International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.694

Amendment 1

(05/2007)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

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

Amendment 1: Efficiency enhancements

ITU-T Recommendation X.694 (2004) – Amendment 1

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	**X.680–X.699**
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
TELECOMMUNICATION SECURITY	X.1000–

For further details, please refer to the list of ITU-T Recommendations.

INTERNATIONAL STANDARD ISO/IEC 8825-5
ITU-T RECOMMENDATION X.694

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

Amendment 1

Efficiency enhancements

Summary

Amendment 1 to ITU-T Rec. X.694 | ISO/IEC 8825-5 adds the specification of Version 2 of the mapping from XSD into ASN.1. The Version 2 mapping is more efficient in two main areas: the ASN.1 time types are used rather than VisibleString for mappings of dates and times; the Fast Infoset specification (ITU-T Rec. X.891 | ISO/IEC 24824-1) is used for the mapping of XSD wild-cards. Both these changes to the mapping provide much more compact binary encodings for the XML specified by the XSD.

This amendment also includes editorial clarifications, and corrections to a number of defects that were identified. Both these changes apply to both Version 1 and Version 2.

Source

Amendment 1 to ITU-T Recommendation X.694 (2004) was approved on 29 May 2007 by ITU-T Study Group 17 (2005-2008) under the ITU-T Recommendation A.8 procedure. An identical text is also published as ISO/IEC 8825-5, Amendment 1.

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 not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

		Page
1)	Summary	1
2)	Introduction	1
3)	Clause 1	2
4)	Subclause 2.1	2
5)	Subclause 2.2	3
6)	Subclause 3.1.2	3
7)	Subclause 7.1	3
8)	Subclause 7.3	4
9)	Subclauses 7.4, 7.5 and 7.6	4
10)	Subclause 7.8	4
11)	Clause 9	5
12)	Subclause 9.1	5
13)	Subclause 9.4	5
14)	Subclause 9.4 *bis*	5
15)	Subclause 10.1.1	5
16)	Subclause 10.2.1	6
17)	Subclause 10.2.2	6
18)	Subclause 10.3.2	6
19)	Subclause 10.3.4.1	6
20)	Subclause 10.3.5	6
21)	Subclause 10.4.1	7
22)	Subclause 10.4.2.1	7
23)	Subclauses 10.4.3, 10.4.4 and 10.4.5	7
24)	Subclause 10.4.6	8
25)	Clause 11 and Table 2	8
26)	Subclause 12.1.1	9
27)	Subclause 12.3.1	9
28)	Subclause 12.4.1	10
29)	Subclause 12.4.1.4	10
30)	Subclause 12.4.3	10
31)	Subclause 12.5.2.1	11
32)	Subclause 13.1	11
33)	Subclause 13.2	12
34)	Subclause 13.3	12
35)	Subclause 13.4	12
36)	Subclause 13.7	12
37)	Subclause 13.8	12
38)	Subclauses 13.9.2 *bis* and 13.9.2 *ter*	12
39)	Subclause 13.9.3	13
40)	Subclause 13.10.2	13
41)	Subclause 13.10.5	13
42)	Subclause 14.1	13
43)	Subclause 14.5	14
44)	Subclause 14.6	14

Page

45) Subclause 18.2 15
46) Subclause 18.3 15
47) Subclause 18.4 15
48) Subclause 19.1 15
49) Subclause 19.2.1 15
50) Subclause 19.2 *bis* 15
51) Subclause 19.4.2 and Table 5 16
52) Subclause 19.5 16
53) Subclause 19.6 16
54) Subclauses 20.1, 20.2, 20.3, 20.4 and 20.5 17
55) Subclause 20.8 17
56) Subclause 20.9.1 17
57) Subclause 21.1 17
58) Subclause 21.1 *bis* 18
59) Subclause 21.2 18
60) Subclause 21.2 *bis* 18
61) Subclause 21.3 19
62) Subclause 22.4 19
63) Subclause 22.5 20
64) Subclause 23.1 20
65) Subclause 23.2 20
66) Subclause 23.3 20
67) Subclause 23.7 20
68) Subclause 23.8 20
69) Subclause 23.8.3 21
70) Subclause 24.1 21
71) Subclause 24.3 21
72) Subclause 24.7 21
73) Subclause 24.8 21
74) Subclause 25.1 22
75) Subclause 25.3 22
76) Subclause 25.7 22
77) Subclause 25.8 22
78) Subclause 26.1 23
79) Subclause 26.5 23
80) Subclause 26.6 23
81) Subclause 27.1 23
82) Subclause 27.1 *bis* 24
83) Subclause 27.2 24
84) Subclause 27.7 24
85) Subclause 27.7.1 24
86) Subclause 27.7.1 *bis* 24
87) Subclause 27.9 24
88) Subclause 27.11 25
89) Subclause 27.12 25

Page

90) Subclause 27.12.2 25
91) Subclause 28.2 25
92) Subclause 28.3 25
93) Subclause 28.4 25
94) Clause 29 26
95) Clause 30 27
96) Subclause 31.2 28
97) Annex A 28
98) Annex A *bis* 32
99) Annex B 36
100) Annex C 36

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

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

Amendment 1

Efficiency enhancements

Conventions used in this amendment: Original, unchanged, text is in normal font*. Deleted text is struck-through, thus:* ~~deleted text~~*. Inserted text is underlined, thus*: inserted text*. Text in new clauses is not underlined.*

1) Summary

Replace the Summary with the following:

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, but also provides the ability to encode the specified XML with an efficient binary representation (binary XML). An XML document can be converted to binary XML (for storage or transfer) using the ASN.1 generated by this mapping, and the resulting binary can be converted back to the same XML document for further XML processing.

Two versions of the mapping are defined. Version 1 of the mapping was published in 2004, and a Corrigendum was issued to this Version renaming the types **DATE-TIME** and **DURATION** in Annex A (in order to avoid conflict with the application of ITU-T Rec. X.680/Amd. 3 | ISO/IEC 8824-1/Amd. 3 (known as the time types amendment). The Version 2 mapping is more efficient in two areas: the ASN.1 time types are used rather than VisibleString for mappings of dates and times; the Fast Infoset specification (ITU-T Rec. X.891 | ISO/IEC 24824-1) is used for the mapping of XSD wild-cards. Both these changes to the mapping provide much more compact binary encodings for the XML specified by the XSD.

NOTE – The specification of the Version 1 mapping (with applicable corrections) will be maintained in the next edition of this Recommendation | International Standard, but it is expected that subsequent editions will document only the Version 2 mapping.

Application of the ASN.1 extended XML Encoding Rules to both versions of the mapping will produce the same XML (which is the same as that specified by the XSD). However, application of other ASN.1 encoding rules to the Version 1 mapping results in a verbose character-based encoding of date and time types and of XSD wild-cards, whilst application to the Version 2 mapping results in a more compact binary encoding using ASN.1 time types and the Fast Infoset specification.

2) Introduction

Replace the Introduction with the following:

This Recommendation | International Standard specifies Version 1 and Version 2 of a mapping from a W3C XML Schema definition (an XSD Schema) into an ASN.1 schema. The mappings can be applied to any XSD Schema. Both mappings~~It~~ specify~~ies~~ 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 (produced by either Version of the mapping), 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, but produce encodings for Version 2 of the mapping that differ from (and are less verbose than) those produced by Version 1 for XSD constructs involving dates and times or wildcards.

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 originally 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 ~~data~~types for Version 1 of the mapping. 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 A *bis* also forms an integral part of this Recommendation | International Standard and provides the ASN.1 module for Version 2 of the mapping.

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 provide compact and canonical encodings for data defined using an XSD Schema.

3) Clause 1

Replace the first paragraph of clause 1 with the following (retaining the remaining paragraphs):

This Recommendation | International Standard specifies two Versions of a mapping from any XSD Schema into an ASN.1 schema. The ASN.1 schema for both Versions support~~s~~ the same semantics and validate~~s~~ the same set of XML documents.

4) Subclause 2.1

Replace subclause 2.1 with the following:

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.680 (2002)/Amd. 3 (2006)| ISO/IEC 8824-1:2002/Amd. 3:2006, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation – Amendment 3: Time type support.*
- 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.690 (2002)/Amd.2 (2006) | ISO/IEC 8825-1:2002/Amd.2:2007, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) – Amendment 2: Time type support.*

– 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.691 (2002)/Amd.2 (2006) | ISO/IEC 8825-2:2002/Amd.2:2007, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER) – Amendment 2: Time type support.* †

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

– ITU-T Recommendation X.693 (2001) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER).*

– ITU-T Recommendation X.693 (2001)/Amd.1 (2003) | ISO/IEC 8825-4:2002/Amd.1:2004, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) – Amendment 1: XER encoding instructions and EXTENDED-XER*.

– ITU-T Recommendation X.693 (2002)/Amd.2 (2006) | ISO/IEC 8825-4:2002/Amd.2:2006, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) – Amendment 2: Time type support.*

– ITU-T Recommendation X.891 (2005) | ISO/IEC 24824-1:2007, *Information technology – Generic applications of ASN.1: Fast Infoset.*

5) Subclause 2.2

In subclause 2.2, replace "ISO 8601:2000" with "ISO 8601:2004".

6) Subclause 3.1.2

Replace subclause 3.1.2 with the following:

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 terms "element information item" and "attribute information item" are~~is~~ defined in W3C XML Information Set.

NOTE 3 – The terms "top-level **simple type definition**" and "top-level **complex type definition**" do not include XSD built-in types, when used in this Recommendation | International Standard.

7) Subclause 7.1

Replace subclause 7.1 with the following:

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

NOTE 2~~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 3~~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.

8) Subclause 7.3

Replace subclause 7.3 with the following:

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.~~ One or more ASN.1 modules shall be generated for a source XSD Schema. The number of ASN.1 modules generated is an implementation option. ~~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).~~ Each ASN.1 module shall contain zero or more type assignments corresponding to top-level schema components (see 7.9), and zero or more special ASN.1 type assignments (see clauses 29, 30 and 31). The physical order of type assignments within each ASN.1 module is an implementation option. When multiple ASN.1 modules are generated, the way the generated type assignments are distributed across those ASN.1 modules is also an implementation option.

~~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.~~

NOTE 1 – The inclusion in the same ASN.1 module of type assignments generated from XSD schema components with different **target namespace**s is permitted by this subclause but not recommended. The preferred mapping is to generate one ASN.1 module per namespace whenever possible. It is also recommended that each special ASN.1 type assignment be inserted in the same ASN.1 module as its associated ASN.1 type assignment (see 29.4, 30.4 and 31.4).

NOTE 2 – The generation of ASN.1 type assignments (see 7.9 and 10.4) is not affected by the number of ASN.1 modules being generated (except for the possible use of "ExternalTypeReference" as specified in 10.2.2), nor by the way the generated type assignments are distributed across those modules, nor by the physical order of the type assignments within each module. In particular, the type reference names of those type assignments are the same whatever mapping style is used by the implementation.

NOTE 3 – 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 where needed.

9) Subclauses 7.4, 7.5 and 7.6

Delete subclauses 7.4, 7.5 and 7.6.

10) Subclause 7.8

Replace subclause 7.8 with the following:

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.

11) Clause 9

Replace clause 9 with the following:

9 ~~The~~ ASN.1 module~~s 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.~~

12) Subclause 9.1

Replace subclause 9.1 with the following:

9.1 The mapping of an XSD Schema generates one or more ASN.1 modules (see 7.3) ~~corresponding to all schema components in the Schema that have the same **target namespace**~~.

13) Subclause 9.4

Replace subclause 9.4 with the following:

9.4 In each ASN.1 module generated by a Version 1 mapping, there shall be an ASN.1 **IMPORTS** statement importing the ASN.1 type reference names in the module named **XSD {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version1(1)}** specified in Annex A that are referenced in the ASN.1 module.

14) Subclause 9.4 *bis*

Add a new subclause 9.4 bis as follows:

9.4 *bis* In each ASN.1 module generated by a Version 2 mapping, there shall be an ASN.1 **IMPORTS** statement importing the ASN.1 type reference names in the module named **XSD {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version2(2)}** specified in Annex A *bis* that are referenced in the generated ASN.1 module.

NOTE – The term "XSD module" in this Recommendation | International Standard refers to the module defined in Annex A (Version 1 mapping) or in Annex A *bis* (Version 2 mapping), according to the version of the mapping.

15) Subclause 10.1.1

Replace subclause 10.1.1 with the following:

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).

16) Subclause 10.2.1

Replace subclause 10.2.1 with the following:

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

17) Subclause 10.2.2

Replace subclause 10.2.2 with the following:

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~~ the "DefinedType" shall be an "ExternalTypeReference" ~~(constructed as appropriate for module M) for TA~~; for that type assignment, as an implementation option. O~~o~~therwise, it shall be a "typereference" for ~~TA~~that type assignment.

NOTE – All ASN.1 "typereference"s created by the mapping are unique for any legal input schema, so a type defined in another ASN.1 module does not need to be an "ExternalTypeReference".

18) Subclause 10.3.2

Replace subclause 10.3.2 with the following:

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.

19) Subclause 10.3.4.1

Replace subclause 10.3.4.1 with the following:

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:

a) the type reference name of another ASN.1 type assignment previously (see 10.4) generated by the mapping (in any ASN.1 module);~~in the same ASN.1 module or in another ASN.1 module with the same namespace (including absence of a namespace),~~ or

b) the type reference name of a type assignment in the **`XSD`** module (see Annex A); or

c) ~~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 ASN.1 module)~~in any of those ASN.1 modules~~.

NOTE – As a consequence of this rule, all type reference names defined in an ASN.1 specification generated from a source XSD schema (including the standardized type references defined in the **`XSD`** module) will be unique within that ASN.1 specification. This allows maximum flexibility in the way that the generated ASN.1 type assignments are distributed across multiple ASN.1 modules (see 7.3).

20) Subclause 10.3.5

Replace subclause 10.3.5 with the following:

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.

21) Subclause 10.4.1

Replace subclause 10.4.1 with the following:

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 ~~data~~types are mapped in a special way.

22) Subclause 10.4.2.1

Replace subclause 10.4.2.1 with the following:

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

23) Subclauses 10.4.3, 10.4.4 and 10.4.5

Replace subclauses 10.4.3, 10.4.4 and 10.4.5 with the following:

10.4.3 ~~The mapping generates~~Two sets of ASN.1 type assignments are generated by the mapping:

a) one set of ASN.1 type assignments (generated by clauses 13, 14, 15, 17 and 20) correspond directly to top-level schema components, and their type reference names are derived from the name of the schema component with no suffix appended;

b) another set of ASN.1 type assignments (generated by clauses 29, 30 and 31) correspond to special uses of top-level schema components, and their type reference names are generated from the name of the schema component followed by a suffix and (in some cases) by a post-suffix.

NOTE – For each top-level schema component in the source XSD Schema, at most one ASN.1 type assignment in the set in 10.4.3 (a) can be generated, but multiple ASN.1 type assignments in the set in 10.4.3 (b) can be generated.

~~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 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 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 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 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.~~4~~ ~~5~~ ASN.1 type assignments ~~that correspond directly to the XSD schema components~~ in the set in 10.4.3 (a) shall be generated in the order of the corresponding XSD schema components (see 10.4.1), and shall all be generated before any type assignments in 10.4.3 (b) are generated.

10.4.5 ASN.1 type assignments ~~listed~~ in 10.4.3 (b~~if any~~) shall be generated in the following order:

a) given two top-level schema components SC1 and SC2, where SC1 precedes SC2 in the order specified in 10.4.1, all the ASN.1 type assignments corresponding to SC1 (if any) shall be generated before any type assignments corresponding to SC2 are generated;

b) within each set of type assignments corresponding to any given schema component, type assignments shall be generated in an order based on the suffix specified in clauses 29 to 31, as follows:

1) suffix "**-nillable**";

2) suffix "**-nillable-default-**";

3) suffix "**-nillable-fixed-**";

4) suffix "**-derivations**";

5) suffix "**-deriv-default-**";

6) suffix "**-deriv-fixed-**";

7) suffix "**-deriv-nillable**";

8) suffix "**-deriv-nillable-default-**";

9) suffix "**-deriv-nillable-fixed-**";

10) suffix "**-group**";

c) for items 2, 3, 5, 6, 8 and 9 of (b), within each set of type assignments corresponding to any given schema component and any given suffix, type assignments shall be generated in ascending lexicographical order of the post-suffix specified in clause 29 (if any).

~~order of the XSD schema components (see 10.4.1) corresponding to the "associated type assignment" (see clauses 29, 30, and 31).~~

24) Subclause 10.4.6

Delete subclause 10.4.6.

25) Clause 11 and Table 2

Replace clause 11 and Table 2 with the following:

11 Mapping uses of XSD built-in ~~data~~types

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 ~~data~~type.

NOTE – All XSD built-in types are **simple type definition**s with the exception of **xsd:anyType**, which is a **complex type definition**.

11.2 A use of an XSD built-in ~~data~~type 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 {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version1(1)}** module (Version 1 mapping – see Annex A) or the **XSD {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version2(2)}** module (Version 2 mapping – see Annex A *bis*).

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

XSD built-in ~~data~~type	ASN.1 type definition	XSD built-in ~~data~~type	ASN.1 type definition
anyURI	XSD.AnyURI	int	XSD.Int
anySimpleType	XSD.AnySimpleType	integer	INTEGER
anyType	XSD.AnyType or XSD.AnyType-nillable (see 11.3)	language	XSD.Language
base64Binary	[BASE64] OCTET STRING	long	XSD.Long
boolean	BOOLEAN	Name	XSD.Name
byte	INTEGER (-128..127)	NCName	XSD.NCName
date	XSD.Date	negativeInteger	INTEGER (MIN..-1)
dateTime	XSD.DateTime	NMTOKEN	XSD.NMTOKEN
decimal	XSD.Decimal	NMTOKENS	XSD.NMTOKENS
double	XSD.Double	nonNegativeInteger	INTEGER (0..MAX)
duration	XSD.Duration	nonPositiveInteger	INTEGER (MIN..0)
ENTITIES	XSD.ENTITIES	normalizedString	XSD.NormalizedString
ENTITY	XSD.ENTITY	NOTATION	XSD.NOTATION
float	XSD.Float	positiveInteger	INTEGER (1..MAX)
gDay	XSD.GDay	QName	XSD.QName
gMonth	XSD.GMonth	short	XSD.Short
gMonthDay	XSD.GMonthDay	string	XSD.String
gYear	XSD.GYear	time	XSD.Time
gYearMonth	XSD.GYearMonth	token	XSD.Token
hexBinary	OCTET STRING	unsignedByte	INTEGER (0..255)
ID	XSD.ID	unsignedInt	XSD.UnsignedInt
IDREF	XSD.IDREF	unsignedLong	XSD.UnsignedLong
IDREFS	XSD.IDREFS	unsignedShort	XSD.UnsignedShort

11.3 A use of **xsd:anyType** as the **type definition** of an **element declaration** that is not **nillable** shall be mapped to **XSD.AnyType**. A use of **xsd:anyType** as the **type definition** of an **element declaration** that is **nillable** shall be mapped to **XSD.AnyType-nillable**.

26) Subclause 12.1.1

Replace subclause 12.1.1 with the following:

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

27) Subclause 12.3.1

Replace subclause 12.3.1 with the following:

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 the three following subclauses apply.

12.3.1.1 ~~n~~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.

12.3.1.2 If the **value** of the **whiteSpace** facet is **replace** and a final **TEXT** encoding instruction with qualifying information is being assigned to the ASN.1 type definition, then a final **WHITESPACE REPLACE** encoding instruction shall also be assigned to it.

12.3.1.3 If the **value** of the **whiteSpace** facet is **collapse** and a final **TEXT** encoding instruction with qualifying information is being assigned to the ASN.1 type definition, then a final **WHITESPACE COLLAPSE** encoding instruction shall also be assigned to it.

28) Subclause 12.4.1

Replace subclause 12.4.1 with the following:

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~~four following subclauses.

29) Subclause 12.4.1.4

Add a new subclause 12.4.1.4 as follows:

12.4.1.4 If the **simple type definition** has a **whiteSpace** facet with the value **preserve** or **replace**, then the enumerated type shall be assigned at least one final **TEXT** encoding instruction with qualifying information indicating one or more of the "EnumerationItem"s.

NOTE – An important example of this is a restriction of **xsd:string** with an **enumeration** facet, which has **whiteSpace preserve** by default.

30) Subclause 12.4.3

Replace subclause 12.4.3 with the following:

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 ~~data~~types **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] [USE-NUMBER] 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}
```

31) Subclause 12.5.2.1

Replace subclause 12.5.2.1 with the following:

12.5.2.1 If the **simple type definition** is derived by restriction (directly or indirectly) from an XSD built-in date or time ~~data~~type (**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 12.5.4).

32) Subclause 13.1

Replace subclause 13.1 with the following:

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**.

NOTE – This clause is not invoked for **simple type definition**s that are XSD built-in types.

33) Subclause 13.2

Delete subclause 13.2.

34) Subclause 13.3

Replace subclause 13.3 with the following:

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.

35) Subclause 13.4

Replace subclause 13.4 with the following:

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.

36) Subclause 13.7

Replace subclause 13.7 with the following:

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.

37) Subclause 13.8

Replace subclause 13.8 with the following:

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 ~~data~~type (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.

38) Subclauses 13.9.2 *bis* and 13.9.2 *ter*

Add new subclauses 13.9.2 bis and 13.9.2 ter as follows:

13.9.2 ***bis*** If the **item type definition** of the list is **xsd:string** or a restriction of **xsd:string** and is mapped to an ASN.1 character string type, then the permitted alphabet constraint **`(FROM((0, 0, 0, 33) .. (0, 16, 255, 253)))`** shall be applied to the ASN.1 character string type.

13.9.2 ***ter*** If the **item type definition** of the list is a union type, then the subtype constraint specified in 13.9.2 *bis* shall be applied to each alternative of the ASN.1 choice type that is a character string type by using an inner subtype constraint applied to the choice type.

39) Subclause 13.9.3

Replace subclause 13.9.3 with the following:

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
```

40) Subclause 13.10.2

Replace subclause 13.10.2 with the following:

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 ~~data~~type 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**.

41) Subclause 13.10.5

Replace subclause 13.10.5 with the following:

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 }
```

42) Subclause 14.1

Replace subclause 14.1 with the following:

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, 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 ~~data~~types is mapped as if it had no **value constraint**.

43) Subclause 14.5

Replace subclause 14.5 with the following:

14.5 ~~One of the two following subclauses (14.5.1 and 14.5.2) applies.~~ The ASN.1 type definition shall be generated either by applying clause 23, clause 26 or clause 27 (see 14.5 *bis*) to the **simple** or **complex type definition** that is the **type definition** of the **element declaration**, or by applying 10.2 to the ASN.1 type assignment generated by applying clause 29 to the **type definition**. In both cases, the **value constraint** in the **element declaration** (if any) shall be provided to the applicable clause (23, 26, 27 or 29) and shall be used when generating the ASN.1 type definition as specified in that clause.

14.5 *bis* The applicable clause number shall be obtained from the last column of Table 4 *bis* after selecting a row of the table based on the following conditions:

a) whether the **element declaration** has a substitutable or a non-substitutable **type definition** (see 14.6);

b) whether the **element declaration** is **nillable** or non-**nillable**;

c) whether the **type definition** is a **simple type definition** or a **complex type definition**; and

d) whether the **type definition** is an XSD built-in, anonymous, or top-level type definition.

Table 4 *bis* – Applicable clause numbers for the mapping of element declarations

Substitutable	Nillable	Simple/Complex	Type definition	Applicable clause number
no	no	**simple** or **complex**	XSD built-in, anonymous, or top-level	23
no	yes	**simple**	XSD built-in or anonymous	26
no	yes	**simple**	top-level	29
no	yes	**complex**	XSD built-in or anonymous	27
no	yes	**complex**	top-level	29
yes	yes or no	**simple** or **complex**	XSD built-in, anonymous, or top-level	29

~~**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.~~

44) Subclause 14.6

Replace subclause 14.6 with the following:

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**.

NOTE – According to this definition, **element declaration**s whose **type definition** is the XSD built-in type **xsd:anyType** do not have a substitutable **type definition**.

45) Subclause 18.2

Replace subclause 18.2 with the following:

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, an ordered list of zero or more ASN.1 "NamedType"s shall be generated by applying clause 19 to the **particle**, and ~~that~~those "NamedType"s shall be added to the sequence type in the same order~~as one of its components~~. A final **UNTAGGED** encoding instruction shall be assigned to the sequence type.

46) Subclause 18.3

Replace subclause 18.3 with the following:

18.3 A **model group** with a **compositor** of **choice** having at least one **particle** 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.

47) Subclause 18.4

Add a new subclause 18.4 as follows:

18.4 A **model group** with a **compositor** of **choice** having no **particle**s shall be mapped to the ASN.1 built-in type **NULL**.

48) Subclause 19.1

Replace subclause 19.1 with the following:

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

NOTE 1 – 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).

NOTE 2 – In most cases, this clause generates a single "NamedType". It can generate zero "NamedType"s or two or more "NamedType"s only when a sequence model group particle contains another sequence model group particle with both **min occurs** and **max occurs** equal to one, in which case the particles of the inner sequence model group are mapped to ASN.1 as though they were particles of the outer sequence model group.

49) Subclause 19.2.1

Replace subclause 19.2.1 with the following:

19.2.1 If both **min occurs** and **max occurs** of a **particle** are one, then:

a) if the **term** of the **particle** is a **model group** with a **compositor** of **sequence** unrelated to a **model group definition** and the **particle** itself belongs to a **model group** with a **compositor** of **sequence**, the **particle** is called a "pointless sequence particle";

b) otherwise, the **particle** is called a "mandatory presence particle".

50) Subclause 19.2 *bis*

Add a new subclause 19.2 bis as follows:

19.2 *bis* A "pointless sequence particle" shall be mapped to an ordered list (L, say) of zero or more "NamedType"s as follows. The list L shall be initially empty. For each **particle** (P, say) in the **model group** that is the **term** of the **particle** in order, an ordered list of zero or more "NamedType"s shall be generated by recursively applying clause 19 to the **particle** P, and those "NamedType"s shall be added to the list L in the same order.

51) Subclause 19.4.2 and Table 5

Replace subclause 19.4.2 and Table 5 with the following:

19.4.2 Unless **min occurs** is zero and **max occurs** is **unbounded**~~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.

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* ≥ 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*~~

52) Subclause 19.5

Replace subclause 19.5 with the following:

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

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

~~d~~c) 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~~d) 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~~e) if the **term** of the **particle** is a **wildcard**, the character string "**elem**".

53) Subclause 19.6

Replace subclause 19.6 with the following:

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 containing only the head itself, 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**;

NOTE 1 – This includes the frequent case in which there is no **element declaration** that references this **element declaration** as its **substitution group affiliation**.

b) if the **term** of the **particle** is a top-level **element declaration** which is the head of an element substitution group containing at least one member other than the head, 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**;

NOTE 2 – If the head is an **element declaration** that is **abstract**, it is not itself a member of the substitution group. If, in this case, the substitution group has at least one member, then this item (b) applies, and if the number of members is exactly one, then the substitution group will be mapped to an ASN.1 choice with a single alternative.

c) if the **term** of the **particle** is an **abstract** top-level **element declaration** which is the head of an empty element substitution group, the ASN.1 built-in type **NULL**;

~~e~~d) 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~~c) 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~~f) 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**;

NOTE 3 – This includes the case in which a **model group definition** within a **redefine** contains a self-reference. The **model group** of the original **model group definition**, copied into the new schema, is here considered unrelated to a **model group definition** because the original **model group definition** itself is not copied into the new schema (the new **model group definition** will have a different **model group**, which will contain a copy of the original **model group**).

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

54) Subclauses 20.1, 20.2, 20.3, 20.4 and 20.5

Replace subclauses 20.1, 20.2, 20.3, 20.4 and 20.5 with the following:

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**.

NOTE – This clause is not invoked for **complex type definition**s that are XSD built-in types.

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, and~~. Z~~zero or more components shall be added to it~~the ASN.1 sequence type~~ as specified in~~by~~ the following subclauses 20.5 to 20.11, 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 ~~data~~type **xsd:string**. A final **`EMBED-VALUES`** encoding instruction shall be assigned to the ASN.1 sequence type.

55) Subclause 20.8

Replace subclause 20.8 with the following:

20.8 If the **complex type definition** has an **attribute wildcard**, then a component generated from the **attribute wildcard** ~~(see 21.3)~~ as specified in clause 21 shall be added to the ASN.1 sequence type.

56) Subclause 20.9.1

Replace subclause 20.9.1 with the following:

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, an ordered list of zero or more ASN.1 "NamedType"s~~a component~~ shall be generated by applying clause 19 to the **particle** in the **model group**, and those "NamedType"s shall be added to the ASN.1 sequence type in the same order.

57) Subclause 21.1

Replace subclause 21.1 with the following:

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 or a "NamedType" corresponding to a ~~**simple type definition**~~**wildcard**.

58) Subclause 21.1 *bis*

Add a new subclause 21.1 bis as follows:

21.1 *bis* For a Version 1 mapping, 21.2 shall be applied. For a Version 2 mapping, 21.2 *bis* shall be applied.

59) Subclause 21.2

Replace subclause 21.2 with the following:

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 ~~data~~type **xsd:string**. A final ANY-ELEMENT encoding instruction shall be assigned to the ASN.1 type definition.

60) Subclause 21.2 *bis*

Add a new subclause 21.2 bis as follows:

21.2 *bis* A **wildcard** that is the **term** of a **particle** shall be mapped as specified in the following subclauses 21.2 *bis*.1 to 21.2 *bis*.7.

21.2 *bis*.1 The phrase "wildcard mapping attribute" (used only in this clause) designates an attribute information item with a **[namespace name]** property of "**urn:oid:2.1.5.2.0.1**" and a **[local name]** property of "**wildcard-mapping**", which is a member of the **attributes** of an **annotation** present in a **wildcard**. The phrase "value of a wildcard mapping attribute" (used only in this clause) designates the **[normalized value]** property of a wildcard mapping attribute.

NOTE – The namespace name specified in this subclause is the ASN.1 namespace name defined in ITU-T Rec. X.693 | ISO/IEC 8825-4, 16.9.

21.2 *bis*.2 A wildcard mapping attribute shall have one of the following values: "**CHOICE-FI**", "**CHOICE-UTF-8**", "**FI**", and "**UTF-8**".

EXAMPLE – The following is an example of a wildcard mapping attribute:

```
<xsd:any>
        <xsd:annotation
                a:wildcard-mapping="FI"
                xmlns:a="urn:oid:2.1.5.2.0.1"/>
</xsd:any>
```

21.2 *bis*.3 A **wildcard** without a wildcard mapping attribute shall be treated as though it has a wildcard mapping attribute with the value "**CHOICE-FI**" (if **process contents** is **strict** or **lax)** or "**FI**" (if **process contents** is **skip**).

21.2 *bis*.4 A **wildcard** whose **process contents** is **skip** shall not have a wildcard mapping attribute with the value "**CHOICE-FI**" or "**CHOICE-UTF-8**".

21.2 *bis*.5 A **wildcard** whose wildcard mapping attribute has the value "**UTF-8**" shall be mapped to the ASN.1 built-in type **UTF8String** with the following user-defined constraint:

```
(CONSTRAINED BY
             {/* Every character string abstract value shall be a well-formed XML document
             encoded in UTF-8. */})
```

and with a final **ANY-ELEMENT** encoding instruction.

21.2 *bis*.6 A **wildcard** whose wildcard mapping attribute has the value "**FI**" shall be mapped to the ASN.1 built-in type **OCTET STRING** with the following user-defined constraint:

```
(CONSTRAINED BY
             {/* Every octet string abstract value shall be a well-formed fast infoset
             document (see ITU-T Rec. X.891 | ISO/IEC 24824-1). */})
```

and with a final **ANY-ELEMENT** encoding instruction.

21.2 *bis*.7 A **wildcard** whose wildcard mapping attribute has the value "**CHOICE-FI**" or "**CHOICE-UTF-8**" shall be mapped to an ASN.1 choice type constructed as follows:

a) one alternative shall be added to the choice type for each top-level **element declaration** in the source XSD Schema which is not **abstract** and whose **target namespace** is a namespace name (or the **absent** namespace) allowed by the **namespace constraint** of the **wildcard**;

b) 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 10.2 to the ASN.1 type assignment generated by applying clause 14 to the top-level **element declaration**;

c) these 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**;

d) if the wildcard mapping attribute has the value "**CHOICE-UTF-8**", another alternative shall be added to the end of the choice type; the "identifier" in the "NamedType" shall be generated by applying 10.3 to the character string "**elem**", and the "Type" in the "NamedType" shall be the ASN.1 type specified in 21.2 *bis*.5;

e) if the wildcard mapping attribute has the value "**CHOICE-FI**", another alternative shall be added to the end of the choice type; the "identifier" in the "NamedType" shall be generated by applying 10.3 to the character string "**elem**", and the "Type" in the "NamedType" shall be the ASN.1 type specified in 21.2 *bis*.6;

f) if **process contents** is **strict**, then the following user-defined constraint shall be applied to the choice type:

```
(CONSTRAINED BY
{/* The last alternative shall be used if and only if xsi:type is present*/})
```

g) if **process contents** is **lax**, then the following user-defined constraint shall be applied to the choice type:

```
(CONSTRAINED BY
{/* The last alternative shall be used when xsi:type is present, and shall
not be used when xsi:type is not present and one of the other alternatives
can be used. */})
```

h) a final **UNTAGGED** encoding instruction shall be assigned to the choice type.

61) Subclause 21.3

Replace subclause 21.3 with the following:

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 ~~data~~type **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.

62) Subclause 22.4

Replace subclause 22.4 with the following:

22.4 If either the **attribute use** or its **attribute declaration** has a **value constraint** and the **attribute use** is not **required**, 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).

63) Subclause 22.5

Replace subclause 22.5 with the following:

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". The "Value" in the ASN.1 single value constraint shall be 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).

64) Subclause 23.1

Replace subclause 23.1 with the following:

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 one of the following~~a~~ uses of a top-level, anonymous, or XSD built-in **simple type definition** or **complex type definition**:~~-~~

a) a **simple type definition** used as the **base type** of another **simple type definition**;

b) a **simple type definition** used as the **item type** of a list type;

c) a **simple type definition** used as the **member type** of a union type;

d) ~~This includes their use~~ a **simple** or **complex type definition** used as the **type definition** of **element declaration**s that do not have a substitutable **type definition** (see 14.6)~~,~~ and are not **nillable**~~, and may or may not have a **value constraint**~~;

e) a **simple type definition** used as the **type definition** of an **attribute declaration**;

f) a **simple type definition** used as the **content type** of a **complex type definition**; and

g) a **simple type definition** used as the **type definition** of an **element declaration** that does not have a substitutable **type definition** (see 14.6) and is **nillable**.

65) Subclause 23.2

Replace subclause 23.2 with the following:

23.2 A use of a n XSD built-in ~~top-level **simple type definition**~~ or **complex type definition** ~~that is an XSD built-in datatype~~ shall be mapped to an ASN.1 type definition (a "DefinedType") as specified in clause 11.

66) Subclause 23.3

Replace subclause 23.3 with the following:

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**.

67) Subclause 23.7

Replace subclause 23.7 with the following:

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** has been provided in the invocation of this clause, 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.

68) Subclause 23.8

Replace subclause 23.8 with the following:

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** has been provided in the invocation of this clause and the value in the **value constraint**~~that~~ is a **fixed** value, then one of the three following subclauses applies.

69) Subclause 23.8.3

Replace subclause 23.8.3 with the following:

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

~~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.~~

70) Subclause 24.1

Replace subclause 24.1 with the following:

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 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)~~,~~ and are not **nillable**~~, and may or may not have a **value constraint**~~.

71) Subclause 24.3

Replace subclause 24.3 with the following:

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.

72) Subclause 24.7

Replace subclause 24.7 with the following:

24.7 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause, then a final **DEFAULT-FOR-EMPTY** encoding instruction shall be assigned to each alternative of the ASN.1 choice type corresponding to a **simple** or **complex type definition** that would validate a hypothetical element containing the canonical lexical representation of the value in the **value constraint**, but not to the other alternatives (if any). One of the three following subclauses applies.

73) Subclause 24.8

Replace subclause 24.8 with the following:

24.8 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause and the value in the **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~~four following subclauses applies to each alternative of the choice type.

24.8.1 If the alternative has been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 24.7 and 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 has been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 24.7 and 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 has been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 24.7 and 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, which in turn shall apply 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. ~~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.~~

24.8.4 If the alternative has not been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 24.7, the inner subtype constraint shall apply a presence constraint of **ABSENT** to the alternative.

74) Subclause 25.1

Replace subclause 25.1 with the following:

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 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) and, ~~are~~ are **nillable**~~, and may or may not have a **value constraint**~~.

75) Subclause 25.3

Replace subclause 25.3 with the following:

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.

76) Subclause 25.7

Replace subclause 25.7 with the following:

25.7 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause, then a final **DEFAULT-FOR-EMPTY** encoding instruction shall be assigned to each alternative of the ASN.1 choice type corresponding to a **simple** or **complex type definition** that would validate a hypothetical element containing the canonical lexical representation of the value in the **value constraint**, but not to the other alternatives (if any). One of the three following subclauses applies.

77) Subclause 25.8

Replace subclause 25.8 with the following:

25.8 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause and the value in the **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~~four following subclauses applies to each alternative of the choice type.

25.8.1 If the alternative has been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 25.7 and 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 has been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 25.7 and 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 has been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 25.7 and 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; and

b) to the **content** component, the keyword **PRESENT**.

~~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.~~

25.8.4 If the alternative has not been assigned a final **DEFAULT-FOR-EMPTY** encoding instruction in 25.7, the inner subtype constraint shall apply a presence constraint of **ABSENT** to the alternative.

78) Subclause 26.1

Replace subclause 26.1 with the following:

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 either:

a) a top-level, anonymous, or XSD built-in **simple type definition** used as the **type definition** of **element declaration**s that do not have a substitutable **type definition** (see 14.6)~~,~~ and are **nillable**~~, and may or may not have a **value constraint**~~; or

b) a top-level **simple type definition** that is a member of the derivation hierarchy of the **type definition** of **element declaration**s that have a substitutable **type definition** (see 14.6) and are **nillable**.

79) Subclause 26.5

Replace subclause 26.5 with the following:

26.5 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause, 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**.

80) Subclause 26.6

Replace subclause 26.6 with the following:

26.6 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause and the value in the **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.

81) Subclause 27.1

Replace subclause 27.1 with the following:

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 either:

a) a top-level, anonymous, or XSD built-in **complex type definition** used as the **type definition** of **element declaration**s that do not have a substitutable **type definition** (see 14.6)~~,~~ and are **nillable**~~, and may or may not have a **value constraint**~~ ; or

b) a top-level **complex type definition** that is a member of the derivation hierarchy of the **type definition** of **element declaration**s that have a substitutable **type definition** (see 14.6) and are **nillable**.

82) Subclause 27.1 *bis*

Add a new subclause 27.1 bis as follows:

27.1 *bis* A use of an XSD built-in **complex type definition** shall be mapped to an ASN.1 type definition (a "DefinedType") as specified in clause 11.

83) Subclause 27.2

Replace subclause 27.2 with the following:

27.2 A use of a top-level or anonymous **complex type definition** shall be mapped to an ASN.1 type definition as specified in subclauses 27.2 *bis* to 27.10.

27.2 *bis* The ASN.1 type definition shall be an ASN.1 sequence type, and~~. O~~one or more components shall be added to it~~the ASN.1 sequence type~~ as specified in~~by~~ the following subclauses 27.3 to 27.9, in the specified order.

84) Subclause 27.7

Replace subclause 27.7 with the following:

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

85) Subclause 27.7.1

Replace subclause 27.7.1 with the following:

27.7.1 If the **term** of the **particle** is a **model group** with a **compositor** of **sequence** ~~or **choice**~~, whose **min occurs** and **max occurs** are both one, 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 generated as follows. For each particle of the **model group** in order, an ordered list of zero or more "NamedType"s shall be generated by applying clause 19 to the **particle** in the **model group**, and those "NamedType"s shall be added to the inner ASN.1 sequence type in the same order.~~with a single component, which shall be generated by applying clause 19 to the **particle** in the **content type**.~~

86) Subclause 27.7.1 *bis*

Add a new subclause 27.7.1 bis as follows:

27.7.1 *bis* 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, or a **model group** with a **compositor** of **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**.

87) Subclause 27.9

Replace subclause 27.9 with the following:

27.9 If the **content type** of the **complex type definition** is **empty**, then an **OPTIONAL** component ~~no further components~~ 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 built-in type **NULL**.

88) Subclause 27.11

Replace subclause 27.11 with the following:

27.11 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause, then a final **DEFAULT-FOR-EMPTY** encoding instruction shall be assigned to the ASN.1 sequence type. One of the two following subclauses applies.

89) Subclause 27.12

Replace subclause 27.12 with the following:

27.12 If ~~there is~~ a **value constraint** has been provided in the invocation of this clause and the value in the **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.

90) Subclause 27.12.2

Replace subclause 27.12.2 with the following:

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; and

b) to the **content** component, the keyword **PRESENT**.

~~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.~~

91) Subclause 28.2

Replace subclause 28.2 with the following:

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

92) Subclause 28.3

Replace subclause 28.3 with the following:

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** (including H itself) in the source XSD Schema ~~that~~which is not **abstract** and ~~whose **substitution group affiliation**~~ is a member of the substitution group headed by H.

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.

93) Subclause 28.4

Replace subclause 28.4 with the following:

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 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.~~

94) Clause 29

Replace clause 29 with the following:

29 Generating special ASN.1 type assignments for types used in 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 ~~invoked by other clauses~~ generates a special ASN.1 type assignment for a given combination of the following conditions and data provided in the invocation of this clause:

a) whether the **element declaration** has a substitutable or a non-substitutable **type definition** (see 14.6);

b) whether the **element declaration** is **nillable** or non-**nillable**;

c) whether the **type definition** is a **simple type definition** or a **complex type definition**;

d) whether the **element declaration** has a **value constraint**, and if it does, whether the value in the **value constraint** is a **default** value or a **fixed** value;

e) the name of the **type definition;** and

f) the value in the **value constraint** (if any).

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

~~b) whether the **element declaration**s have 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 over the mapping of a source XSD Schema (but see 29.3 *bis*).

NOTE – For example, if two ~~or more~~ **element declaration**s in a large XSD Schema have identical **type definition**s, 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.

29.3 *bis* When this clause is invoked for a **simple type definition** or **complex type definition** used as the **type definition** of an **element declaration** that is **nillable** and does not have a substitutable **type definition**, it produces an ASN.1 type assignment (using the suffix "`-nillable`") which could also be produced by an invocation of clause 30 for the same **simple type definition** or **complex type definition**. In such cases, there shall be only one such ASN.1 type assignment generated either by this clause or by clause 30, whichever is invoked first.

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 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 definition**s and **complex type definition**s 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 (see 29.6) and a post-suffix (see 29.6) 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~~ one of the clauses 24, 25, 26 and 27 (see 29.6) to the **simple type definition** or **complex type definition** that is the **type definition** of the **element declaration.** The **value constraint** in the **element declaration** (if any) shall be provided to the applicable clause (24, 25, 26 or 27) and shall be used when generating the ASN.1 type definition as specified in that clause. ~~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.~~

29.6 The suffix and the applicable clause number shall be obtained from the last two columns of Table 6 after selecting a row of the table based on the conditions listed in 29.2 (a) to (d). If there is a **value constraint**, the post-suffix shall be the canonical lexical representation (see W3C XML Schema Part 2, 2.3.1) of the value in the **value constraint**, otherwise it shall be an empty string.

Table 6 – Suffixes and applicable clause numbers for the generation of special ASN.1 type assignments

Substitutable	Nillable	Simple/Complex	Value constraint	Suffix	Applicable clause number
no	yes	**simple**	none	`-nillable`	26
no	yes	**simple**	**default**	`-nillable-default-`	26
no	yes	**simple**	**fixed**	`-nillable-fixed-`	26
no	yes	**complex**	none	`-nillable`	27
no	yes	**complex**	**default**	`-nillable-default-`	27
no	yes	**complex**	**default**	`-nillable-fixed-`	27
yes	no	**simple** or **complex**	none	`-derivations`	24
yes	no	**simple** or **complex**	**default**	`-deriv-default-`	24
yes	no	**simple** or **complex**	**fixed**	`-deriv-fixed-`	24
yes	yes	**simple** or **complex**	none	`-deriv-nillable`	25
yes	yes	**simple** or **complex**	**default**	`-deriv-nillable-default-`	25
yes	yes	**simple** or **complex**	**fixed**	`-deriv-nillable-fixed-`	25

95) Clause 30

Replace clause 30 with the following:

30 Generating special ASN.1 type assignments for types belonging to a derivation hierarchy~~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 ~~is invoked by other clauses~~ generates a special ASN.1 type assignment 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)")~~ provided in the invocation of this clause.

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 over the mapping of a source XSD Schema (but see 29.3 *bis*).

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.

NOTE – This clause specifies only the suffix "**-nillable**" (and not the suffixes "**-nillable-default-**" and "**-nillable-fixed-**"), because even if the **element declaration** has a **value constraint**, that **value constraint** is not visible to clauses 26 and 27 when invoked by this clause.

96) Subclause 31.2

Replace subclause 31.2 with the following:

31.2 This clause ~~is invoked by other clauses~~ generates a special ASN.1 type assignment 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)")~~ provided in the invocation of this clause.

97) Annex A

Replace Annex A with the following:

Annex A

ASN.1 type definitions corresponding to XSD built-in ~~data~~types for the Version 1 mapping

(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 ~~data~~types and that are used for the mapping from W3C XML Schema to ASN.1 for the Version 1 mapping.

A.2 W3C XML Schema defines many built-in date and time ~~data~~types 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 ~~data~~types 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 for the Version 1 mapping is:

```
XSD {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version1(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 */ } )
```

```
AnyType-nillable ::= 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,
            content SEQUENCE {
                  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 */ } )
            } OPTIONAL }
            (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 ::= DateTimeType (DateOnly)
/* xsd:dateTime */
DateTime ::= DateTimeType
/* 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 ::= DurationType
/* 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 ::= DateTimeType (Day)
/* xsd:gMonth */
GMonth ::= DateTimeType (Month)
/* xsd:gMonthDay */
GMonthDay ::= DateTimeType (MonthDay)
/* xsd:gYear */
GYear ::= DateTimeType (Year)
/* xsd:gYearMonth */
GYearMonth ::= DateTimeType (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 ::= DateTimeType (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 */
DurationType ::= VisibleString  (FROM ("0".."9" | "DHMPSTY:.+-"))
                                    (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.6 */})
DateTimeType ::= VisibleString  (FROM ("0".."9" | "TZ:.+-"))
                                    (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.7 */})
DateOnly ::= DateTimeType  (FROM ("0".."9" | "Z:+-"))
                                    (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.9 */})
Day ::= DateTimeType       (FROM ("0".."9" | "Z:+-"))
                               (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.13 */})
Month ::= DateTimeType     (FROM ("0".."9" | "Z:+-"))
                  (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.14 */})
MonthDay ::= DateTimeType(FROM ("0".."9" | "Z:+-"))
                               (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.12 */})
Year ::= DateTimeType      (FROM ("0".."9" | "Z:+-"))
                               (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.11 */})
YearMonth ::= DateTimeType      (FROM ("0".."9" | "Z:+-"))
                                    (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.10 */})
TimeOnly ::= DateTimeType  (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, AnyType-nillable
                  ANY-ATTRIBUTES AnyType.attr, AnyType-nillable.attr
                  ANY-ELEMENT AnyType.elem-list.* , AnyType-nillable.content.elem-list.*
                  UNTAGGED AnyType.elem-list, AnyType-nillable.content.elem-list

                  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
                  USE-NIL AnyType-nillable
                  WHITESPACE AnyURI, Language, Token, DurationType, DateTimeType COLLAPSE
                  WHITESPACE NormalizedString REPLACE
END
```

98) Annex A *bis*

Add a new Annex A bis as follows:

Annex A *bis*

ASN.1 type definitions corresponding to XSD built-in types for the Version 2 mapping

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

A *bis*.1 This annex specifies a module that defines the ASN.1 types that correspond to the XSD built-in types and that are used for the mapping from W3C XML Schema to ASN.1 for the Version 2 mapping.

A *bis*.2 W3C XML Schema defines many built-in date and time types 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 types are normally mapped to one of the ASN.1 time types, but where XSD provides additional abstract values, the mapping is to a CHOICE of an ASN.1 time type and a **VisibleString** with a user-defined constraint referencing the applicable XSD clause.

A *bis*.3 The **XSD** module for the Version 2 mapping is:

```
XSD {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version2(2)}
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 */ } )
AnyType-nillable ::= 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,
              content SEQUENCE {
                    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 */ } )
              } OPTIONAL }
              (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:date */
Date ::= GenericTimeTypeChoice {
                              TIME (SETTINGS "Basic=Date Date=YMD"),
                              VisibleString
                              (FROM ("0".."9" | "DHMPSTY:.+-"))
                              (CONSTRAINED BY {/* W3C XML Schema 1.0 Part 2, 3.2.9
                                                  and used if a time-zone is present */})}
```

```
/* xsd:dateTime */
DateTime ::= TIME ((SETTINGS "Basic=Date-Time Date=YMD")
                                    EXCEPT (SETTINGS "Midnight=End"))
             (CONSTRAINED BY {/*The time-zone shall be in the range -14 to +14*/})
             (CONSTRAINED BY {/*The seconds and fractions of a second shall be less
                               than 60 (no leap seconds supported, in accordance with
                               W3C XML Schema 1.0 Part 2, 3.2.7)*/})
(CONSTRAINED BY {/*The type is constrained to "Time=HMSFn" for any n*/})/* 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 ::= GenericTimeTypeChoice {
                              DURATION
                               ((WITH COMPONENTS {...,
                                                  seconds ABSENT,
                                                  fractional-part ABSENT})|
                                (WITH COMPONENTS {...,
                                                  seconds PRESENT})),
                              VisibleString
                                     (FROM ("0".."9" | "DHMPSTY:.+-"))
                               (CONSTRAINED BY {/* W3C XML Schema 1.0 Part 2, 3.2.6
                                                and used for negative durations */})}

/* 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 ::= DateTimeType (Day)
             /* This is an integer followed optionally by a time-zone.
                It is not supported in either ISO 8601 or in ASN.1, so the Version 1
                mapping has been retained (similarly for other "G" types). */
/* xsd:gMonth */
GMonth ::= DateTimeType (Month)
/* xsd:gMonthDay */
GMonthDay ::= DateTimeType (MonthDay)
/* xsd:gYear */
GYear ::= GenericTimeTypeChoice {
                              TIME (SETTINGS "Basic=Date Date=Y"),
                              VisibleString
                               (FROM ("0".."9" | "Z:+-"))
                               (CONSTRAINED BY {/* W3C XML Schema 1.0 Part 2, 3.2.11
                                                and used if a time-zone is present */})}

/* xsd:gYearMonth */
GYearMonth ::= GenericTimeTypeChoice {
                              TIME (SETTINGS "Basic=Date Date=YM"),
                              VisibleString
                               (FROM ("0".."9" | "Z:+-"))
                               (CONSTRAINED BY {/* W3C XML Schema 1.0 Part 2, 3.2.14
                                                and used if a time-zone is present */})}
/* xsd:ID */
ID ::= NCName
/* xsd:IDREF */
IDREF ::= NCName
/* xsd:IDREFS */
IDREFS ::= SEQUENCE (SIZE(1..MAX)) OF IDREF
/* xsd:int */
```

```
Int ::= INTEGER (-2147483648..2147483647)
/* 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: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: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:QName */
QName ::= SEQUENCE {
             uri   AnyURI OPTIONAL,
             name  NCName }
/* xsd:short */
Short ::= INTEGER (-32768..32767)
/* xsd:string */
String ::= XMLCompatibleString
/* xsd:time */
Time ::= TIME ((SETTINGS "Basic=Time")
                                    EXCEPT (SETTINGS "Midnight=End"))
             (CONSTRAINED BY {/*The time-zone shall be in the range -14 to +14*/})
             (CONSTRAINED BY {/*The seconds and fractions of a second shall be less
                               than 60 (no leap seconds supported, in accordance with
                               W3C XML Schema 1.0 Part 2, D.2)*/})
             (CONSTRAINED BY {/*Constrained to "Time=HMSFn" for any n*/})
/* xsd:token */
Token ::= NormalizedString (CONSTRAINED BY {
             /* The NormalizedString shall be a token as defined in W3C XML Schema Part 2,
                3.3.2 */})
/* 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 types */
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 types */
GenericTimeTypeChoice {BasicType, Alternative} ::= CHOICE {
                       asn1supportedvalue     BasicType,
                       othervalues            Alternative}
                       (CONSTRAINED BY
                         {/* The "othervalues" alternative shall not be used for abstract
                             values in the "asn1supportedvalue" alternative */})

DateTimeType ::= VisibleString  (FROM ("0".."9" | "TZ:.+-"))
                       (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.7 */})
Day ::= DateTimeType       (FROM ("0".."9" | "Z:+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.13 */})
Month ::= DateTimeType     (FROM ("0".."9" | "Z:+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.14 */})
MonthDay ::= DateTimeType(FROM ("0".."9" | "Z:+-"))
                             (CONSTRAINED BY {/* W3C XML Schema Part 2, 3.2.12 */})
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
                   DECIMAL Decimal
                   LIST ENTITIES, IDREFS, NMTOKENS
                   EMBED-VALUES AnyType, AnyType-nillable
                   ANY-ATTRIBUTES AnyType.attr, AnyType-nillable.attr
                   ANY-ELEMENT AnyType.elem-list.*, AnyType-nillable.content.elem-list.*
                   UNTAGGED AnyType.elem-list, AnyType-nillable.content.elem-list
                   NAME AnySimpleType, AnyURI, Date, DateTime, Decimal, Double, Duration,
                         Float, GDay, GMonth, GMonthDay, GYear, GYearMonth,
                         Int, Language, Long,
                                    NormalizedString, Short,
                         String, Time, Token,
                         UnsignedInt, UnsignedLong, UnsignedShort
                         AS UNCAPITALIZED
                   NAME GenericTimeTypeChoice.ALL AS ""
                   USE-NIL AnyType-nillable
                   USE-UNION GenericTimeTypeChoice
                   WHITESPACE AnyURI, Language, Token, DateTimeType COLLAPSE
                   WHITESPACE NormalizedString REPLACE
END
```

99) Annex B

Replace Annex B with the following:

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 ~~data~~types:

```
{ joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version1(1)}
"ASN.1 XSD Module for Version 1 mapping "
{ joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version2(2)}
     "ASN.1 XSD Module for Version 2 mapping"
```

100) Annex C

Replace Annex C with the following:

Annex C

Examples of mappings

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

This annex illustrates the Version 1 mapping specified in this Recommendation | International Standard by giving an ASN.1 module corresponding to an XSD Schema. The Version 2 mapping is similar, and differs (for this example) only in the use of the XSD module from Annex A *bis* instead of the XSD module from Annex A.

C.1 A Schema using simple type definitions

The following Schema contains examples of XSD built-in ~~data~~types (**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" type="xsd:anyType"/>
            <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 AUTOMATIC TAGS::=
BEGIN
```

```
IMPORTS String, Decimal, Int, Date, AnyType

        FROM XSD
          {joint-iso-itu-t asn1(1) specification(0) modules(0) xsd-module(2) version1(1)};
        /* For a Version 2 mapping, the last component of the module identifier would be
        version2(2) */

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       XSD.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 [NAME AS CAPITALIZED]ChildInformation }
ChildInformation ::= SEQUENCE {
              name             Name,
              dateOfBirth      XSD.Date }
ListOfDays ::=  [LIST] SEQUENCE OF Day
Day ::=    ENUMERATED ~~{monday, tuesday, wednesday, thursday, friday,
            saturday, sunday}~~
              { friday, monday, saturday, sunday, thursday, tuesday, wednesday }
              -- Note that 12.4.1.3 specifies use of a lexicographical order, as
              -- the members of an enumeration are not ordered in an XML Schema
NamesOfMemberNations ::= [NAME AS UNCAPITALIZED] [LIST] SEQUENCE OF
              XSD.String (FROM({0, 0, 0, 33} .. {0, 16, 255, 253}))
FileIdentifier ::= [NAME AS UNCAPITALIZED] SEQUENCE {
              choice [UNTAGGED] CHOICE {
                    serialNumber     XSD.Int,
                    relativeName     XSD.String,
                    unidentified     Unidentified~~NIDENTIFIED~~  } }
~~UNIDENTIFIED ::=  [NAME AS LOWERCASED] XSD.AnyType~~
Unidentified ::=  [NAME AS UNCAPITALIZED] XSD.AnyType

ENCODING-CONTROL XER
              GLOBAL-DEFAULTS MODIFIED-ENCODINGS
              GLOBAL-DEFAULTS CONTROL-NAMESPACE
                    "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
              TEXT Day:ALL
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: xyz abc="http://example.com/xyzabc"
        targetNamespace="http://example.com/abcxyz">

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

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

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

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

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

        <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 XER INSTRUCTIONS AUTOMATIC TAGS ::=
        BEGIN
        IMPORTS
        String FROM XSD{joint-iso-itu-t asn1(1) specification(0) modules(0)
                        xsd-module(2) version1(1)};
                    /* For a Version 2 mapping, the last component of the module
                       identifier would be version2(2) */
        ;
        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"
                    NAMESPACE ALL AS "http://example.com/xyz"

        END
```

```
ABC  -- The module reference is not standardized
        DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::=
        BEGIN
        IMPORTS
        Xyz-type FROM XYZ
        ~~Token,~~ String FROM XSD{joint-iso-itu-t asn1(1) specification(0) modules(0)
                              xsd-module(2) version1(1)};
                        /* For a Version 2 mapping, the last component of the module
                           identifier would be version2(2) */
        ;

        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~~String

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

        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:max~~in~~Inclusive 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~~memberType="xsd:integer xsd:boolean"/>
</xsd:simpleType>

<xsd:simpleType name="Time-or-int-or-boolean--or-dateRestriction">
      <xsd:union ~~itemType~~memberType=" 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 definition**s 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:simpleContent>
            <xsd:extension base="Int-10-to-50">
                  <xsd:attribute name="procurement" type="Int-10-to-50"/>
            </xsd:extension>
      </xsd:simpleContent>
</xsd:complexType>
```

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

```
Int-10-to-50 ::= INTEGER (10<..<50)
Ten-multiples ::= [USE-NUMBER] ENUMERATED {int20(20), int30(30), int40(40)}
Twenty-multiples ::= [USE-NUMBER] ENUMERATED {int20(20), int40(40)}
Stock-level ::= SEQUENCE {
            procurement [ATTRIBUTE] Int-10-to-50 OPTIONAL,
            base        [UNTAGGED] 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,
            stock-level      [NAME AS CAPITALIZED] Stock-level,
            ten-multiples    [NAME AS CAPITALIZED] Ten-multiples,
            twenty-multiples [NAME AS CAPITALIZED] Twenty-multiples}~~,~~
            ~~stock-level      [NAME AS CAPITALIZED] Stock-level }~~
```

if and only if:

a) the **simple type definition** "**Int-10-to-50**" occurs as the **type definition** of at least one **element declaration** (not shown in the example) that is being mapped to ASN.1;

b) the **simple type definition** "**Ten-multiples**" occurs as the **type definition** of at least one **element declaration** (not shown in the example) that is being mapped to ASN.1; and

c) there are no other schema components being mapped to ASN.1 which are generating the ASN.1 type reference names `Int-10-to-50`, `Ten-multiples`, `Twenty-multiples`, `Stock-level`, `Ten-multiples-derivations`, and `Int-10-to-50-derivations`.

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="[&# x2000;-&# x 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
                       "[&# x2000;-&# x FF; ]* " | "/?([^/]*&#x2F;)*[^/]*&#x2F;*" */})
```

C.3.3.3 whiteSpace

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

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

<xsd:simpleType name="My-TokenString">
        <xsd:restriction base="xsd:string">
                <xsd:whiteSspace 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 REPLACECOLLAPSE] 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 value ="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 ::= [WHITESPACE REPLACE]
        -- This encoding instruction ensures that an enumeration such as
        -- value="Slow Loris" (containing a space) is encoded correctly.
        ENUMERATED {horse, bull, cow, pig, duck, goose}
        { bull, cow, duck, goose, horse, pig }
```

```
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 ::= [WHITESPACE REPLACE] 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~~value="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"]
                                        INTEGER (0..MAX)~~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 {
        bar [NAME AS CAPITALIZED] Bar,
        foo [NAME AS CAPITALIZED] Foo}~~,~~
        ~~bar [NAME AS CAPITALIZED] Bar }~~
```

if and only if:

a) the **element declaration** "**Tic**" occurs as the **term** of at least one **particle** (not shown in the example) that is being mapped to ASN.1;

b) the **element declaration** "**Foo**" occurs as the **term** of at least one **particle** (not shown in the example) that is being mapped to ASN.1; and

c) there are no other schema components being mapped to ASN.1 which are generating the ASN.1 type reference names `Tac`, `Toe`, `Foo`, `Bar`, `Tic-group`, and `Foo-group`.

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

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

```
InternationalTelephone ::= [DEFAULT-FOR-EMPTY AS "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:complexType 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:complexType>
</xsd:element>
```

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

```
Description ::= [EMBED-VALUES] [DEFAULT-FOR-EMPTY AS "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 AS 20]
                                               Int-10-to-50,
        stock-level      [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY AS 20]
                                               Stock-level,
        ten-multiples    [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY AS int20]
                                               Ten-multiples,
        twenty-multiples [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY AS 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 AS "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:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:token">
                <xsd:attribute name="country-code" type="xsd:integer"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```

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

```
UnknownInternationalTelephone ::= [DEFAULT-FOR-EMPTY AS "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:complexType 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:complexType>
</xsd:element>
```

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

```
UnknownDescription ::= [EMBED-VALUES] [DEFAULT-FOR-EMPTY AS "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 in 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 AS 20]
                             Int-10-to-50,
        stock-level      [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY AS 20]
                             Stock-level,
        ten-multiples    [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY AS int20]
                             Ten-multiples,
        twenty-multiples [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY AS int20]
                             Twenty-multiples},
        stock-level      [NAME AS CAPITALIZED] [DEFAULT-FOR-EMPTY 20]
                             Stock-level }
```

```
(WITH COMPONENTS {
      int-10-to-50 (20),
      stock-level (WITH COMPONENTS {..., base (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 an **element declaration** that is **nillable** and whose **type definition** is an XSD built-in ~~data~~type.

```
<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="xsd: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 {
            form [ATTRIBUTE] Form DEFAULT qualified,
            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
INTEGER (0..MAX)
MaxOccurs ::= [ATTRIBUTE] [NAME AS UNCAPITALIZED] [USE-UNION] CHOICE {
            nonNegativeInteger     [NAMESPACE AS "http://www.w3.org/2001/XMLSchema"]
                                          INTEGER (0..MAX)XSD.NonNegativeInteger,
            alt                    [NAME AS ""] [WHITESPACE COLLAPSE] ENUMERATED
                                   {unbounded} }
Particle ::= SEQUENCE {
            maxOccurs          [ATTRIBUTE] MaxOccurs DEFAULT alt : unbounded,
            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 ""myAll">
        <xsd:all>
            <xsd:element name="a" type="xsd:string"/>
            <xsd:element name="b" type="xsd:boolean"/>
        </xsd:all>
    </xsd: group></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(51..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 },
            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 definitions

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 {
            ~~F~~format      [ATTRIBUTE] [WHITESPACE COLLAPSE] ENUMERATED {bold, italic}
                              OPTIONAL,
            ~~B~~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" mixed="true">
        <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 }
```

if and only if:

a) the **simple type definition** "**MyComplexType-19**" occurs as the **type definition** of at least one **element declaration** (not shown in the example) that is being mapped to ASN.1;

b) the **simple type definition** "**MyComplexType-20**" occurs as the **type definition** of at least one **element declaration** (not shown in the example) that is being mapped to ASN.1; and

c) there are no other schema components being mapped to ASN.1 which are generating the ASN.1 type reference names **`MyComplexType-19`**, **`MyComplexType-20`**, **`MyComplexType-21`**, **`MyComplexType-19-derivations`**, and **`MyComplexType-20-derivations`**.

C.3.9 Mapping **wildcards**

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 ABSENT "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" maxsOccurs="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 198 */}) }
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 198 */})
                        OPTIONAL }
Any-3 ::= SEQUENCE {
            elem-list [UNTAGGED] 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 198 */}) }
```

NOTE – For more examples on the computation of the "NamespaceRestriction", see examples on attribute wildcards in C.3.9.1.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks, open system communications and security**
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2008