

This **model group definition** is not mapped to ASN.1. See C.3.8.3.1 for an example of the mapping of a **complex type definition** where the **model group** of this **model group definition** occurs as the topmost **model group**.

C.3.6.3 The following is a **model group definition** whose **model group** has a **compositor** of **choice**.

```
<xsd:group name="myChoice">
      <xsd:choice>
            <xsd:element name="am" type="xsd:string"/>
            <xsd:element name="bm" type="xsd:boolean"/>
      </xsd:choice>
</xsd:group>
```

This **model group definition** is mapped to the following ASN.1 type assignment:

```
MyChoice ::= [UNTAGGED] CHOICE {
      am XSD.String,
      bm BOOLEAN }
```

C.3.7 Mapping particles

The **model group definition** of C.3.6.3, and its corresponding ASN.1 type are used in some of the **particle** examples.

C.3.7.1 The following example shows **particle**s of a **model group** with a **compositor** of **sequence**.

```
<xsd:complexType name="ElementSequence">
	<xsd:sequence>
		<xsd:element name="elem1" type="xsd:boolean"/>
		<xsd:element name="elem2" type="xsd:boolean" minOccurs="0"/>
		<xsd:element name="elem3" type="xsd:boolean" minOccurs="2" maxOccurs="5"/>
		<xsd:element name="elem4" type="xsd:boolean" minOccurs="0" maxOccurs="unbounded"/>
		<xsd:element name="elem5" type="xsd:boolean" minOccurs="5" maxOccurs="unbounded"/>
	<xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ModelGroupSequence">
	<xsd:sequence>
		<xsd:group ref="myChoice"/>
		<xsd:choice>
			<xsd:element name="a" type="xsd:string"/>
			<xsd:element name="b" type="xsd:string"/>
		</xsd:choice>
		<xsd:sequence>
			<xsd:element name="c" type="xsd:string"/>
			<xsd:element name="d" type="xsd:string"/>
		</xsd:sequence>
		<xsd:choice minOccurs="3" maxOccurs="12">
			<xsd:element name="e" type="xsd:string"/>
			<xsd:element name="f" type="xsd:string"/>
		</xsd:choice>

	</xsd:sequence>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
ElementSequence ::= SEQUENCE {
      elem1             BOOLEAN,
      elem2             BOOLEAN OPTIONAL,
      elem3-list        [UNTAGGED] SEQUENCE (SIZE(2..5)) OF elem3 BOOLEAN,
      elem4-list        [UNTAGGED] SEQUENCE OF elem4 BOOLEAN,
      elem5-list        [UNTAGGED] SEQUENCE (SIZE(1..MAX)) OF elem5 BOOLEAN }

ModelGroupSequence ::= SEQUENCE {
      myChoice          MyChoice,
      choice            [UNTAGGED] CHOICE {
            a                 XSD.String,
            b                 XSD.String },
      sequence          [UNTAGGED] SEQUENCE {
            c                 XSD.String,
            d                 XSD.String },
      choice-list       [UNTAGGED] SEQUENCE (SIZE(3..12)) OF [UNTAGGED] CHOICE {
            e                 XSD.String,
            f                 XSD.String } }
```

C.3.7.2 The following example shows **particle**s of a **model group** with a **compositor** of **all**.

```
<xsd:complexType name="ElementAll">
	<xsd:all>
		<xsd:element name="elem1" type="xsd:boolean"/>
		<xsd:element name="elem2" type="xsd:boolean" minOccurs="0"/>
	</xsd:all>
</xsd:complexType>
```

This **complex type definition** is mapped to the following ASN.1 type assignments:

```
ElementAll ::= [USE-ORDER] SEQUENCE {
      order SEQUENCE OF ENUMERATED {elem1, elem2},
      elem1 XSD.String,
      elem2 XSD.String OPTIONAL }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 35 */})
```

C.3.7.3 The following example shows **particle**s of a **model group** with a **compositor** of **choice**.

```
<xsd:complexType name="ElementSequence">
	<xsd:choice>
		<xsd:element name="elem1" type="xsd:boolean"/>
		<xsd:element name="elem2" type="xsd:boolean" minOccurs="0"/>
```

```
                <xsd:element name="elem3" type="xsd:boolean" minOccurs="2" maxOccurs="5"/>
                <xsd:element name="elem4" type="xsd:boolean" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="elem5" type="xsd:boolean" minOccurs="5" maxOccurs="unbounded"/>
        <xsd:choice>
</xsd:complexType>

<xsd:complexType name="ModelGroupChoice">
        <xsd:choice>
                <xsd:group ref="myChoice"/>
                <xsd:choice>
                        <xsd:element name="a" type="xsd:string"/>
                        <xsd:element name="b" type="xsd:string"/>
                </xsd:choice>
                <xsd:sequence>
                        <xsd:element name="c" type="xsd:string"/>
                        <xsd:element name="d" type="xsd:string"/>
                </xsd:sequence>
                <xsd:choice minOccurs="3" maxOccurs="12">
                        <xsd:element name="e" type="xsd:string"/>
                        <xsd:element name="f" type="xsd:string"/>
                </xsd:choice>

        </xsd:choice>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
ElementSequence ::= SEQUENCE {
      choice [UNTAGGED] CHOICE {
            elem1       BOOLEAN,
            elem2-list [UNTAGGED] SEQUENCE (SIZE(0..1)) OF elem2 BOOLEAN,
            elem3-list [UNTAGGED] SEQUENCE (SIZE(2..5)) OF elem3 BOOLEAN,
            elem4-list [UNTAGGED] SEQUENCE OF elem4 BOOLEAN,
            elem5-list [UNTAGGED] SEQUENCE (SIZE(5..MAX)) OF elem5 BOOLEAN } }

ModelGroupChoice ::= SEQUENCE {
      choice [UNTAGGED] CHOICE {
            myChoice    MyChoice,
            choice      [UNTAGGED] CHOICE {
                  a           XSD.String,
                  b           XSD.String },
            sequence    [UNTAGGED] SEQUENCE {
                  c           XSD.String,
                  d           XSD.String }
            choice-list [UNTAGGED] SEQUENCE (SIZE(3..12)) OF [UNTAGGED] CHOICE {
                  e           XSD.String,
                  f           XSD.String } }
```

C.3.8 Mapping **complex type definition**s

C.3.8.1 The following example is a **complex type definition** whose **content type** is **empty**.

```
<xsd:complexType name="Null"/>

<xsd:complexType name="Ack">
        <xsd:sequence/>
        <xsd:attribute name="packetNumber" type="xsd:integer"/>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
Null ::= SEQUENCE {}

Ack ::= SEQUENCE {
      packetNumber [ATTRIBUTE] INTEGER OPTIONAL }
```

C.3.8.2 The following example is a **complex type definition** whose **content type** is a **simple type definition**.

```
<xsd:complexType name="Formatted">
        <xsd:simpleContent>
                <xsd:extension base="xsd:token">
                        <xsd:attribute name="format">
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:token">
                                                <xsd:enumeration value="bold"/>
                                                <xsd:enumeration value="italic"/>
                                        </xsd:restriction>
```

```
                        </xsd:simpleType>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
```

This **complex type definition** is mapped to the following ASN.1 type assignment:

```
Formatted ::= SEQUENCE {
      Format       [ATTRIBUTE] ENUMERATED {bold, italic} OPTIONAL,
      Base         [UNTAGGED] XSD.Token }
```

C.3.8.3 The following examples are **complex type definition**s whose **content type** is an **element-only** content model.

C.3.8.3.1 In the following example, the **content type** is the **model group** of a **model group definition**.

This example uses the types defined in C.3.6.

```
<xsd:complexType name="MyComplexType-1">
      <xsd:group ref="myAll"/>
</xsd:complexType>

<xsd:complexType name="MyComplexType-2">
      <xsd:group ref="myChoice" />
</xsd:complexType>

<xsd:complexType name="MyComplexType-3">
      <xsd:group ref="mySequence" maxOccurs="100"/>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-1 ::= [USE-ORDER] SEQUENCE {
      order       SEQUENCE OF ENUMERATED {a,b},
      a           XSD.String,
      b BOOLEAN }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 35 */})

MyComplexType-2 ::= SEQUENCE {
      myChoice MyChoice }

MyComplexType-3 ::= SEQUENCE {
      mySequence-list SEQUENCE (SIZE(1..100)) OF MySequence }
```

C.3.8.3.2 In the following example, the **content type** is a **model group** whose **compositor** is **choice**.

```
<xsd:complexType name="MyComplexType-4">
      <xsd:choice>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:choice>
</xsd:complexType>

<xsd:complexType name="MyComplexType-5">
      <xsd:choice minOccurs="0">
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:choice>
</xsd:complexType>

<xsd:complexType name="MyComplexType-6">
      <xsd:choice maxOccurs="5">
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:choice>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-4 ::= SEQUENCE {
      choice [UNTAGGED] CHOICE {
            a XSD.String,
            b BOOLEAN } }
MyComplexType-5 ::= SEQUENCE {
      choice [UNTAGGED] CHOICE {
```

```
            a XSD.String,
            b BOOLEAN } OPTIONAL }

MyComplexType-6 ::= SEQUENCE {
      choice-list [UNTAGGED] SEQUENCE (SIZE(1..5)) OF [UNTAGGED] CHOICE {
            a XSD.String,
            b BOOLEAN } }
```

C.3.8.3.3 In the following example, the **content type** is a **model group** whose **compositor** is **all**.

```
<xsd:complexType name="MyComplexType-7">
      <xsd:all>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:all>
</xsd:complexType>

<xsd:complexType name="MyComplexType-8">
      <xsd:all minOccurs="0">
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:all>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-7 ::= [USE-ORDER] SEQUENCE {
      order       SEQUENCE OF ENUMERATED {a,b},
      a           XSD.String,
      b           BOOLEAN }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 35 */})

MyComplexType-8 ::= [USE-ORDER] SEQUENCE {
      order       SEQUENCE OF ENUMERATED {a,b},
      a           XSD.String OPTIONAL,
      b           BOOLEAN OPTIONAL }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 35 */ })
```

C.3.8.3.4 In the following example, the **content type** is a **model group** whose **compositor** is **sequence**.

```
<xsd:complexType name="MyComplexType-9">
      <xsd:sequence>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MyComplexType-10">
      <xsd:sequence minOccurs="0">
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MyComplexType-11">
      <xsd:sequence maxOccurs="5">
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
      </xsd:sequence>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-9 ::= SEQUENCE {
      a XSD.String,
      b BOOLEAN }

MyComplexType-10 ::= SEQUENCE {
      sequence [UNTAGGED] SEQUENCE {
            a XSD.String,
            b BOOLEAN } OPTIONAL }

MyComplexType-11 ::= SEQUENCE {
      sequence-list [UNTAGGED] SEQUENCE (SIZE(1..5)) OF [UNTAGGED] SEQUENCE {
            a XSD.String,
            b BOOLEAN } }
```

C.3.8.4 The following example shows a **complex type definition** whose **content type** is a **mixed** content model.

```
<xsd:complexType name="MyComplexType-12" mixed="true">
        <xsd:sequence>
                <xsd:element name="a" type="xsd:string"/>
                <xsd:element name="b" type="xsd:boolean"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MyComplexType-13" mixed="true">
        <xsd:all>
                <xsd:element name="a" type="xsd:string"/>
                <xsd:element name="b" type="xsd:boolean"/>
        </xsd:all>
</xsd:complexType>

<xsd:complexType name="MyComplexType-14" mixed="true">
        <xsd:choice>
                <xsd:element name="a" type="xsd:string"/>
                <xsd:element name="b" type="xsd:boolean"/>
        </xsd:choice>
</xsd:complexType>

<xsd:complexType name="MyComplexType-15" mixed="true">
        <xsd:all minOccurs="0">
                <xsd:element name="a" type="xsd:string"/>
                <xsd:element name="b" type="xsd:boolean"/>
        </xsd:all>
</xsd:complexType>

<xsd:complexType name="MyComplexType-16">
        <xsd:sequence maxOccurs="unbounded" minOccurs="0">
                <xsd:element name="a" type="xsd:string"/>
                <xsd:element name="b" type="xsd:boolean"/>
        </xsd:sequence>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-12 ::= [EMBED-VALUES] SEQUENCE {
      embed-values     SEQUENCE OF XSD.String,
      a                XSD.String,
      b                BOOLEAN }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */})

MyComplexType-13 ::= [EMBED-VALUES] [USE-ORDER] SEQUENCE {
      embed-values     SEQUENCE OF XSD.String,
      order            SEQUENCE OF ENUMERATED {a,b},
      a                XSD.String,
      b                BOOLEAN }
            (CONSTRAINED BY
               {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25
      */})
            (CONSTRAINED BY
               {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 35
      */})

MyComplexType-14 ::= [EMBED-VALUES] SEQUENCE {
      embed-values     SEQUENCE OF XSD.String,
      choice           [UNTAGGED] CHOICE {
            a                XSD.String,
            b                BOOLEAN } }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */})

MyComplexType-15 ::= [EMBED-VALUES] [USE-ORDER] SEQUENCE {
      embed-values     SEQUENCE OF XSD.String,
      order            SEQUENCE OF ENUMERATED {a,b},
      a                XSD.String OPTIONAL,
      b                BOOLEAN OPTIONAL }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 35 */})
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */})
```

```
MyComplexType-16 ::= [EMBED-VALUES] SEQUENCE {
      embed-values      SEQUENCE OF XSD.String,
      sequence-list     [UNTAGGED] SEQUENCE OF [UNTAGGED] SEQUENCE {
            a                 XSD.String,
            b                 BOOLEAN } }
      (CONSTRAINED BY
            {/* Shall conform to ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 25 */})
```

C.3.8.5 The following example shows **attribute use**s of a **complex type definition** built using an **attribute group definition**.

```
<xs:attributeGroup name="AG1">
      <xs:attribute name="a1" type="xs:string"/>
      <xs:attribute name="a2" type="xs:string"/>
      <xs:attribute name="a3" type="xs:decimal"/>
</xs:attributeGroup>

<xs:attributeGroup name="AG2">
      <xs:attribute name="a1" use="prohibited"/>
      <xs:attribute name="a3" type="xs:integer"/>
</xs:attributeGroup>

<xs:complexType name="MyComplexType-17">
      <xs:attribute name="a4" type="xs:boolean"/>
      <xs:attribute name="a5" type="xs:boolean"/>
      <xs:attributeGroup ref="AG1"/>
 </xs:complexType>

<xs:complexType name="MyComplexType-18">
      <xs:complexContent>
            <xs:restriction base="MyComplexType-17">
                  <xs:attributeGroup ref="AG2"/>
                  <xs:attribute name="a4" use="prohibited"/>
            </xs:restriction>
      </xs:complexContent>
</xs:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-17 ::= SEQUENCE {
      a1 [ATTRIBUTE] XSD.String OPTIONAL,
      a2 [ATTRIBUTE] XSD.String OPTIONAL,
      a3 [ATTRIBUTE] XSD.Decimal OPTIONAL,
      a4 [ATTRIBUTE] BOOLEAN OPTIONAL,
      a5 [ATTRIBUTE] BOOLEAN OPTIONAL }

MyComplexType-18 ::= SEQUENCE {
      a2 [ATTRIBUTE] XSD.String OPTIONAL,
      a3 [ATTRIBUTE] INTEGER OPTIONAL,
      a5 [ATTRIBUTE] BOOLEAN OPTIONAL }

MyComplexType-17-derivations ::= [USE-TYPE] CHOICE {
      myComplexType-17 [NAME AS CAPITALIZED] MyComplexType-17,
      myComplexType-18 [NAME AS CAPITALIZED] MyComplexType-18 }
```

C.3.8.6 Derivation of **complex type definition**s.

```
<xsd:complexType name="MyComplexType-19">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
            <xsd:element name="c" type="xsd:boolean" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="attr1" type="xsd:short" use="required"/>
      <xsd:attribute name="attr2" type="xsd:short"/>
</xsd:complexType>

<xsd:complexType name="MyComplexType-20">
      <xsd:complexContent>
            <xsd:restriction base="MyComplexType-19">
                  <xsd:sequence>
                        <xsd:element name="a" type="xsd:token"/>
                        <xsd:element name="b" type="xsd:boolean"/>
                  </xsd:sequence>
                  <xsd:attribute name="attr2" type="xsd:short" use="prohibited"/>
            </xsd:restriction>
      </xsd:complexContent>
```

```
</xsd:complexType>

<xsd:complexType name="MyComplexType-21">
	<xsd:complexContent>
		<xsd:extension base="MyComplexType-20">
			<xsd:sequence>
				<xsd:element name="d" type="xsd:string"/>
			</xsd:sequence>
			<xsd:attribute name="attr3" type="xsd:boolean"/>
		</xsd:extension>
	</xsd:complexContent>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
MyComplexType-19 ::= SEQUENCE {
      attr1             [ATTRIBUTE] XSD.Short,
      attr2             [ATTRIBUTE] XSD.Short OPTIONAL,
      sequence-list     [UNTAGGED] SEQUENCE OF [UNTAGGED] SEQUENCE {
            a                 XSD.String,
            b                 BOOLEAN,
            c                 BOOLEAN OPTIONAL } }

MyComplexType-20 ::= SEQUENCE {
      attr1             [ATTRIBUTE] XSD.Short,
      a                 XSD.Token,
      b                 BOOLEAN }

MyComplexType-21 ::= SEQUENCE {
      attr1             [ATTRIBUTE] XSD.Short,
      attr3             [ATTRIBUTE] BOOLEAN OPTIONAL,
      a                 XSD.String,
      b                 BOOLEAN,
      d                 XSD.String }

MyComplexType-20-derivations ::= [USE-TYPE] CHOICE {
      myComplexType-20 [NAME AS CAPITALIZED] MyComplexType-20,
      myComplexType-21 [NAME AS CAPITALIZED] MyComplexType-21 }

MyComplexType-19-derivations ::= [USE-TYPE] CHOICE {
      myComplexType-19 [NAME AS CAPITALIZED] MyComplexType-19,
      myComplexType-20 [NAME AS CAPITALIZED] MyComplexType-20,
      myComplexType-21 [NAME AS CAPITALIZED] MyComplexType-21 }
```

C.3.9 Mapping **wildcard**s

For these examples, the **target namespace** is assumed to be the following URI: "http://www.asn1.org/X694/wildcard".

C.3.9.1 Attribute **wildcard**.

```
<xsd:complexType name="AnyAttribute-1">
	<xsd:anyAttribute namespace="##any"/>
</xsd:complexType>

<xsd:complexType name="AnyAttribute-2">
	<xsd:anyAttribute namespace="##other"/>
</xsd:complexType>

<xsd:complexType name="AnyAttribute-3">
	<xsd:anyAttribute namespace="##targetNamespace"/>
</xsd:complexType>

<xsd:complexType name="AnyAttribute-4">
	<xsd:anyAttribute namespace="##local http://www.asn1.org/X694/attribute"/>
</xsd:complexType>

<xsd:complexType name="AnyAttribute-5">
	<xsd:complexContent>
		<xsd:extension base="AnyAttribute-4">
			<xsd:anyAttribute namespace="##targetNamespace"/>
		</xsd:extension>
	</xsd:complexContent>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
AnyAttribute-1 ::= SEQUENCE {
      attr [ANY-ATTRIBUTES] SEQUENCE (CONSTRAINED BY {
            /* Each item shall conform to the "AnyAttributeFormat" specified in
                  ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
            OF XSD.String }

AnyAttribute-2 ::= SEQUENCE {
      attr [ANY-ATTRIBUTES EXCEPT "http://www.asn1.org/X694/wildcard"]
            SEQUENCE (CONSTRAINED BY {
            /* Each item shall conform to the "AnyAttributeFormat" specified in
                  ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
            OF XSD.String }

AnyAttribute-3 ::= SEQUENCE {
      attr [ANY-ATTRIBUTES FROM "http://www.asn1.org/X694/wildcard"]
            SEQUENCE (CONSTRAINED BY {
            /* Each item shall conform to the "AnyAttributeFormat" specified in
                  ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
            OF XSD.String }


AnyAttribute-4 ::= SEQUENCE {
      attr [ANY-ATTRIBUTES FROM ABSENT
                                    "http://www.asn1.org/X694/attribute"]
            SEQUENCE (CONSTRAINED BY  {
            /* Each item shall conform to the "AnyAttributeFormat" specified in
                  ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
            OF XSD.String }

AnyAttribute-5 ::= SEQUENCE {
      attr [ANY-ATTRIBUTES FROM ABSENT
                                    "http://www.asn1.org/X694/attribute"
                                    "http://www.asn1.org/X694/wildcard"]
            SEQUENCE (CONSTRAINED BY {
            /* Each item shall conform to the "AnyAttributeFormat" specified in
                  ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
            OF XSD.String }
```

C.3.9.2 The following is an example of a content model **wildcard**.

```
<xsd:complexType name="Any-1">
      <xsd:sequence>
            <xsd:any namespace="##any"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Any-2">
      <xsd:sequence>
            <xsd:any minOccurs="0" namespace="##other"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Any-3">
      <xsd:sequence>
            <xsd:any minOccurs="0" masOccurs="unbounded" namespace="##local"/>
      </xsd:sequence>
</xsd:complexType>
```

These **complex type definition**s are mapped to the following ASN.1 type assignments:

```
Any-1 ::= SEQUENCE {
      elem [ANY-ELEMENT] XSD.String (CONSTRAINED BY {
                  /* Shall conform to the "AnyElementFormat" specified in
                        ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */}) }

Any-2 ::= SEQUENCE {
      elem [ANY-ELEMENT EXCEPT ABSENT
                              "http://www.asn1.org/X694/wildcard"]
            XSD.String (CONSTRAINED BY {
                  /* Shall conform to the "AnyElementFormat" specified in
                        ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */})
                        OPTIONAL }

Any-3 ::= SEQUENCE {
      elem-list SEQUENCE OF elem
            [ANY-ELEMENT FROM ABSENT] XSD.String (CONSTRAINED BY {
```

```
                    /* Shall conform to the "AnyElementFormat" specified in
                         ITU-T Rec. X.693 | ISO/IEC 8825-4, clause 18 */}) }
```

NOTE – For more examples on the computation of the "NamespaceRestriction" see examples on attribute wildcards in C.3.9.1.

Annex D
Use of the mapping to provide binary encodings for W3C XML Schema

(This annex does not form an integral part of this Recommendation | International Standard)

This annex describes the use of the mapping specified in this Recommendation | International Standard in conjunction with standardized ASN.1 Encoding Rules to provide canonical and compact binary encodings for data defined by an XSD Schema.

D.1 Encoding XSD Schemas

D.1.1 XSD Schemas can be mapped to ASN.1 type definitions as specified in the body of this Recommendation | International Standard, and the top-level type can then be encoded using any of the ASN.1 encoding rules specified in ITU-T Rec. X.690 | ISO/IEC 8825-1, ITU-T Rec. X.691 | ISO/IEC 8825-2, and ITU-T Rec. X.693 | ISO/IEC 8825-4.

D.1.2 Each of these encodings has an associated object identifier value that can be used to identify the encoding in transfer. The way in which such identification is communicated to a decoder is outside the scope of this Recommendation | International Standard.

D.1.3 When the XSD Schema is not sent to the receiver by the method described in clause D.3, the way in which the receiver obtains the Schema is outside the scope of this Recommendation | International Standard.

D.2 Transfer without using the XSD Schema for Schemas

D.2.1 This method makes the assumption that the receiver knows the XSD Schema used by the sender.

D.2.2 Figure 1 shows how to use the mapping defined in this Recommendation | International Standard to encode XML documents by means of ASN.1 encoding rules.

D.2.3 The sender and the receiver use the same (fixed) XSD Schema to generate an ASN.1 schema which in turn is given to an ASN.1 compiler to generate the BER, DER, PER or XER encoding table for XML documents conforming to that XSD Schema.

D.3 Transfer using the XSD Schema for Schemas

D.3.1 Since a unique XSD Schema for Schemas is available, it is possible to proceed in two steps (see Figure 2).

D.3.2 The sender and the receiver build an ASN.1 module and an encoder/decoder from the XSD Schema for Schemas.

D.3.3 In the first step, the sender encodes in BER, DER or PER the XSD Schema for the document and sends the encoded Schema to the receiver. The receiver decodes that Schema and, using the mapping from XSD Schema to ASN.1 and an ASN.1 compiler, generates an ASN.1 module and an encoder/decoder for XML documents conforming to that Schema.

D.3.4 In the second step the sender encodes in BER, DER, PER, or XER the XML document and sends the encoded document to the receiver.