BS ISO/IEC 8825-3:2021

BSI Standards Publication

Information technology — ASN.1 encoding rules

Part 3: Specification of Encoding Control Notation (ECN)

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 8825-3:2021

National foreword

This British Standard is the UK implementation of ISO/IEC 8825-3:2021.

The UK participation in its preparation was entrusted to Technical Committee IST/6, Data communications.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2021
Published by BSI Standards Limited 2021

ISBN 978 0 539 14460 4

ICS 35.100.60

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 July 2021.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 8825-3:2021

INTERNATIONAL STANDARD

ISO/IEC 8825-3

Fourth edition
2021-06

Information technology — ASN.1 encoding rules —

Part 3: Specification of Encoding Control Notation (ECN)

Technologies de l'information — Règles de codage ASN.1 —
Partie 3: Spécification de la notation de contrôle de codage (ECN)

Reference number
ISO/IEC 8825-3:2021(E)

Documents
Document Sharing Hub

© ISO/IEC 2021

Full document details: BS ISO/IEC 8825-3:2021

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 8825-3:2021

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs)

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems,* in collaboration with ITU-T. The identical text is published as ITU-T X.692 (02/2021).

This fourth edition cancels and replaces the third edition (ISO/IEC 8825-3:2015), which has been technically revised.

A list of all parts in the ISO/IEC 8825 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

Documents
Document Sharing Hub

CONTENTS

Page

Introduction x

1 Scope 1

2 Normative references 1
- 2.1 Identical Recommendations | International Standards 1
- 2.2 Additional references 2

3 Definitions 2
- 3.1 ASN.1 definitions 2
- 3.2 ECN-specific definitions 2

4 Abbreviations 5

5 Definition of ECN syntax 5

6 Encoding conventions and notation 5

7 The ECN character set 5

8 ECN lexical items 6
- 8.1 Encoding object references 6
- 8.2 Encoding object set references 6
- 8.3 Encoding class references 6
- 8.4 Reserved word items 7
- 8.5 Reserved encoding class name items 7
- 8.6 Non-ECN item 7

9 ECN Concepts 7
- 9.1 Encoding Control Notation (ECN) specifications 8
- 9.2 Encoding classes 8
- 9.3 Encoding structures 9
- 9.4 Encoding objects 9
- 9.5 Encoding object sets 9
- 9.6 Defining new encoding classes 9
- 9.7 Defining encoding objects 11
- 9.8 Differential encoding-decoding 11
- 9.9 Encoders options in encodings 12
- 9.10 Properties of encoding objects 12
- 9.11 Parameterization 12
- 9.12 Governors 13
- 9.13 General aspects of encodings 13
- 9.14 Identification of information elements 13
- 9.15 Reference fields and determinants 14
- 9.16 Replacement classes and structures 14
- 9.17 Mapping abstract values onto fields of encoding structures 15
- 9.18 Transforms and transform composites 15
- 9.19 Contents of Encoding Definition Modules 16
- 9.20 Contents of the Encoding Link Module 17
- 9.21 Defining encodings for primitive encoding classes 17
- 9.22 Application of encodings 19
- 9.23 Combined encoding object set 19
- 9.24 Application point 19
- 9.25 Conditional encodings 20
- 9.26 Other conditions for applying encodings 20
- 9.27 Encoding control for the open type 21
- 9.28 Changes to ASN.1 Recommendations | International Standards 21

10 Identifying encoding classes, encoding objects, and encoding object sets 21

11 Encoding ASN.1 types 24
- 11.1 General 24

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

11.2 Built-in encoding classes used for implicitly generated encoding structures ... 24
11.3 Simplification and expansion of ASN.1 notation for encoding purposes ... 25
11.4 The implicitly generated encoding structure ... 27

12 The Encoding Link Module (ELM) ... 27
12.1 Structure of the ELM ... 28
12.2 Encoding types ... 28

13 Application of encodings ... 29
13.1 General ... 29
13.2 The combined encoding object set and its application ... 29

14 The Encoding Definition Module (EDM) ... 31

15 The renames clause ... 33
15.1 Explicitly generated and exported structures ... 33
15.2 Name changes ... 34
15.3 Specifying the region for name changes ... 35

16 Encoding class assignments ... 36
16.1 General ... 36
16.2 Encoding structure definition ... 39
16.3 Alternative encoding structure ... 41
16.4 Repetition encoding structure ... 41
16.5 Concatenation encoding structure ... 42

17 Encoding object assignments ... 42
17.1 General ... 42
17.2 Encoding with a defined syntax ... 43
17.3 Encoding with encoding object sets ... 44
17.4 Encoding using value mappings ... 44
17.5 Encoding an encoding structure ... 45
17.6 Differential encoding-decoding ... 47
17.7 Encoding options ... 48
17.8 Non-ECN definition of encoding objects ... 49

18 Encoding object set assignments ... 49
18.1 General ... 49
18.2 Built-in encoding object sets ... 50

19 Mapping values ... 51
19.1 General ... 51
19.2 Mapping by explicit values ... 52
19.3 Mapping by matching fields ... 53
19.4 Mapping by `#TRANSFORM` encoding objects ... 54
19.5 Mapping by abstract value ordering ... 54
19.6 Mapping by value distribution ... 56
19.7 Mapping integer values to bits ... 57

20 Defining encoding objects using defined syntax ... 58

21 Types used in defined syntax specification ... 59
21.1 The `Unit` type ... 59
21.2 The `EncodingSpaceSize` type ... 59
21.3 The `EncodingSpaceDetermination` type ... 60
21.4 The `UnusedBitsDetermination` type ... 60
21.5 The `OptionalityDetermination` type ... 61
21.6 The `AlternativeDetermination` type ... 62
21.7 The `RepetitionSpaceDetermination` type ... 62
21.8 The `Justification` type ... 63
21.9 The `Padding` type ... 64
21.10 The `Pattern` and `Non-Null-Pattern` types ... 64
21.11 The `RangeCondition` type ... 65

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

21.12 The `Comparison` type ... 65
21.13 The `SizeRangeCondition` type ... 66
21.14 The `ReversalSpecification` type ... 66
21.15 The `ResultSize` type ... 67
21.16 The `HandleValueSet` type ... 67
21.17 The `IntegerMapping` type ... 68

22 Commonly used encoding property groups ... 68
22.1 Replacement specification ... 68
22.1.1 Encoding properties, syntax and purpose ... 68
22.1.2 Specification restrictions ... 69
22.1.3 Encoder actions ... 70
22.1.4 Decoder actions ... 70
22.2 Pre-alignment and padding specification ... 71
22.2.1 Encoding properties, syntax and purpose ... 71
22.2.2 Specification constraints ... 71
22.2.3 Encoder actions ... 71
22.2.4 Decoder actions ... 71
22.3 Start pointer specification ... 72
22.3.1 Encoding properties, syntax and purpose ... 72
22.3.2 Specification constraints ... 72
22.3.3 Encoder actions ... 72
22.3.4 Decoder actions ... 72
22.4 Encoding space specification ... 73
22.4.1 Encoding properties, syntax and purpose ... 73
22.4.2 Specification restrictions ... 73
22.4.3 Encoder actions ... 74
22.4.4 Decoder actions ... 74
22.5 Optionality determination ... 75
22.5.1 Encoding properties, syntax and purpose ... 75
22.5.2 Specification restrictions ... 75
22.5.3 Encoder actions ... 76
22.5.4 Decoder actions ... 76
22.6 Alternative determination ... 77
22.6.1 Encoding properties, syntax and purpose ... 77
22.6.2 Specification restrictions ... 77
22.6.3 Encoder actions ... 77
22.6.4 Decoder actions ... 78
22.7 Repetition space specification ... 78
22.7.1 Encoding properties, syntax and purpose ... 78
22.7.2 Specification constraints ... 79
22.7.3 Encoder actions ... 80
22.7.4 Decoder actions ... 80
22.8 Value padding and justification ... 81
22.8.1 Encoding properties, syntax, and purpose ... 81
22.8.2 Specification restrictions ... 82
22.8.3 Encoder actions ... 82
22.8.4 Decoder actions ... 83
22.9 Identification handle specification ... 83
22.9.1 Encoding properties, syntax and purpose ... 83
22.9.2 Specification constraints ... 84
22.9.3 Encoders actions ... 84
22.9.4 Decoders actions ... 84
22.10 Concatenation specification ... 84
22.10.1 Encoding properties, syntax and purpose ... 84
22.10.2 Specification constraints ... 85
22.10.3 Encoder actions ... 85
22.10.4 Decoder actions ... 85
22.11 Contained type encoding specification ... 85
22.11.1 Encoding properties, syntax and purpose ... 85
22.11.2 Encoder actions ... 86

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

22.11.3 Decoder actions ... 86
22.12 Bit reversal specification ... 86
22.12.1 Encoding properties, syntax, and purpose ... 86
22.12.2 Specification constraints ... 86
22.12.3 Encoder actions ... 87
22.12.4 Decoder actions ... 87
23 Defined syntax specification for bit-field and constructor classes ... 87
23.1 Defining encoding objects for classes in the alternatives category ... 87
23.1.1 The defined syntax ... 87
23.1.2 Purpose and restrictions ... 88
23.1.3 Encoder actions ... 88
23.1.4 Decoder actions ... 88
23.2 Defining encoding objects for classes in the bitstring category ... 88
23.2.1 The defined syntax ... 88
23.2.2 Model for the encoding of classes in the bitstring category ... 89
23.2.3 Purpose and restrictions ... 90
23.2.4 Encoder actions ... 90
23.2.5 Decoder actions ... 91
23.3 Defining encoding objects for classes in the boolean category ... 91
23.3.1 The defined syntax ... 91
23.3.2 Purpose and restrictions ... 92
23.3.3 Encoder actions ... 93
23.3.4 Decoder actions ... 93
23.4 Defining encoding objects for classes in the characterstring category ... 93
23.4.1 The defined syntax ... 93
23.4.2 Model for the encoding of classes in the characterstring category ... 94
23.4.3 Purpose and restrictions ... 94
23.4.4 Encoder actions ... 95
23.4.5 Decoder actions ... 95
23.5 Defining encoding objects for classes in the concatenation category ... 95
23.5.1 The defined syntax ... 95
23.5.2 Purpose and restrictions ... 97
23.5.3 Encoder actions ... 97
23.5.4 Decoder actions ... 98
23.6 Defining encoding objects for classes in the integer category ... 98
23.6.1 The defined syntax ... 98
23.6.2 Purpose and restrictions ... 98
23.6.3 Encoder actions ... 98
23.6.4 Decoder actions ... 98
23.7 Defining encoding objects for the `#CONDITIONAL-INT` class ... 98
23.7.1 The defined syntax ... 98
23.7.2 Purpose and restrictions ... 100
23.7.3 Encoder actions ... 101
23.7.4 Decoder actions ... 102
23.8 Defining encoding objects for classes in the null category ... 102
23.8.1 The defined syntax ... 102
23.8.2 Purpose and restrictions ... 103
23.8.3 Encoder actions ... 103
23.8.4 Decoder actions ... 104
23.9 Defining encoding objects for classes in the octetstring category ... 104
23.9.1 The defined syntax ... 104
23.9.2 Model for the encoding of classes in the octetstring category ... 105
23.9.3 Purpose and restrictions ... 105
23.9.4 Encoder actions ... 106
23.9.5 Decoder actions ... 106
23.10 Defining encoding objects for classes in the open type category ... 106
23.10.1 The defined syntax ... 106
23.10.2 Model for the encoding of classes in the open type category ... 107
23.10.3 Purpose and restrictions ... 108
23.10.4 Encoder actions ... 108
23.10.5 Decoder actions ... 108

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

23.11 Defining encoding objects for classes in the optionality category ... 109
23.11.1 The defined syntax ... 109
23.11.2 Purpose and restrictions ... 109
23.11.3 Encoder actions ... 110
23.11.4 Decoder actions ... 110
23.12 Defining encoding objects for classes in the pad category ... 110
23.12.1 The defined syntax ... 110
23.12.2 Purpose and restrictions ... 111
23.12.3 Encoder actions ... 111
23.12.4 Decoder actions ... 111
23.13 Defining encoding objects for classes in the repetition category ... 112
23.13.1 The defined syntax ... 112
23.13.2 Purpose and restrictions ... 112
23.13.3 Encoder actions ... 112
23.13.4 Decoder actions ... 112
23.14 Defining encoding objects for the `#CONDITIONAL-REPETITION` class ... 112
23.14.1 The defined syntax ... 112
23.14.2 Purpose and restrictions ... 114
23.14.3 Encoder actions ... 114
23.14.4 Decoder actions ... 115
23.15 Defining encoding objects for classes in the tag category ... 115
23.15.1 The defined syntax ... 115
23.15.2 Purpose and restrictions ... 116
23.15.3 Encoder actions ... 117
23.15.4 Decoder actions ... 117
23.16 Defining encoding objects for classes in the other categories ... 117

24 Defined syntax specification for the `#TRANSFORM` encoding class ... 118
24.1 Summary of encoding properties and defined syntax ... 118
24.2 Source and target of transforms ... 120
24.3 The int-to-int transform ... 121
24.4 The bool-to-bool transform ... 122
24.5 The bool-to-int transform ... 122
24.6 The int-to-bool transform ... 123
24.7 The int-to-chars transform ... 123
24.8 The int-to-bits transform ... 124
24.9 The bits-to-int transform ... 125
24.10 The char-to-bits transform ... 125
24.11 The bits-to-char transform ... 127
24.12 The bit-to-bits transform ... 128
24.13 The bits-to-bits transform ... 128
24.14 The chars-to-composite-char transform ... 129
24.15 The bits-to-composite-bits transform ... 129
24.16 The octets-to-composite-bits transform ... 130
24.17 The composite-char-to-chars transform ... 130
24.18 The composite-bits-to-bits transform ... 130
24.19 The composite-bits-to-octets transform ... 130

25 Complete encodings and the `#OUTER` class ... 131
25.1 Encoding properties, syntax and purpose for the `#OUTER` class ... 131
25.2 Encoder actions for `#OUTER` ... 132
25.3 Decoder actions for `#OUTER` ... 132

Annex A Addendum to Rec. ITU-T X.680 | ISO/IEC 8824-1 ... 133
A.1 Exports and imports clauses ... 133
A.2 Addition of `REFERENCE` ... 133
A.3 Notation for character string values ... 134

Annex B Addendum to Rec. ITU-T X.681 | ISO/IEC 8824-2 ... 135
B.1 Definitions ... 135
B.2 Additional lexical items ... 135

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

B.2.1 Ordered value list field references ... 135
B.2.2 Ordered encoding object list field references ... 135
B.2.3 Encoding class field references ... 135
B.3 Addition of "ENCODING-CLASS" ... 135
B.4 FieldSpec additions ... 135
B.5 Fixed-type ordered value list field spec ... 136
B.6 Fixed-class encoding object field spec ... 136
B.7 Variable-class encoding object field spec ... 136
B.8 Fixed-class encoding object set field spec ... 137
B.9 Fixed-class ordered encoding object list field spec ... 137
B.10 Encoding class field spec ... 137
B.11 Ordered value list notation ... 137
B.12 Ordered encoding object list notation ... 137
B.13 Primitive field names ... 138
B.14 Additional reserved words ... 138
B.15 Definition of encoding objects ... 139
B.16 Additions to "Setting" ... 139
B.17 Encoding class field type ... 139
Annex C Addendum to Rec. ITU-T X.683 | ISO/IEC 8824-4 ... 141
C.1 Parameterized assignments ... 141
C.2 Parameterized encoding assignments ... 141
C.3 Referencing parameterized definitions ... 142
C.4 Actual parameter list ... 142
Annex D Examples ... 144
D.1 General examples ... 144
D.1.1 An encoding object for a boolean type ... 144
D.1.2 An encoding object for an integer type ... 145
D.1.3 Another encoding object for an integer type ... 145
D.1.4 An encoding object for an integer type with holes ... 145
D.1.5 A more complex encoding object for an integer type ... 145
D.1.6 Positive integers encoded in BCD ... 146
D.1.7 An encoding object of class `#BITS` ... 147
D.1.8 An encoding object for an octetstring type ... 148
D.1.9 An encoding object for a character string type ... 148
D.1.10 Mapping character values to bit values ... 148
D.1.11 An encoding object for a sequence type ... 149
D.1.12 An encoding object for a choice type ... 149
D.1.13 Encoding a bitstring containing another encoding ... 150
D.1.14 An encoding object set ... 150
D.1.15 ASN.1 definitions ... 151
D.1.16 EDM definitions ... 151
D.1.17 ELM definitions ... 151
D.2 Specialization examples ... 152
D.2.1 Encoding by distributing values to an alternative encoding structure ... 152
D.2.2 Encoding by mapping ordered abstract values to an alternative encoding structure ... 152
D.2.3 Compression of non-continuous value ranges ... 153
D.2.4 Compression of non-continuous value ranges using a transform ... 153
D.2.5 Compression of an unevenly distributed value set by mapping ordered abstract values ... 154
D.2.6 Presence of an optional component depending on the value of another component ... 154
D.2.7 The presence of an optional component depends on some external condition ... 155
D.2.8 A variable length list ... 155
D.2.9 Equal length lists ... 156
D.2.10 Uneven choice alternative probabilities ... 157
D.2.11 A version 1 message ... 158
D.2.12 The encoding object set ... 159
D.2.13 ASN.1 definitions ... 159
D.2.14 EDM definitions ... 160
D.2.15 ELM definitions ... 160
D.3 Explicitly generated structure examples ... 160

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

D.3.1 Sequence with optional components defined by a pointer ... 161
D.3.2 Addition of a boolean type as a presence determinant ... 161
D.3.3 Sequence with optional components identified by a unique tag and delimited by a length field ... 163
D.3.4 Sequence-of type with a count ... 164
D.3.5 Encoding object sets ... 164
D.3.6 ASN.1 definitions ... 165
D.3.7 EDM definitions ... 165
D.3.8 ELM definitions ... 165
D.4 A more-bit encoding example ... 166
D.4.1 Description of the problem ... 166
D.4.2 Use of ASN.1 to provide the more-bit determinant ... 166
D.4.3 Use of value mappings to provide the more-bit determinant ... 167
D.4.4 Use of the replacement mechanism to provide the more-bit determinant ... 168
D.5 Legacy protocol specified with tabular notation ... 168
D.5.1 Introduction ... 168
D.5.2 Encoding definition for the top-level message structure ... 170
D.5.3 Encoding definition for a message structure ... 170
D.5.4 Encoding for the sequence type "B" ... 171
D.5.5 Encoding for an octet-aligned sequence-of type with a length determinant ... 171
D.5.6 Encoding for an octet-aligned sequence-of type which continues to the end of the PDU ... 171
D.5.7 EDM definitions ... 171
D.5.8 ELM definitions ... 172
Annex E Support for Huffman encodings ... 173
Annex F Additional information on the Encoding Control Notation (ECN) ... 175
Annex G Summary of the ECN notation ... 176

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

Introduction

The Encoding Control Notation (ECN) is a notation for specifying encodings of ASN.1 types that differ from those provided by standardized encoding rules. ECN can be used to encode all types of an ASN.1 specification, but can also be used with standardized encoding rules such as BER or PER (Rec. ITU-T X.690 | ISO/IEC 8825-1 and Rec. ITU-T X.691 | ISO/IEC 8825-2) to specify only the encoding of types that have special requirements.

An ASN.1 type specifies a set of abstract values. Encoding rules specify the representation of these abstract values as a series of bits. ECN is designed to meet the following encoding needs:

a) The need to write ASN.1 types (and get the support of ASN.1 tools in implementations) for established ("legacy") protocols where the encoding is already determined and differs from all standardized encoding rules.

b) The need to produce encodings that are minor variations on standardized rules.

The linkage provided in an ECN specification to an ASN.1 specification is well-defined and machine processable, so encoders and decoders can be automatically generated from the combined specifications. This is a significant factor in reducing both the amount of work and the possibility of errors in making interoperable systems. Another significant advantage is the ability to provide automatic tool support for testing.

These advantages are available with ASN.1 alone when standardized encoding rules suffice, but the ECN work provides these advantages in circumstances where the standardized encoding rules are not sufficient.

NOTE 1 – Currently ECN support only binary-based encodings, but could be extended in the future to cover character-based encodings.

Annex A forms an integral part of this Recommendation | International Standard, and details modifications to be made to Rec. ITU-T X.680 | ISO/IEC 8824-1 to support the notation used in this Recommendation | International Standard.

Annex B forms an integral part of this Recommendation | International Standard, and details modifications to be made to Rec. ITU-T X.681 | ISO/IEC 8824-2 to support the notation used in this Recommendation | International Standard.

Annex C forms an integral part of this Recommendation | International Standard, and details modifications to be made to Rec. ITU-T X.683 | ISO/IEC 8824-4 to support the notation used in this Recommendation | International Standard.

NOTE 2 – It is not intended that Annexes A, B and C be progressed as amendments to the referenced Recommendations | International Standards. The modifications are solely for the purpose of ECN definition (see clause 5 and 9.28).

Annex D does not form an integral part of this Recommendation | International Standard, and contains examples of the use of ECN.

Annex E does not form an integral part of this Recommendation | International Standard and provides more detail on the support for Huffman encodings in ECN.

Annex F does not form an integral part of this Recommendation | International Standard, and identifies a Web site providing access to further information and links relevant to ECN.

Annex G does not form an integral part of this Recommendation | International Standard, and provides a summary of ECN using the notation of clause 5.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – ASN.1 encoding rules: Specification of Encoding Control Notation (ECN)

1 Scope

This Recommendation | International Standard defines a notation for specifying encodings of ASN.1 types or of parts of types.

It provides several mechanisms for such specification, including:

- direct specification of the encoding using standardized notation;
- specification of the encoding by reference to standardized encoding rules;
- specification of the encoding of an ASN.1 type by reference to an encoding structure;
- specification of the encoding using non-ECN notation.

It also provides the means to link the specification of encodings to the type definitions to which they are to be applied.

ECN does not currently provide any support for specifications using the OID internationalized resource identifier type or the relative OID internationalized resource identifier type (see Rec. ITU-T X.680 | ISO/IEC 8824-1), and these are not referred to further in this Standard.

2 Normative references

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

NOTE – This Recommendation | International Standard is based on ISO/IEC 10646:2003. It cannot be applied using later versions of this standard.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.660 (2011) | ISO/IEC 9834-1:2012, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: General procedures and top arcs of the international object identifier tree.*
- Recommendation ITU-T X.680 (2021) | ISO/IEC 8824-1:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*
- Recommendation ITU-T X.681 (2021) | ISO/IEC 8824-2:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification.*
- Recommendation ITU-T X.682 (2021) | ISO/IEC 8824-3:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- Recommendation ITU-T X.683 (2021) | ISO/IEC 8824-4:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.*
- Recommendation ITU-T X.690 (2021) | ISO/IEC 8825-1:2021, *Information technology – ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER).*
- Recommendation ITU-T X.691 (2021) | ISO/IEC 8825-2:2021, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

NOTE – The above references shall be interpreted as references to the identified Recommendations | International Standards together with all their published amendments and technical corrigenda.

2.2 Additional references

– ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*.

NOTE – The above reference shall be interpreted as a reference to ISO/IEC 10646 together with all its published amendments and technical corrigenda.

3 Definitions

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

3.1 ASN.1 definitions

This Recommendation | International Standard uses the terms defined in clause 3 of Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4, Rec. ITU-T X.690 | ISO/IEC 8825-1 and Rec. ITU-T X.691 | ISO/IEC 8825-2.

3.2 ECN-specific definitions

3.2.1 alignment point: The point in an encoding (usually its start) which serves as a reference point when an encoding specification requires alignment to some boundary.

3.2.2 auxiliary field: A field of a replacement structure (that is added in the ECN specification) whose value is set directly by the encoder without the use of any abstract value provided by the application.

NOTE – An example of an auxiliary field is a length determinant for an integer encoding or for a repetition.

3.2.3 bit-field: Contiguous bits or octets in an encoding which are decoded as a whole, and which either represent an abstract value, or provide information (such as a length determinant for some other field – see 3.2.31) needed for successful decoding, or both.

NOTE – It is in legacy protocols that "or both" sometimes occurs.

3.2.4 bit-field class: An encoding class whose objects specify the encoding of abstract values (of some ASN.1 type) into bits.

NOTE – Other encoding classes are concerned with more general encoding procedures, such as those required to determine the end of repetitions of bit-field class encodings, or to determine which of a set of alternative bit-field encodings is present.

3.2.5 bounds condition: A condition on the existence of bounds of an integer field (and whether they allow negative values or not) which, if satisfied, means that specified encoding rules are to be applied.

3.2.6 choice determinant: A bit-field which determines which of several possible encodings (each representing different abstract values) is present in some other bit-field.

3.2.7 combined encoding object set: A temporary set of encoding objects produced by the combination of two sets of encoding objects for the purpose of applying encodings.

3.2.8 conditional encoding: An encoding which is to be applied only if some specified condition is satisfied.

NOTE – The condition may be a bounds condition or a size range condition, or other more complex conditions.

3.2.9 containing type: An ASN.1 type (or encoding structure field) where a contents constraint has been applied to the values of that type (or to the values associated with that encoding structure field).

NOTE – The ASN.1 types to which a contents constraint (using **`CONTAINING/ENCODED BY`**) can be applied are the bitstring and the octetstring types.

3.2.10 current application point: The point in an encoding structure at which a combined encoding object set is being applied.

3.2.11 differential encoding-decoding: The specification of rules for a decoder that require the acceptance of encodings that cannot be produced by an encoder conforming to the current specification.

NOTE – Differential encoding-decoding supports the specification of decoding by a decoder (conforming to an initial version of a standard) which is intended to enable it to successfully decode encodings produced by a later version of that standard. This is sometimes referred to as support for extensibility.

3.2.12 encoding class: The set of all possible encodings for a specific part of the procedures needed to perform the encoding or decoding of an ASN.1 type.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

NOTE – Encoding classes are defined for the encoding of primitive ASN.1 types, but are also defined for the procedures associated with ASN.1 tag notation, the use of **OPTIONAL** and for encoding constructors.

3.2.13 encoding class category: Encoding classes with some common characteristics.

NOTE – Examples are the integer category, the boolean category, and the concatenation category.

3.2.14 encoding constructor: An encoding class whose encoding objects define procedures for combining, selecting, or repeating parts of an encoding. (Examples are the **#ALTERNATIVES**, **#CHOICE**, **#CONCATENATION**, **#SEQUENCE**, etc. classes.)

3.2.15 Encoding Definition Modules (EDM): Modules that define encodings for application in the Encoding Link Module.

3.2.16 Encoding Link Module (ELM): The (unique, for any given application) module that assigns encodings to ASN.1 types.

3.2.17 encoding object: The specification of some part of the procedures needed to perform the encoding or decoding of an ASN.1 type.

NOTE – Encoding objects can specify the encoding of primitive ASN.1 types, but can also specify the procedures associated with ASN.1 tag notation, the use of **OPTIONAL** and with encoding constructors.

3.2.18 encoding object set: A set of encoding objects.

NOTE – An encoding object set is normally used in the Encoding Link Module to determine the encoding of all the top-level types used in an application.

3.2.19 encoding property: A piece of information used to define an encoding using the notation specified in clauses 23, 24 and 25.

3.2.20 encoding space: The number of bits (or octets, words or other units) used to encode an abstract value into a bit-field (see 9.21.5).

3.2.21 encoding structure: The structure of an encoding, defined either from the structure of an ASN.1 type definition, or in an EDM using bit-field classes and encoding constructors.

NOTE 1 – Use of an encoding structure is only one of several mechanisms (but an important one) that the Encoding Control Notation provides for the definition of encodings for ASN.1 types.

NOTE 2 – Definition of an encoding structure is also the definition of a corresponding encoding class.

3.2.22 explicitly generated encoding structure: An encoding structure derived from an implicitly generated encoding structure by use of the renames clause in an EDM.

3.2.23 extensibility: Provisions in an early version of a standard that are designed to maximize the interworking of implementations of that early version with the expected implementations of a later version of that standard.

3.2.24 fully-qualified name: A reference to an encoding class, object, or object set that includes either the name of the EDM module in which that encoding class, object, or object set was defined, or (in the case of an implicitly generated encoding class) the name of the ASN.1 module in which it was generated. (See also 3.2.43.)

NOTE – A fully-qualified name (see production "ExternalEncodingClassReference" in 10.6) has to be used in the body of a module if the encoding class is an implicitly generated encoding structure whose name is the same as a reserved class name, or if use of the name alone would produce ambiguity due to multiple imports of classes with that name. (See A.1/13.16).

3.2.25 generated encoding structure: An implicitly or explicitly generated encoding structure whose purpose is to define the encodings of the corresponding ASN.1 type through application of encodings in the ELM.

3.2.26 governor: A part of an ECN specification which determines the syntactic form (and semantics) of some other part of the ECN specification.

NOTE – A governor is an encoding class reference, and it determines the syntax to be used for the definition of an encoding object (of that class). The concept is the same as the concept of a type reference in ASN.1 acting as the governor for ASN.1 value notation.

3.2.27 handle value set: The specified set of all possible values of the identification handle that is exhibited by an encoding object.

3.2.28 identification handle: Part of an encoding which serves to distinguish the encodings produced by one encoding object (of a given class) from those produced by other encoding objects (of other classes).

NOTE – The ASN.1 Basic Encoding Rules use tags to provide identification handles in BER encodings.

3.2.29 implicitly generated encoding structure: The encoding structure that is implicitly generated and exported whenever a type is defined in an ASN.1 module.

3.2.30 initial application point: The point in an encoding structure at which any given combined encoding object set is first applied (in the ELM and in EDMs) .

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

3.2.31 length determinant: A bit-field that determines the length of some other bit-field.

3.2.32 negative integer value: A value less than zero.

3.2.33 non-negative integer value: A value greater than or equal to zero.

3.2.34 non-positive integer value: A value less than or equal to zero.

3.2.35 optional bit-field: A bit-field that is sometimes included (to encode an abstract value) and is sometimes omitted.

3.2.36 positive integer value: A value greater than zero.

3.2.37 presence determinant: A bit-field that determines whether an optional bit-field is present or not.

3.2.38 primitive class: An encoding class which is not an encoding structure, and which cannot be de-referenced to some other class (see 16.1.14).

3.2.39 recursive definition (of a reference name): A reference name for which resolution of the reference name, or of the governor of the definition of the reference name, requires resolution of the original reference name.

NOTE – Recursive definition of an encoding class (including an encoding structure) or an encoding object is permitted (but see 17.1.4). Recursive definition of an encoding object set is forbidden by 18.1.3.

3.2.40 recursive instantiation (of a parameterized reference name): An instantiation of a reference name, where resolution of the actual parameters requires resolution of the original reference name.

NOTE – Recursive instantiation of an encoding class (including an encoding structure) or an encoding object is permitted (but see 17.1.4). Recursive instantiation of an encoding object set is forbidden by 18.1.3.

3.2.41 replacement structure: A parameterized structure used to replace some or all parts of a construction before encoding the construction.

3.2.42 self-delimiting encoding: An encoding for a set of abstract values such that there is no abstract value that has an encoding that is an initial sub-string of the encoding of any other abstract value in the set.

NOTE – This includes not only fixed-length encodings of a bounded integer, but also encodings generally described as "Huffman encodings" (see Annex E).

3.2.43 simple reference name: A reference to an encoding class, object, or object set that includes neither the name of the EDM module in which that encoding class, object, or object set was defined, nor (in the case of an implicitly generated encoding class) the name of the ASN.1 module in which it was generated.

NOTE – A simple reference name can only be used when the reference to the encoding class is unambiguous, otherwise a fully-qualified name (see 3.2.24) has to be used in the body of a module.

3.2.44 size range condition: A condition on the existence of effective size constraints on a string or repetition field (and whether the constraint includes zero, and/or allows multiple sizes) which, if satisfied, means that specified encoding rules are to be applied.

3.2.45 source governor (or source class): The governor that determines the notation for specifying abstract values associated with a source class when mapping them to a target class.

3.2.46 start pointer: An auxiliary field indicating the presence or absence of an optional bit-field, and in the case of presence, containing the offset from the current position to the bit-field.

3.2.47 target governor (or target class): The governor that determines the notation for specifying abstract values associated with a target class when mapping to them from a source class.

3.2.48 top-level type(s): Those ASN.1 type(s) in an application that are used by the application in ways other than to define the components of other ASN.1 types.

NOTE 1 – Top-level types may also be used (but usually are not) as components of other ASN.1 types.

NOTE 2 – Top-level types are sometimes referred to as "the application's messages", or "PDUs". Such types are normally treated specially by tools, as they form the top-level of programming language data-structures that are presented to the application.

3.2.49 transforms: Encoding objects of the class **`#TRANSFORM`** which specify that the encoding of the abstract values associated with some class (or of transform composites – see 3.2.50) is to be the encoding of different abstract values associated with the same or a different class (or of transform composites).

NOTE – Transforms can be used, for example, to specify simple arithmetic operations on integer values, or to map integer values into characterstrings or bitstrings.

3.2.50 transform composites: An ordered list of elements that can itself be the source or the result of transforms.

NOTE – All the elements of a composite are required to have the same classification (see 9.18.2).

3.2.51 value encoding: The way in which an encoding space is used to represent an abstract value (see 9.21.5).

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

4 Abbreviations

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

ASN.1	Abstract Syntax Notation One
BCD	Binary Coded Decimal
BER	Basic Encoding Rules of ASN.1
CER	Canonical Encoding Rules of ASN.1
DER	Distinguished Encoding Rules of ASN.1
ECN	Encoding Control Notation for ASN.1
EDM	Encoding Definition Module
ELM	Encoding Link Module
PDU	Protocol Data Unit
PER	Packed Encoding Rules of ASN.1

5 Definition of ECN syntax

5.1 This Recommendation | International Standard employs the notational convention defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 5.

5.2 This Recommendation | International Standard employs the notation for information object classes defined in Rec. ITU-T X.681 | ISO/IEC 8824-2 as modified by Annex B.

5.3 This Recommendation | International Standard references productions defined in Rec. ITU-T X.680 | ISO/IEC 8824-1 as modified by Annex A, Rec. ITU-T X.681 | ISO/IEC 8824-2 as modified by Annex B, and Rec. ITU-T X.683 | ISO/IEC 8824-4 as modified by Annex C.

6 Encoding conventions and notation

6.1 This Recommendation | International Standard defines the value of each octet in an encoding by use of the terms "most significant bit" and "least significant bit".

NOTE – Lower layer specifications use the same notation to define the order of bit transmission on a serial line, or the assignment of bits to parallel channels.

6.2 For the purpose of this Recommendation | International Standard, the bits of an octet are numbered from 8 to 1, where bit 8 is the "most significant bit" and bit 1 is the "least significant bit".

6.3 For the purposes of this Recommendation | International Standard, encodings are defined as a string of bits starting from a "leading bit" through to a "trailing bit". On transmission, the first eight bits of this string of bits starting with the "leading bit" shall be placed in the first transmitted octet with the leading bit as the most significant bit of that octet. The next eight bits shall be placed in the next octet, and so on. If the encoding is not a multiple of eight bits, then the remaining bits shall be transmitted as if they were bits 8 downwards of a subsequent octet.

NOTE – A complete ECN encoding is not necessarily always a multiple of eight bits, but an ECN specification can determine the addition of padding to ensure this property.

6.4 When figures are shown in this Recommendation | International Standard, the "leading bit" is always shown on the left of the figure.

7 The ECN character set

7.1 Use of the term "character" throughout this Recommendation | International Standard refers to the characters specified in ISO/IEC 10646, and full support for all possible ECN specifications can require the representation of all these characters.

7.2 With the exception of comment (as defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 12.6), non-ECN definition of encoding objects (see 17.8) and character string values, ECN specifications use only the characters listed in Table 1.

7.3 Lexical items defined in clause 8 consist of a sequence of the characters listed in Table 1.

NOTE – Additional restrictions on the permitted characters for each lexical item are specified in clause 8.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

Table 1 – ECN characters

0 to 9	(DIGIT ZERO to DIGIT 9)
A to Z	(LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z)
a to z	(LATIN SMALL LETTER A to LATIN SMALL LETTER Z)
"	(QUOTATION MARK)
#	(NUMBER SIGN)
&	(AMPERSAND)
'	(APOSTROPHE)
(	(LEFT PARENTHESIS)
)	(RIGHT PARENTHESIS)
,	(COMMA)
-	(HYPHEN-MINUS)
.	(FULL STOP)
:	(COLON)
;	(SEMICOLON)
<	LESS-THAN SIGN
=	(EQUALS SIGN)
>	GREATER-THAN SIGN
{	(LEFT CURLY BRACKET)
|	(VERTICAL LINE)
}	(RIGHT CURLY BRACKET)

7.4 There shall be no significance placed on the typographical style, size, colour, intensity, or other display characteristics.

7.5 The upper-case and lower-case letters shall be regarded as distinct.

8 ECN lexical items

In addition to the ASN.1 lexical items specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12, this Recommendation | International Standard uses lexical items specified in the following subclauses. The general rules specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 12.1, apply in this clause.

NOTE – Annex G lists all lexical items and all the productions used in this Recommendation | International Standard, identifying those that are defined in Rec. ITU-TX.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2 and Rec. ITU-T X.683 | ISO/IEC 8824-4.

8.1 Encoding object references

Name of item – encodingobjectreference

An "encodingobjectreference" shall consist of the sequence of characters specified for a "valuereference" in Rec. ITU-T X.680 | ISO/IEC 8824-1, 12.4. In analyzing an instance of use of this notation, an "encodingobjectreference" is distinguished from an "identifier" by the context in which it appears.

8.2 Encoding object set references

Name of item - encodingobjectsetreference

An "encodingobjectsetreference" shall consist of the sequence of characters specified for a "typereference" in Rec. ITU-T X.680 | ISO/IEC 8824-1, 12.2. It shall not be one of the character sequences listed in 8.4.

8.3 Encoding class references

Name of item – encodingclassreference

An "encodingclassreference" shall consist of the character "#" followed by the sequence of characters specified for a "typereference" in Rec. ITU-T X.680 | ISO/IEC 8824-1, 12.2. It shall not be one of the character sequences listed in 8.5

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

except in an EDM imports list (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.20, as modified by A.1) or in an "ExternalEncodingClassReference" (see the Note in 14.11).

8.4 Reserved word items

Names of reserved word items:

```
ALL                          FIELDS                       PER-BASIC-UNALIGNED
AS                           FROM                         PER-CANONICAL-ALIGNED
BEGIN                        GENERATES                    PER-CANONICAL-UNALIGNED
BER                          IF                           PLUS-INFINITY
BITS                         IMPORTS                      REFERENCE
BY                           IN                           REMAINDER
CER                          LINK-DEFINITIONS             RENAMES
COMPLETED                    MAPPING                      SIZE
DECODE                       MAX                          STRUCTURE
DER                          MIN                          STRUCTURED
DISTRIBUTION                 MINUS-INFINITY               TO
ENCODE                       NON-ECN-BEGIN                TRANSFORMS
ENCODING-CLASS               NON-ECN-END                  TRUE
ENCODE-DECODE                NULL                         UNION
ENCODING-DEFINITIONS         OPTIONAL-ENCODING            USE
END                          OPTIONS                      USE-SET
EXCEPT                       ORDERED                      VALUES
EXPORTS                      OUTER                        WITH
FALSE                        PER-BASIC-ALIGNED
```

Items with the above names shall consist of the sequence of characters in the name.

NOTE – The words (see Rec. ITU-T X.681 | ISO/IEC 8824-2, 7.9) used in the definition of encoding classes (within a "WITH SYNTAX" statement) in clause 23 are not reserved words (see also B.14).

8.5 Reserved encoding class name items

Names of reserved encoding class name items:

```
#ALTERNATIVES                #EXTERNAL                    #PrintableString
#BITS                        #GeneralizedTime              #REAL
#BIT-STRING                  #GeneralString                #RELATIVE-OID
#BMPString                   #GraphicString                #REPETITION
#BOOL                        #IA5String                    #SEQUENCE
#BOOLEAN                     #INT                          #SEQUENCE-OF
#CHARACTER-STRING            #INTEGER                      #SET
#CHARS                       #NUL                          #SET-OF
#CHOICE                      #NULL                         #TAG
#CONCATENATION               #NumericString                #TeletexString
#CONDITIONAL-INT             #OBJECT-IDENTIFIER            #TIME
#CONDITIONAL-REPETITION      #ObjectDescriptor             #TIME-OF-DAY
#DATE                        #OCTETS                       #TRANSFORM
#DATE-TIME                   #OCTET-STRING                 #UniversalString
#DURATION                    #OPEN-TYPE                    #UTCTime
#EMBEDDED-PDV                #OPTIONAL                     #UTF8String
#ENCODINGS                   #OUTER                        #VideotexString
#ENUMERATED                  #PAD                          #VisibleString
```

Items with the above names shall consist of the sequence of characters in the name.

8.6 Non-ECN item

Name of item – anystringexceptnonecnend

An "anystringexceptnonecnend" shall consist of one or more characters from the ISO/IEC 10646 character set, except that it shall not be the character sequence **NON-ECN-END** nor shall that character sequence appear within it.

9 ECN Concepts

This clause describes the main concepts underlying this ITU-T Recommendation | International Standard.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.1 Encoding Control Notation (ECN) specifications

9.1.1 ECN specifications consist of one or more Encoding Definition Modules (EDMs) which define encoding rules for ASN.1 types, and a single Encoding Link Module (ELM) that applies those encoding rules to ASN.1 types.

9.1.2 The most important part of ECN is the concept of an **encoding structure definition**. ASN.1 is used to define complex abstract values using primitive types and constructors. In the same way, complex encodings can be defined using a similar notation where construction mechanisms are used to combine simple bit-fields into more complex encodings, and eventually into complete messages. This is called encoding structure definition. In using ECN with ASN.1, it is necessary in principle to:

a) define the abstract syntax (the set of abstract values to be communicated, and their semantics); and

b) the encoding structure (the structure of fields) used to carry these abstract values; and

c) to relate the components of the abstract value to the encoding structure fields; and

d) to define the encoding of each encoding structure field and mechanisms for identifying repetitions of fields and identification of alternatives, etc.

9.1.3 The above process normally takes part in several stages. First an ASN.1 definition is produced detailing the abstract syntax. From this a crude encoding structure is automatically generated (conceptually within the ASN.1 module). This implicitly generated structure contains only fields that carry the application semantics, without fields for things like length determination, alternative selection, and so on.

9.1.4 This structure can be transformed by a series of mechanisms into the structure of fields that is actually required, including all fields needed to support the decoding activity (determinants). These mechanisms all involve some form of replacement of a simple field carrying application semantics by a more complex structure. Such replacements form an important part of ECN specification.

9.1.5 We can further define **encoding objects** for each of the fields in the final structure. These determine not only the encoding of fields, but also the way in which one field determines the length (for example) of another, or has its optionality resolved.

9.1.6 The above definitions occur in Encoding Definition Modules (EDMs). The last step is to apply a set of defined encoding objects to the final encoding structure in order to completely determine an encoding. This is done in the Encoding Link Module (ELM).

9.2 Encoding classes

9.2.1 An encoding class is an implicit property of all ASN.1 types, and represents the set of all possible encoding specifications for that type. It provides a reference that allows Encoding Definition Modules to define encoding rules for encoding structure fields corresponding to the type. Encoding class names begin with the character "**#**".

Example: Encoding rules for the ASN.1 built-in type **INTEGER** are defined by reference to the encoding class **#INTEGER**, and encoding rules for a user-defined type "**My-Type**" are defined by reference to the encoding class **#My-Type**.

9.2.2 There are several kinds of encoding classes:

9.2.2.1 Built-in encoding classes – There are built-in encoding classes with names such as **#INTEGER** and **#BOOLEAN**. These enable the definition of special encodings for primitive ASN.1 types. There are also built-in encoding classes for encoding constructors such as **#SEQUENCE**, **#SEQUENCE-OF** and **#CHOICE** (see also 9.3.2), and for the definition of encoding rules for handling optionality through **#OPTIONAL**. Encoding of tags is supported by the **#TAG** class. Finally, there are some built-in classes (**#OUTER**, **#TRANSFORM** and others) that allow the definition of encoding procedures which are part of the encoding/decoding process, but which do not directly relate to any actual bit-field or ASN.1 construct.

9.2.2.2 Encoding classes for implicitly generated encoding structures – These have names consisting of the character "**#**" followed by the "typereference" appearing in a "TypeAssignment" in an ASN.1 module. Such encoding classes are implicitly generated whenever a (non-parameterized) "typereference" is assigned in an ASN.1 module, and can be imported into an Encoding Definition Module to enable the definition of special encodings for the corresponding ASN.1 type. These encoding classes represent the structure of an ASN.1 encoding, and are formed from the built-in encoding classes mirroring the structure of the ASN.1 type definition.

9.2.2.3 Encoding classes for user-defined encoding structures – These are encoding classes defined by the ECN user by specifying an encoding structure (see 9.3) as a structure made up of bit-fields and encoding constructors. These encoding structures are similar to the implicitly generated encoding structures, but the ECN user has full control of their structure. These classes enable complex encoding rules to be defined, and are important for the use of ASN.1 with ECN for specifying legacy protocols, where additional bit-fields are needed in the encoding for determinants.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.2.2.4 Encoding classes for explicitly generated encoding structures – These are encoding classes produced from an implicitly generated encoding structure by selectively changing the names of certain classes in order to indicate places where specialized encodings are needed for optionality, sequence-of termination, etc.

9.3 Encoding structures

9.3.1 Encoding structure definitions have some similarity to ASN.1 type definitions, and have a name beginning with the character "**#**", then an upper-case letter. Each encoding structure definition defines a new encoding class (the set of all possible encodings of that encoding structure). Encoding structures are formed from fields which are either built-in encoding classes or the names of other encoding structures, combined using encoding constructors (which represent the set of all possible encoding rules that support their type of construction mechanism, and are hence called encoding classes). (See D.2.8.4 for an example of an encoding structure definition.)

9.3.2 The most basic encoding constructors are **#CONCATENATION**, **#REPETITION**, and **#ALTERNATIVES**, corresponding roughly to ASN.1 sequence (and set), sequence-of (and set-of), and choice types. There is also an encoding class **#OPTIONAL** that represents the optional presence of encodings, corresponding roughly to ASN.1 **DEFAULT** and **OPTIONAL** markers.

9.3.3 An encoding structure definition defines a structure-based encoding class. Such classes cannot have the same names as encoding classes that are imported into the module. (See Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.13, as modified by A.1 of this Recommendation | International Standard).

9.3.4 Encoding structure names can be exported and imported between Encoding Definition Modules and can be used whenever an encoding class name in the bit-field group of categories (see 9.6) is required.

9.3.5 Values of ASN.1 types (primitive or user-defined) can be mapped to fields of an encoding structure, and encoding rules for that structure then provide encodings of the ASN.1 type. (Values mapped to encoding structures can be further mapped to fields of more complex encoding structures.) This provides a very powerful mechanism for defining complex encoding rules.

9.4 Encoding objects

9.4.1 Encoding objects represent the specific definition of encoding rules for a given encoding class. Usually the rules relate to the actual bits to be produced, but can also specify procedures related to encoding and decoding, for example the way in which the presence or absence of optional components is determined.

9.4.2 In order to fully define the encoding of ASN.1 types (typically the top-level type(s) of an application), it is necessary to define (or obtain from standardized encoding rules) encoding objects for all the classes that correspond to components of those ASN.1 types and for the encoding constructors that are used.

9.4.3 For legacy protocols, this may have to be done by defining a separate encoding object for every component of an ASN.1 type, but it is more commonly possible to use encoding objects defined by standardized encoding rules (such as PER).

9.4.4 Although BER and PER encoding specifications pre-date ECN, within the ECN model they simply define encoding objects for all classes corresponding to the ASN.1 primitive types and constructors (that is, for all the built-in encoding classes). BER and PER are also considered to provide encoding objects for encoding classes used in the definition of encoding structures (see 18.2).

9.5 Encoding object sets

9.5.1 Encoding objects can be grouped into sets in the same way as information objects in ASN.1, and it is these sets of encoding objects that are (in an ELM) applied to an ASN.1 type to determine its encoding. The governor used when forming these encoding object sets is the reserved word **#ENCODINGS**. (See D.1.14 for an example.)

9.5.2 A fundamental rule of encoding object set construction is that any set can contain only one encoding object of a given encoding class (see also 9.6.2). Thus there is no ambiguity when an encoding object set is applied to a type to define its encoding.

9.5.3 There are built-in encoding object sets for all the variants of BER and PER, and these can be used to complete sets of user-defined encoding objects.

9.6 Defining new encoding classes

9.6.1 Those familiar with ASN.1 will be aware that a type assignment can be used to create new names (new types) from, for example, the types **INTEGER** or **BOOLEAN**. The new names identify types that are the same as **INTEGER** or

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

BOOLEAN, but carry different semantics. This concept is extended in ECN to allow the creation (in a class assignment – see 16.1.1) of new names (new classes) for constructors such as **#SEQUENCE**. The new names identify classes that perform a similar function in structuring encodings (for example, concatenation), but which are to have different encoding objects applied to them. A new class name assigned for an old class retains certain characteristics of that old class. So an assignment such as "**#My-Sequence ::= #SEQUENCE**" creates the new class name **#My-Sequence** which is still an encoding class concerned with the concatenation of components. We say that such encoding classes are in the same category.

9.6.2 If a new encoding class is created from an existing encoding class, encoding objects of both the old encoding class and the new encoding class can appear in an encoding object set.

9.6.3 All built-in encoding classes are derived from one of a small number of primitive encoding classes. Thus **#SEQUENCE** and **#SET** are both derived from the **#CONCATENATION** class, **#INTEGER** and **#ENUMERATED** are both derived from the **#INT** class, and the classes for the different ASN.1 character string types are all derived from the **#CHARS** class. An encoding structure (for example, one implicitly generated from an ASN.1 type) can contain a mix of different classes all derived from the same primitive class, enabling different encodings to be applied to **#SEQUENCE** and **#SET** (for example).

9.6.4 It is often convenient to put encoding classes into categories, based on the primitive class they are derived from. Thus we say that **#INTEGER**, **#ENUMERATED** and **#INT** (and any class derived from them in a class assignment statement such as "**#My-int ::= #INT**") are in the integer category. There are also groups of categories that contain very different classes that share some characteristic. Thus any class that can have abstract values directly associated with it, and hence which produces bits in an encoding, is said to be in the bit-field group of categories. Thus all classes that are in the integer or the boolean or the characterstring category are in the bit-field group of categories. Classes that are responsible for grouping or repeating encodings (for example classes in the alternatives or the repetition category) are in the encoding constructor group of categories. There are also two classes whose encoding objects define procedures not directly related to constructing an encoding (**#TRANSFORM** and **#OUTER**): these are described as being in the encoding procedure group of categories. Encoding structures are defined using classes in the bit-field group of categories that are combined using classes in the encoding constructor group of categories, together with classes in the optionality (representing encoding procedures for resolving optionality) and tag (representing encoding of tags) categories. All such classes are in the encoding structure category (and also in the bit-field group of categories).

9.6.5 For the primitive classes, the category is directly assigned. For classes created in an encoding class assignment statement, the category is determined by the notation to the right of the "**::=**" symbol. If that notation is an encoding structure definition, then the class is in both the encoding structure category and in the bit-field group of categories. If the notation is a simple class reference name, then the category of the new class is the same as the category of the class being assigned.

9.6.6 The categories of encoding class (see 16.1.3) are:

- The alternatives category (classes that are derived by class assignment from **#ALTERNATIVES**).
- The concatenation category (classes that are derived by class assignment from **#CONCATENATION**).
- The repetition category (classes that are derived by class assignment from **#REPETITION**).
- The optionality category (classes that are derived by class assignment from **#OPTIONAL**).
- The tag category (classes that are derived by class assignment from **#TAG**).
- The boolean, bitstring, characterstring, integer, null, objectidentifier, octetstring, opentype, pad, real, and time categories (categories for classes that are derived from the corresponding primitive classes).
- The encoding structure category (classes generated from ASN.1 type definitions, or by explicit definition of an encoding structure).

9.6.7 The following groups of categories are defined:

- The bit-field group of categories (classes that correspond to actual fields in an encoding such as those in the integer or boolean categories, together with any class in the encoding structure category). Classes in this group of categories are also referred to as bit-field classes.
- The encoding constructor group of categories (classes that are in the alternatives, concatenation, or repetition categories). Classes in this group of categories are also referred to as encoding constructor classes.
- The encoding procedure group of categories (classes not directly related to ASN.1 constructs, and which cannot be assigned new names – **#OUTER**, **#TRANSFORM**, **#CONDITIONAL-INT**, **#CONDITIONAL-REPETITION**). Classes in this group of categories are also referred to as encoding procedure classes.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.7 Defining encoding objects

There are eight mechanisms available for defining an encoding object of a given encoding class. They are not all available for all encoding classes.

9.7.1 The first is to specify it as the same as some other defined encoding object of the required class. This does nothing more than provide a synonym for encoding objects.

9.7.2 The second, available for a restricted set of encoding classes, is to use a defined syntax (see 17.2) to specify the information needed to define an encoding object of that class. Much of the information needed is common to all encoding classes, but some of the information always depends on the specific encoding class. (See D.1.1.2 for an example of defining an encoding object of class **#BOOLEAN** which contains encodings for the ASN.1 type boolean.)

9.7.3 The third, available for all encoding classes, is to define an encoding object as the encoding of the required class which is contained in some existing encoding object set. This is mainly of use in naming an encoding object for a particular class that will perform BER or PER encodings for that class.

NOTE – This can often be useful, but requires knowledge of the encodings produced by standardized encoding rules.

9.7.4 The fourth is to map the abstract values associated with an encoding class ("**#A**", say) to abstract values associated with another (typically more complex) encoding class ("**#B**", say), and to define an encoding object for "**#B**" (using any of the available mechanisms). An encoding object for the abstract values associated with "**#A**" can now be defined as the application to the corresponding abstract values associated with "**#B**" of the encoding object for "**#B**". (See D.2.8.3 for an example.) There are many variants of this (see 9.17).

NOTE – This is the model underlying the definition of an object for encoding an integer type in BER. The integer is mapped to an encoding structure that contains a tag class (**UNIVERSAL**, **APPLICATION**, **PRIVATE**, or context-specific) field, a primitive/constructor boolean, a tag number field, and a value part that encodes the abstract values of the original integer.

9.7.5 The fifth mechanism is to define an encoding object for a class (for example, one corresponding to a user-defined ASN.1 type) by separately defining encoding objects for the components and for the encoding constructor used in defining the encoding class.

9.7.6 The sixth is to define an encoding object for differential encoding-decoding (see 9.8), using two separate encoding objects, one of which defines the encoder's behaviour, and the other of which tells a decoder what encoding should be assumed.

NOTE – An example would be to encode a field which is "reserved for future use" as all zeros, but to accept any value when decoding.

9.7.7 The seventh is to define an encoding options encoding object, which contains an ordered list of encoding objects of the same class. It is an encoder's choice which encoding object from the list is to be applied, subject to the restriction that if only one encoding option can encode a given abstract value, that shall be used, and to the recommendation that the first available encoding in the list should be used.

NOTE – An encoding options encoding object could, for example, be used in the specification of short-form length encodings where these can encode a particular string length, using long-form length encodings where the short-form cannot be used. There is no current mechanism for the ECN specifier to require the use of the first available encoding object (if more than one can encode the abstract value), other than by comment.

9.7.8 Finally, an encoding object can be defined using non-ECN notation. This is a facility to allow use of any desired notation (including natural language) to define the encoding object (see D.2.7.3).

NOTE – Non-ECN notation should be used with caution, as tool-support for implementation is generally not possible in this case.

9.8 Differential encoding-decoding

9.8.1 Differential encoding-decoding is the term applied to a specification that requires an implementation to accept (when decoding) bit-patterns that are in addition to those that it is permitted to generate when performing encoding.

9.8.2 Differential encoding-decoding underlies all support for "extensibility" (the ability for an implementation of an earlier version of a standard to have good interworking capability with an implementation of a later version of the standard).

9.8.3 The precise nature of differential encoding-decoding can be quite complex. It normally includes the requirement that a decoder accepts (and silently ignores) padding fields (usually variable length) which later versions of a standard will use for the transfer of information additional to that transferred in the early version communication.

9.8.4 Support for differential encoding-decoding in ECN is provided by syntax that enables the definition of an encoding object (for any class) that encapsulates two encoding objects. Each encoding object defines rules for encoding. The first encoding object defines the rules that an encoder uses. The decoder uses the second encoding object as a specification of the way the encoding was done.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

NOTE – In ECN, the rules a decoder uses (in an early version of a standard) are always expressed by giving the rules for encoding that it should assume its communicating partner is using. The decoding rules are not given as explicit decoding rules. The ECN specifier will ensure that such decoding rules provide any necessary "extensibility".

9.9 Encoders options in encodings

9.9.1 Encoders options in protocols are generally regarded today as something to be avoided, but ECN has to provide support for such options if a protocol designer decides (or has in the past decided) to include them.

9.9.2 When values are being encoded into an encoding space, it is possible to specify that the size of the encoding space (see 9.21.5) is an encoder's option, provided there is some form of length determinant associated with the encoding. (The extent of the encoder's options may be limited by the maximum value that can be encoded in the length determinant.) This provides a detailed level of support for encoder's options.

9.9.3 A more global mechanism is similar to the support for differential encoding-decoding (see 9.8), but in this case an encoding object for a class can be defined as an encoder's choice of any encoding object from an ordered list of defined encoding objects for that class. In addition to specifying the list of possible encodings, it is also necessary to provide the specification of an encoding object for a class in the alternatives category (see 9.6). This encoding object specifies the encodings and procedures needed to enable a decoder to determine which encoding object was used by the encoder.

9.10 Properties of encoding objects

9.10.1 Encoding objects have some general properties. In most cases, they completely define an encoding, but in some cases they are **encoding constructors**, that is, they define only structural aspects of the encoding, requiring encoding objects for the encoding structure's components to complete the definition of an encoding.

9.10.2 Another key feature of an encoding object is that it may require information from the environment where its rules are eventually applied. One aspect of the environment that is fully supported is the presence of bounds in the ASN.1 type definition, provided they are "PER-visible" (see Rec. ITU-T X.691 | ISO/IEC 8825-2, 10.3).

NOTE – A somewhat different (and not standardized) external dependency would be the definition of a non-ECN encoding object for an **`#ALTERNATIVES`** encoding class which determines the selected alternative based on external data such as the channel the message is being sent on.

9.10.3 A third key feature is that an encoding object may exhibit an **identification handle** in its encodings. This is a part (consisting of a fixed set of bit positions) of all the encodings that it produces and distinguishes those encodings from the encodings produced by other encoding objects (of any class) that exhibit the same identification handle. Identification handles have a name and are visible to decoders without knowledge of either the encoding class or the abstract value that was encoded (but with knowledge of the name of the identification handle that is being used). This concept models (and generalizes) the use of tags in BER encodings: the tag value in BER can be determined without knowledge of the encoding class, for all BER encodings, and serves to identify the encoding for resolution of optionality, ordering of sets, termination of repetitions, and choice alternatives.

9.11 Parameterization

9.11.1 As with ASN.1 types and values, encoding objects, encoding object sets and encoding classes can be parameterized. This is just an extension of the normal ASN.1 mechanism.

9.11.2 A primary use of parameterization is in the definition of an encoding object that needs the identification of a determinant to complete the definition of the encoding (see 9.13.2). (See D.1.11.3 for an example of a parameterized ECN definition.)

9.11.3 Another important use of parameterization is in the definition of an encoding structure that will be used to replace many different classes in an encoding (see also 9.16.5). For example, the mechanism used to handle optionality is often an immediately (mandatory) preceding "presence-bit" for each optional component. A parameterized structure can be defined consisting of a concatenation of a **`#BOOLEAN`** (used as a presence determinant) followed by an optional component defined as a dummy parameter (which will be instantiated with the component that the structure will replace), and whose presence is determined by the **`#BOOLEAN`**. The original **`#OPTIONAL`** encoding procedure is now defined as the replacement of the original component with this mandatory structure, using the original optional component as the actual parameter. (D.3.2 is a more complete example of this process.)

9.11.4 Dummy parameters may be encoding objects, encoding object sets, encoding classes, references to encoding structure fields, and values of any of the ASN.1 types used in the built-in encoding classes defined in clause 23, as specified in Rec. ITU-T X.683 | ISO/IEC 8824-4 as modified by B.10 of this Recommendation | International Standard.

9.11.5 The modification of parameterization syntax that is specified in Annex C requires the use of the symbol "**`{<`**" (without spaces) instead of "**`{`**" to start a dummy or actual parameter list, and of "**`>}`**" to end one.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

NOTE – This was done to make parsing of ECN syntax easier for computers, and to avoid ambiguity when user-defined classes are used in structure definitions in place of **#SEQUENCE**, **#CHOICE**, **#REPETITION**, **#SEQUENCE-OF**, or **#SET-OF**.

9.12 Governors

9.12.1 The concept of a governor and of governed notation will be familiar from ASN.1 value notation, where there is always a type definition that "governs" the value notation and determines its syntax and meaning.

9.12.2 The same concept extends to the definition of encoding objects of a given encoding class. The syntax for defining an encoding object of class **#BOOLEAN** (for example) is very different from the syntax for defining an encoding object of class **#INTEGER** (for example). In all cases where an encoding object definition is required, there is some associated notation that defines the class of that encoding object, and "governs" the syntax to be used in its specification.

9.12.3 The ECN syntax requires governors that are encoding classes to be class reference names, or parameterized class reference names.

9.12.4 If the governed notation is a reference name for an encoding object, then that encoding object is required to be of the same class as the governor (see 17.1.7).

9.13 General aspects of encodings

9.13.1 ECN provides support for a number of techniques typically used in defining encoding rules (not just those techniques used in BER or PER). For example, it recognizes that optionality can be resolved in any of three ways: by use of a presence determinant, by use of an identification handle (see 9.13.3), or by reaching the end of a length-delimited container (or the end of the PDU) before the optional component appears.

9.13.2 Similarly, it recognizes that delimitation of repetitions can be done (for example) by:

- Some form of length count.
- Detecting the end of a container (or PDU) in which it is the last item.
- Use of an identification handle on each of the repetitions and on following encodings (see 9.13.3).
- Some terminating pattern that can never occur in an encoding in the repeated series. (A simple example is a null-terminated character string.)
- Use of a "more bit" with each element, set to one to indicate that another repetition follows, and set to zero to indicate the end of the repetition.

ECN supports all these mechanisms for delimitation of repetitions, and similar mechanisms for identification of alternatives and for resolution of optionality.

9.13.3 In addition to terminating repetitions, the identification handle technique can also be used to determine the presence of optional components or of alternatives and the ordering of sets. The mechanism is similar in all these cases. Given an encoding class that is a "possible next class" and an encoding object applied to it, any encoding produced will contain, at some bit positions (the identification handle), a bit pattern that matches a bit pattern within a specified set of bit patterns (the handle value set) characterizing that class, but does not match any bit pattern characterizing any other "possible next class". All such encodings can be interpreted by a decoder as an encoding of a "possible next class", and the bit pattern found in the encoding will determine which "possible next class" encoding is present. The concept is similar to that of using tags for such purposes in BER. Identification handles have names that are required to be unique within an ECN specification.

9.13.4 It is important here to note that ECN allows the definition of encodings in a very flexible way, but cannot guarantee that an encoding specification is correct – that is, that a decoder can successfully recover the original abstract values from an encoding. For example, an ECN specifier could assign the same bit-pattern for boolean values true and false. This would be an error, and in this case a tool could fairly easily detect the error. Another error would be to claim that an encoding was self-delimiting (and required no length determinant), when in fact it was not. This error could also be detected by a tool. In more subtle and complex cases, however, a tool may find it very hard to diagnose an erroneous (one that cannot always be successfully decoded) specification.

9.14 Identification of information elements

9.14.1 Many protocols have an encoding (usually of a fixed number of bits) to identify what are often called "information elements" or "data elements" in a protocol. These identifications correspond roughly to ASN.1 tags, but are usually less complex. They are often used as identification handles, but are not always so used.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.14.2 ECN contains a **#TAG** class to support the definition of the encoding of information element identifiers through use of the ASN.1 tag notation. (It also supports the inclusion of such elements within an encoding structure with no reference to ASN.1 tags.)

9.14.3 When an encoding structure is implicitly generated from an ASN.1 type definition (see clause 11), the first **textually-present** ASN.1 tag notation in that definition generates an instance of the **#TAG** class, with the number of the ASN.1 tag associated with that instance of the **#TAG** class. Subsequent textually present instances of ASN.1 tag notation are not mapped into #TAG classes in the implicitly generated structure, but these tags and their values become properties of the element. An encoding for this encoding class can be defined in a similar way to an encoding for the **#INTEGER** class, and will encode the number in the tag notation.

9.14.4 The full ASN.1 tag-list (multiple tags each with a class and number) is notionally associated with all the abstract values of a tagged type, in accordance with the ASN.1 model. Such information is, however, only accessible in the current version of ECN through a non-ECN definition of an encoding object (see 9.7.8). The generation of a **#TAG** class is a separate mechanism, is simpler and more specific, and has full support within ECN.

9.14.5 It is, however, important to note that for the purposes of generating a **#TAG** class, it is only textually-present tag notation that is visible. Universal class tags and tags generated by automatic tagging are not visible. Similarly, the class of any textually present tag notation is ignored. Only the tag number is available to encoding objects of the #TAG class.

9.15 Reference fields and determinants

9.15.1 A very common (but not the only) way of determining the presence of an optional field, the length of a repetition, or the selection of an alternative is to include (somewhere in the message) a determinant field. Determinant fields have to be identified if this mechanism is used for determination, and this frequently requires a dummy parameter of an encoding object definition, with the actual parameter, providing the encoding structure fieldname of the determinant, being supplied when the encoding object is applied to an encoding structure.

9.15.2 A new concept – a **reference field** – is introduced to satisfy the need for a dummy parameter that references an encoding structure field. The governor is the reserved word **REFERENCE**, and the allowed notation for an actual parameter with this governor is any encoding structure field name within the encoding structure to which an encoding object or encoding object set with such a parameter is being applied (see 17.5.15). (See D.1.11.3 for an example of references to encoding structure fieldnames.)

9.16 Replacement classes and structures

9.16.1 When writing ASN.1 specifications for legacy protocols (or in order to generate specialized encodings for new protocols), it is normal to ignore encoding issues and, in particular, determinant fields that are present solely to support decoding. Only fields of relevance to application code (carrying application semantics) are included in the ASN.1 specification.

9.16.2 When such protocols use more than one encoding mechanism to support (for example) **SEQUENCE OF** constructions in different places in the protocol, it is not possible (nor would it be appropriate) to formally specify this within the ASN.1 itself.

9.16.3 This means that the implicitly generated encoding structure will not distinguish between such constructions, nor will it contain encoding-related fields for determinants, and it is necessary to modify it to "correct" both problems before a structure is available that matches the encoding requirements.

9.16.4 The first and simplest modification is to replace some instances of a class (within the implicitly generated structure) with new class names that have been assigned the old class in a class assignment statement. This is done by creating an **explicitly generated structure** using a renames clause in an EDM. This clause imports an implicitly generated structure from an ASN.1 module and makes specified replacements of (textual) occurrences of named classes. The replacement can be of all occurrences textually within a list of implicitly generated classes (corresponding to the ASN.1 type definitions in a module), or within components of one of those classes, or "all occurrences except" those in a given definition or a given component (see 15.3). It is important here to note that these replacements are restricted to the use of classes that have been defined with an encoding class assignment statement that assigns the name of a replacement class to an old class (for example: "**#Replacement-class ::= #Old-class**"), so this mechanism is sometimes colloquially referred to as "coloring". The "coloring" identifies those parts of the specification that require different encodings from other parts. (An example of "coloring" is given in D.3.7.)

9.16.5 Even with "coloring", the explicitly generated encoding structure, like the implicitly generated encoding structure, contains only fields corresponding to the fields in the ASN.1 specification, and it is usually necessary to modify the generated structures to add fields for determinants, etc. A new **replacement structure** is needed (for all or part of the original structure), with added fields. It is also important to identify (for each field in the original structure) which fields

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

of the replacement structure (and what abstract values of that field) are used to carry the semantics of the original abstract values. We talk about mapping the abstract values from the original structure to the replacement structure.

9.16.6 There are many mechanisms for defining an encoding object for an existing structure as an encoding object for a totally different replacement structure, with defined **value mappings** between the old structure and the replacement structure. These mechanisms are described in 9.17.

9.16.7 A simpler situation frequently occurs, however, in which the designer requires the old structure to form (in its entirety) a single component of the replacement structure, with all abstract values being mapped from the old structure to the corresponding value of that component of the replacement structure. For this mechanism to be of general use, the replacement structure needs to have a dummy parameter for this single component, and for it to be instantiated with the actual parameter set to the old structure. This was described in 9.11.3.

9.16.8 When defining encoding objects for a class (any class), it is always possible to specify that the first action of that encoding object is to replace the class it is encoding with a parameterized replacement structure, instantiated as described in 9.16.7, and with abstract values mapped from the old class to the component.

9.16.9 It is also possible to define encoding objects for the **#OPTIONAL** class (or for any class of the optionality category) that replace the optional component with a parameterized replacement structure (frequently one containing a #BOOLEAN field as a presence determinant). (An example of this is given in D.3.2.3.)

9.16.10 For constructor classes such as **#CONCATENATION**, **#REPETITION**, and so on, it is also possible to define encoding objects that replace not the entire structure, but each component separately (or just mandatory, or just optional, components).

9.16.11 A more advanced, but powerful, mechanism is to require the replacement action to also include the insertion of a specified field at the head of a **#CONCATENATION** (or similar structure). An example of this is given in D.3.1.5.

9.17 Mapping abstract values onto fields of encoding structures

There are six mechanisms provided for this.

9.17.1 The first is to map specified abstract values associated with one simple encoding class to specified abstract values associated with another simple encoding class. This can be used in many ways. For example, values of a character string (of digits) can be mapped to integer values (and hence encoded as integer values). Values of an enumerated type can be mapped to integer values, and so on (see 19.2). (See D.1.10.2 for an example.)

9.17.2 The second is to map a complete field of one encoding structure into a field of a compatible encoding structure, which can contain additional fields – typically for use as length or choice determinants (see 19.3). (See D.2.8.3 for an example.)

9.17.3 The third is to map by transforming all the abstract values associated with one encoding class into abstract values associated with a different (typically, but not necessarily) encoding class, using a transform encoding object (see 9.18). With this mechanism, it is, for example, possible to map an **#INTEGER** into a **#CHARS** to obtain characters that can then be encoded in whatever way is desired (for example, Binary-Coded Decimal or ASCII). (See D.1.6.3 for an example.)

9.17.4 The fourth mapping mechanism is to use a defined ordering of the abstract values of certain types and constructions, and to map according to the ordering. This provides a very powerful means of encoding abstract values associated with one encoding class as if they were abstract values associated with a wholly unrelated encoding class (see 19.5). (See D.1.4.2 for an example.)

9.17.5 The fifth mechanism is to distribute the abstract values (using value range notation) associated with one encoding class (typically **#INTEGER**) into the fields of another encoding class. (See 19.6 and D.2.1.3 for examples.)

9.17.6 The final mechanism allows the ECN specifier to provide an explicit mapping from integer values (which may have been produced by earlier mappings from, for example, an **#ENUMERATED** class) to the bits that are to be used to encode those values. This is intended to support Huffman encodings, where the frequency of occurrence of each value is (at least approximately) known, and where the optimum encoding is required. Annex E describes Huffman encodings in more detail, and gives examples of this mechanism, together with a reference to software that will generate the ECN syntax for these mappings, given only the relative frequency with which each value of the integer is expected to be used (see 19.7).

9.18 Transforms and transform composites

9.18.1 Transforms are encoding objects of the class **#TRANSFORM**. They can be used to transform abstract values between different encoding classes, and can also be used to define simple arithmetic functions such as multiplication by

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

a fixed value, subtraction of a fixed value, and so on. When applied in succession, they enable general arithmetic to be specified (see 19.4). (See D.2.4.2 for an example.)

9.18.2 A transform can take a single value as its source and then produces a single value as its result. The following is a classification of the values that can be sources and results of transforms:

– an integer;

– a boolean;

– a characterstring;

– a bitstring;

– a single character;

– a single bit (source only, supporting the encoding of a bitstring – see 23.2).

9.18.3 Transform composites are an ordered list of elements, each of which is a single value and has the same classification (as listed in 9.18.2). (For example, an ordered list of single characters, or of single octets, or of integers.) They are only produced as the result of transforms, and can only be used as the source of a following transform.

9.18.4 If the classification is bitstring, the size of each bitstring value in the composite is the same, and is statically determined by the transform that produces the composite. (For example, an ordered list of single bits, or of six-bit units.)

9.18.5 There are transforms from the following abstract values to composites:

– characterstring to a single character composite;

– bitstring to a bitstring composite (all bitstring values of the composite are of the same size);

– octetstring to a bitstring composite (all bitstring values of the composite are of size 8 bits).

9.18.6 There are transforms from the following composites to abstract values:

– single character composites to characterstring values;

– bitstring composites to bitstring values;

– bitstring composites (with bitstring values of size 8 bits) to octetstring values.

9.18.7 All other transforms can take a value as their source and produce a new value (of the same or of a different classification). They can also take a transform composite as their source and produce a composite as its result, transforming each element of the source composite into an element of the result composite.

9.19 Contents of Encoding Definition Modules

9.19.1 Encoding Definition Modules (or EDMs) contain export and import statements exactly like ASN.1 (but can import only encoding objects, encoding object sets, and encoding classes from other EDM modules, or from ASN.1 modules in the case of implicitly generated encoding structures).

9.19.2 An EDM can also contain a renames clause (see clause 15) which references implicitly generated encoding structures from one or more ASN.1 modules and generates, by "coloring" them (see 9.16.4), an explicitly generated encoding structure for each one. These explicitly generated encoding structures are available for use within the EDM, but are also automatically exported for possible import into the Encoding Link Module.

9.19.3 The body of an EDM module contains:

"EncodingObjectAssignment" statements that define and name an encoding object for some encoding class (there are eight forms of this statement, discussed in 9.7 and defined in clause 17).

"EncodingObjectSetAssignment" statements that define sets of encoding objects (see clause 17).

"EncodingClassAssignment" statements that define and name new encoding classes (see clause 15).

9.19.4 The EDM can also contain parameterized versions of these statements, as specified in clause 14 and in C.1.

9.19.5 Encoding objects can be defined for built-in encoding classes within any EDM module. Encoding objects can be defined for a generated encoding structure only in EDM modules that import the implicitly generated encoding structure from the ASN.1 module that defines the corresponding type (using either an imports or a renames clause), or that import the generated encoding structure from an EDM module that has exported it.

NOTE – If an implicitly generated encoding structure happens to have a name that is the same as a reserved encoding class name (see 8.5), it can still be imported into an EDM, but must be referenced in the body of the EDM using a fully-qualified name (see "ExternalEncodingClassReference" in 10.6).

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.20 Contents of the Encoding Link Module

9.20.1 All applications of the Encoding Control Notation require the identification of a single Encoding Link Module (or ELM).

9.20.2 The ELM module applies encoding object sets to ASN.1 types (formally, to a generated encoding structure corresponding to the ASN.1 type). These encoding object sets (or their constituent encoding objects) are imported into the ELM module from one or more EDM modules.

9.20.3 There are restrictions on the application of encoding object sets to ensure that there is no ambiguity about the actual encoding rules that are being applied (see 12.2.5). For example, it is not permitted for an ELM to apply more than one encoding object set to a specific implicitly generated structure.

9.20.4 It is possible in simple cases for an ELM module to contain just a single statement (following an imports clause) that applies an encoding object set to the implicitly generated encoding structure corresponding to the single top-level type of an application. (See D.1.17 for an example.)

9.21 Defining encodings for primitive encoding classes

9.21.1 Encoding rules for some primitive encoding classes can be defined using a user-friendly syntax which is specified in the **`WITH SYNTAX`** statements of encoding class definitions (see clauses 23 and 25). This syntax can also be used to define encoding rules for encoding classes derived from these primitive encoding classes (by encoding class assignment statements).

9.21.2 The notation used for the encoding class definitions in clauses 23 and 25 is based on the notation used for information object class definition. This syntax (and its associated semantics) is defined by reference to Rec. ITU-T X.681 | ISO/IEC 8824-2 as modified by Annex B of this Recommendation | International Standard.

9.21.3 The encoding class definition specifies the information that has to be supplied in order to define encoding rules for particular encoding classes. The set of encoding rules that can be defined in this way is not, of course, all possible rules, but is believed to cover the encoding specifications that ECN users are likely to require.

9.21.4 These encoding class definitions specify a series of fields (with corresponding ASN.1 types and semantics). Encoding rules are specified by providing values for these fields. The values of these fields are effectively providing the values of a series of encoding properties which collectively define an encoding.

9.21.5 The meaning of the encoding properties is specified using an encoding model (see Figure 1) where the value of each bit-field class produces a **value encoding** which is placed (left or right justified) into an **encoding space**.

9.21.6 The encoding space may have its leading edge aligned to some boundary (such as an octet boundary) by encoding space pre-padding, and its size can be fixed or variable. The value encoding fits within it, perhaps left or right justified, and with padding around it. If the size of the encoding space is variable, then either the value encoding has to be self-delimiting, or there has to be some external mechanism to enable a decoder to determine the size of the encoding space. Several mechanisms are available for this determination.

9.21.7 Finally, the complete encoding space with the value encoding and any value pre-padding and value post-padding, is mapped to bits-on-the-line with an optional specification of **bit-reversal**. This handles encodings that require "most significant byte first" or "most significant byte last" for integers, or that require the bits within an octet to be in the reverse of the normal order.

9.21.8 Thus there are three broad categories of information needed:

- the first relates to the encoding space in which the encoding is placed;
- the second relates to the way an abstract value is mapped to bits (value encoding), and the positioning of those bits within the encoding space; and
- the third relates to any required bit-reversals.

9.21.9 Figure 1 shows the encoding space (with pre-padding) and the value encoding (with value pre-padding and value post-padding). Figure 1 also illustrates the specification of an encoding space unit. The encoding space is always an integral multiple of this specified number of bits.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

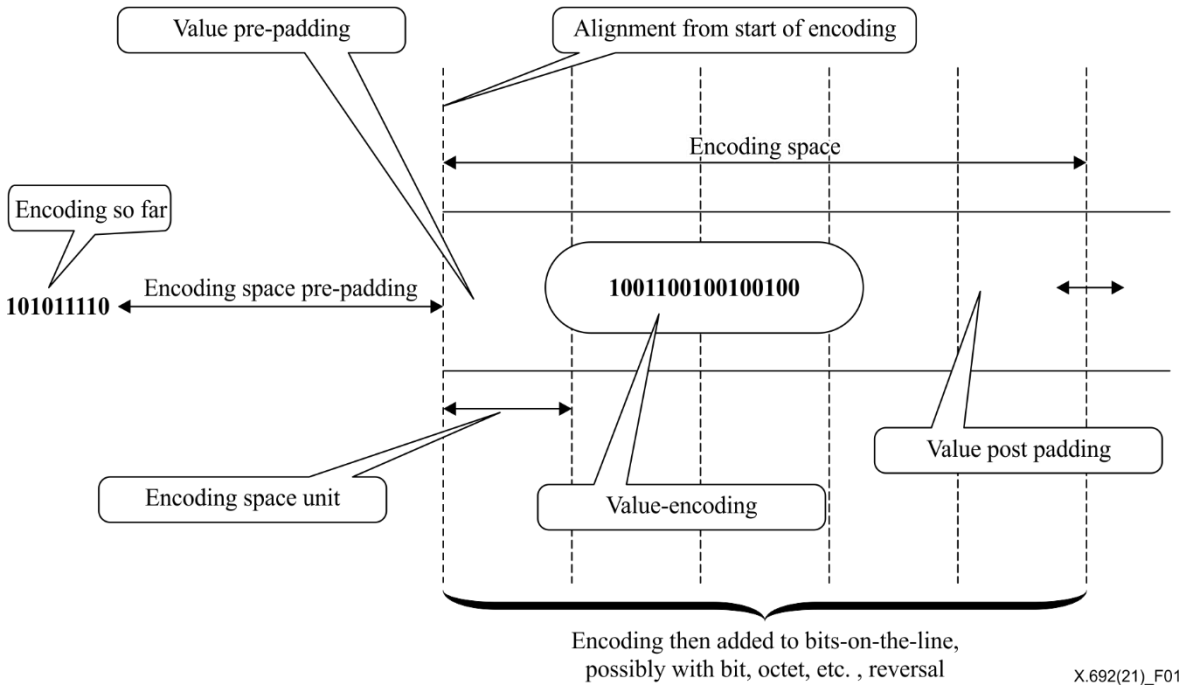

Figure 1 – Encoding space, value-encoding and padding concepts

9.21.10 If the encoding space is not the same size for all values encoded by an encoding object, then some additional mechanism is needed to determine the actual encoding space used in an instance of an encoding.

9.21.11 It is also possible to specify an arbitrary amount of encoder pre-padding (beyond that needed for alignment) that ends when the value of an earlier **start pointer** identifies the start of a field.

9.21.12 The steps in a definition of an encoding for a primitive bit-field encoding class are:

- Specify the alignment (if any) required for the leading edge of the encoding space (relative to the **alignment point** – normally the start of the encoding of the top-level type, that is, the type to which an encoding object set is applied in the ELM). (See 22.2.)
- Specify the form of any necessary padding to that point (encoding space pre-padding). (See 22.2.)
- Specify (if necessary) a field that provides a pointer to the start-point of the encoding space. (See 22.3.)
- Specify the encoding of abstract values into bits (value encoding).
- Specify the units of the encoding space (the encoding space will always be an integral multiple of these units). (See 22.4.)
- Specify the size of the encoding space in these units. This may be fixed (using knowledge of integer or size bounds associated with the abstract values to be encoded), or variable (different for each abstract value). The specification may also (in all cases) specify the use of a length determinant that has to be encoded with the length of the field, and either enables decoding or provides redundant information (in the case of a fixed-size encoding space) that a decoder can check. (See 22.4.)
- Specify the alignment of the value encoding within the encoding space. (See 22.8.)
- Specify the form of any necessary padding from the start of the encoding space to the start of the value encoding (value pre-padding). (See 22.8.)
- Specify the form of any necessary padding between the end of the value encoding and the end of the encoding space (value post-padding). (See 22.8.)
- Specify any necessary bit-reversals of the encoding space contents before adding the bits to the encoding done so far. (See 22.12.)

9.21.13 Encoding properties are available to support the specification of the encoding rules for all these steps.

9.21.14 In real cases, only some (or none!) of these encoding properties will have unusual values, and defaults operate if they are not specified. (See D.1.3 for an example of the definition of the encoding for an integer that is right-aligned in a fixed two-octet field, starting at an octet boundary.)

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.22 Application of encodings

9.22.1 Application of encodings (encoding rules) to encoding structures is a key part of the ECN work, but is very distinct from the definition of the encoding rules. Final application of encodings (to an encoding structure generated from an ASN.1 type definition) only occurs within an Encoding Link Module, but application of encodings to fields of an encoding structure may be used in the definition of encodings for a larger encoding structure.

9.22.2 Encodings are applied by reference to an encoding object set (or to a single encoding object). Such application can occur in an EDM in the definition of encoding objects for any class (including encoding objects for a generated encoding structure and for a user-defined encoding structure). Such application in an EDM is merely the definition of more encoding objects for that encoding class: The definitive application to an actual type occurs only in the ELM.

9.22.3 When a set of encoding objects is being applied, it always results in a complete encoding specification for the encoding classes to which the objects are applied. If, in any given application, encodings are needed for encoding classes (present within an encoding structure being encoded) for which there are no encoding objects in the set being applied, then this is an error (see 13.2.11).

NOTE – Although the specification of the encoding rules will be complete, the precise form of the actual encoding (for example, the presence or absence of encoding space pre-padding, or the effect of the values of bounds referenced in the encoding rules) can only be determined when the encoding definition is applied to a top-level ASN.1 type.

9.22.4 There are two exceptions to 9.22.3. The first exception is when the (ASN.1-like) parameterization mechanism is used to define a parameterized encoding object. In such cases the complete encoding is only defined following instantiation with actual parameters. The second exception is when an encoding object is defined for an encoding constructor (**`#CONCATENATION`**, **`#ALTERNATIVES`**, **`#REPETITION`**, **`#SEQUENCE`**, etc.). In this latter case, the encoding rules associated with the encoding class simply define the rules associated with the structuring aspects. A complete encoding specification for an encoding structure using these encoding classes will also require rules for encoding the components of that encoding structure.

NOTE – There is a distinction here between encoding objects of class **`#SEQUENCE`** (an encoding constructor) and encoding objects for an implicitly generated encoding structure "**`#My-Type`**" (which happens to be defined using the ASN.1 type **`SEQUENCE`**). The latter is not an encoding constructor, and encoding objects of this class will provide full encoding rules for the encoding of values of type "**`My-Type`**".

9.23 Combined encoding object set

9.23.1 In order to provide a complete encoding, the ECN user can supply a primary encoding object set, and a second encoding object set introduced by the reserved words **`COMPLETED BY`**.

9.23.2 The encoding object set that is applied is defined to be the **combined encoding object set** formed by adding to the first set encoding objects for any encoding class for which the first set is lacking an encoding object and the second set contains one (see 13.2). A frequent set to use with **`COMPLETED BY`** is the built-in set **`PER-BASIC-UNALIGNED`**. (See D.1.17 for an example of the application of a combined encoding object set.)

9.23.3 While an encoding object set can contain only one encoding object for a class **`#SEQUENCE-OF`** (for example), it can also contain an encoding object for a class **`#Special-sequence-of`** (for example) which is defined as "**`#Special-sequence-of ::= #SEQUENCE-OF`**". An explicitly generated encoding structure can have both the **`#SEQUENCE-OF`** class and also the **`#Special-sequence-of`** class in its definition. In this way, a single combined encoding object set can be applied to produce standard encodings for some of the original **`SEQUENCE OF`** constructs, and specialized encodings for others.

9.24 Application point

9.24.1 In any given application of encodings, there is a defined starting point (for the ELM, it is the top-level generated encoding structure(s) to which encodings are being applied). This is called the "initial application point" for the structure that is being encoded by the ELM.

9.24.2 The combined encoding object set is applied to a generated encoding structure, and it is the encodings defined for the abstract values of this encoding structure that encode the abstract values of the ASN.1 type.

9.24.3 If there is an encoding object in the combined encoding object set that matches a bit-field encoding class (initially a generated encoding structure) at the application point, it is applied and the process terminates. Otherwise the class at the application point is "expanded" by de-referencing. This expansion by de-referencing will continue until either an encoding object is found, or a primitive class is reached. If the class at the application point is an encoding constructor, and there is an encoding object for that encoding constructor (**`#CHOICE`**, **`#SEQUENCE`**, **`#SEQUENCE-OF`**, etc.), then it is applied, and the application point then passes to each component (as a parallel activity).

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.24.4 In a more complex case, there may be an **#OPTIONAL** class following a component class (and a **#TAG** class preceding it). The application point passes first to the **#OPTIONAL**, and the encoding object for that class may replace the component (see 9.16.9). Then the application point passes to the tag, and finally to the component itself.

9.25 Conditional encodings

9.25.1 Mention has already been made of the **#TRANSFORM** encoding class as a means of performing simple arithmetic on integer values (see 9.17.3). This encoding class does, however, play a more fundamental role in the specification of encodings for some primitive classes. In general, the specification of encodings for many of the ASN.1 built-in types is a two or a three stage process, using encoding objects of class **#TRANSFORM** and (for example) of class **#CONDITIONAL-INT** or **#CONDITIONAL-REPETITION**.

9.25.2 The **#TRANSFORM**, **#CONDITIONAL-INT**, and **#CONDITIONAL-REPETITION** encoding classes are restricted in their use. Encoding objects can only be defined for these classes using either the syntax of clause 24, 23.7 and 23.14 respectively, or by non-ECN definition of an encoding object, and they can only be used in the definition of other encoding objects. They cannot appear in encoding object sets or be applied directly to encode fields of encoding structures (see 18.1.7).

9.25.3 Encoding specification for encoding classes in the integer category proceeds as follows: Encodings (of the **#CONDITIONAL-INT** encoding class) are defined for a particular **bounds condition,** specifying the container size (and how it is delimited), the transform of the integer to bits (using either two's complement or positive integer encodings), and the way these bits fit into the container. (An example of a bounds condition is the existence of an upper bound and a non-negative lower bound.) This is called a **conditional encoding**. The encoding of the class in the integer category is defined as a list of these conditional encodings, with the actual encoding to be applied in any given circumstance being the one that is earliest in the list whose bounds condition is satisfied. (See D.1.5.4 for an example.)

9.25.4 Encoding specification for encoding classes in the repetition category use the **#CONDITIONAL-REPETITION** encoding class, which defines the way in which the encoding space for the repeated items is delimited and how the repeated encodings are to be placed into it, for a given **range condition**, again producing a conditional encoding. As with the encoding of classes in the integer category, the final encoding is defined as an ordered list of conditional encodings.

9.25.5 Encoding specification for the encoding classes in the octetstring category proceeds as follows: First, **#TRANSFORM** encoding objects are defined to map a single octet to a self-delimiting bitstring. Second, one or more **#CONDITIONAL-REPETITION** encoding objects (for specific size-range conditions) are defined to take each of the bitstrings (transformed from an octet in the octet string) and to concatenate them into a delimited container (the definition of such encoding objects is not specific to encoding **#OCTETS**). The final encoding of the class in the octetstring category is defined as an ordered list of **#CONDITIONAL-REPETITION** encoding objects. (See D.1.8.2 for an example.)

9.25.6 Encoding specifications for encoding classes in the bitstring category proceeds as follows: First, **#TRANSFORM** encoding objects are defined to map a single bit into a bitstring, similar to the encoding of an integer into bits, but in this case the mapping of the bit must be to a self-delimiting string. Secondly, one or more **#CONDITIONAL-REPETITION** encoding objects are defined for the repetition of the bits (these could be the same encoding objects that were defined for use with an encoding class in the repetition or octetstring categories). Finally, the encoding of the class in the bitstring category is defined as an ordered list of **#CONDITIONAL-REPETITION** encoding objects. (See D.1.7.3 for an example.)

9.25.7 Encoding specifications for encoding classes in the characterstring category proceeds as follows: First, **#TRANSFORM** encoding objects are defined to map a single character to a self-delimiting bitstring, using several possible mechanisms for defining the encoding of the character, and using the effective permitted alphabet constraint where it is available. Secondly, one or more **#CONDITIONAL-REPETITION** encoding objects are defined, and finally the encoding of the class in the characterstring category is defined as an ordered list of these. (See D.1.9.2 for an example.)

9.26 Other conditions for applying encodings

9.26.1 There are a number of different conditions that can be tested in order to select an appropriate encoding. These include the actual value and the range of bounds.

9.26.2 It is also possible to require that all of a given list of conditions are to be satisfied.

9.26.3 A test for a condition uses either a single enumeration value (such as "**bounded-without-negatives**") which contains the entire test in the specification of the one enumeration, or a triple of enumerations.

9.26.4 If a triple is used, the first identifies (by an enumeration) the item that is being tested (for example "**test-upper-bound**"), the second is the nature of the test (for example "**greater-than**"), and the third provides an integer value for the test.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

9.27 Encoding control for the open type

9.27.1 Open types frequently provide a means of extensibility using an identification field, with new values for the identification field and new types for the open type being added in successive versions (and often being available for vendor-specific extensions).

9.27.2 Both these features mean that a decoder may be asked to decode an open type when that particular implementation has no knowledge of the type that has been encoded into it.

9.27.3 The encoding support provided for the open type is the same as that for most other classes in the bitfield category, but with the added ability to specify that a different encoding object set is to be applied to the type which is to be encoded into the open type.

NOTE – This is in recognition that many protocols choose to use a different style of encoding (often based on a type-length-value approach) for the type contained in an open type, while retaining a more compact style of encoding for the fields of the message containing the open type.

9.27.4 The model used for decoding an open type recognizes that a decoder will not know what type fills the open type (table and relational constraints are not visible to either PER or to ECN), but that the application may be able to determine this from some other field in the protocol, or in a previous message, or (for vendor-specific additions) based on calling address.

9.27.5 The model is therefore that, having dealt with any specified pre-padding, and determined the encoding space and any value pre- and post-padding, the decoder will ask the application for the type which has been encoded. (In the case of tools, the application will almost certainly have pre-configured the tool with a list of the known types that might be present, and would simply return a pointer to one of these.) Decoding can now proceed normally.

9.27.6 The application may, however, say "unknown" (see 9.27.4), and the decoder then needs to know how to determine the end of this unknown encoding. This is satisfied by enabling the ECN specifier in this case to provide an encoding structure, and (optionally) an encoding object set to use with it, which is to be used by decoders for decoding unknown types in the open type. There is syntax provided in clause 23 for this purpose.

NOTE – An example of such an encoding structure could be one that specifies an encoding that is commonly known as a "Type, Length, Value" encoding, whose end can be determined without knowledge of the type being encoded.

9.28 Changes to ASN.1 Recommendations | International Standards

9.28.1 This Recommendation | International Standard references other ASN.1 Recommendations | International Standards in order to define its notation without repetition. For such references to be correct, the semantics of the notation (for example the imports clause, parameterization, and information object definition) needs to be extended to recognize the reference names of encoding classes, encoding objects, and so on that form part of ECN.

9.28.2 There is also a need to extend the information object class notation to allow fields that are ordered lists of values or objects, not just unordered sets of objects, in order to allow the use of that notation in the definition of ECN syntax for the definition of encoding objects of certain classes.

9.28.3 Finally, the rules for parameterization are relaxed to allow a dummy parameter of an encoding object reference (being assigned in an assignment statement) to be used as an actual parameter of the encoding class reference which governs the notation defining the encoding object reference name. In particular, a parameterized encoding class can be used as a governor in an encoding object assignment statement (see C.2/8.4), with the actual parameter being a dummy parameter of the encoding object that is being defined.

9.28.4 These modifications to other ASN.1 Recommendations | International Standards are specified in Annexes A to C, and are solely for the purposes of this Recommendation | International Standard.

10 Identifying encoding classes, encoding objects, and encoding object sets

10.1 Many of the productions within this Recommendation | International Standard require that an encoding class, encoding object, or encoding object set be identified.

10.2 For each of these, there are five ways in which identification can be made:

a) Using a simple reference name.

b) Using a built-in reference name (not applicable for encoding objects, as there are no built-in encoding objects).

c) Using an external reference (also called a fully-qualified name).

d) Using a parameterized reference.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

e) In-line definition.

NOTE – The parameterized reference form may be used with a simple reference name or with an external reference (see C.3).

10.3 There are productions (or lexical items) for all of these means of identification. There are also productions that allow several alternatives. These lexical items or production names are used where appropriate in other productions, and are defined in the remainder of this clause.

10.4 The lexical items for use of a simple reference name are:

encoding class	**"encodingclassreference"** (see 8.3)
encoding object	**"encodingobjectreference"** (see 8.1)
encoding object set	**"encodingobjectsetreference"** (see 8.2)

10.4.1 An "encodingclassreference" is a name which is either:

a) assigned an encoding class in an "EncodingClassAssignment" (see clause 16); or is

b) imported into an EDM from some other EDM from which it has been exported; or is

c) imported as the name of an implicitly generated encoding structure from an ASN.1 module (see 14.11); or is

d) generated by a renames clause in the EDM (see clause 15).

NOTE – Only classes that are generated encoding structures can be imported into an ELM (see 12.1.8).

10.4.2 An "encodingclassreference" shall not be imported from an EDM module (as specified in 10.4.1) unless either:

a) it is defined in or imported into the referenced module, and that module has no exports clause; or

NOTE 1 – If the referenced module has no exports clause, this is equivalent to exporting everything.

b) it is defined in or imported into the referenced module, and appears as a symbol in the exports clause of that module; or

c) it is one of the reference names explicitly generated by a renames clause in the module from which it is being imported.

NOTE 2 – Implicitly generated encoding structures can only be imported from the ASN.1 module which generates them.

10.4.3 An implicitly generated encoding structure reference never appears in the exports clause of any ASN.1 module, but can always be imported from any ASN.1 module in which the corresponding type is defined and exported.

10.4.4 An explicitly generated encoding structure reference (which is automatically exported by the renames clause which generates it) shall not appear in the exports clause of the EDM module in which it is generated, but any use of it in another EDM or the ELM requires its importation from that EDM module.

10.4.5 An "encodingobjectreference" is a name which is either:

a) assigned an encoding object in an "EncodingObjectAssignment" (see clause 17) in an EDM; or is

b) imported into an EDM or an ELM from some other EDM in which it is either assigned an encoding object or is imported.

10.4.6 An "encodingobjectreference" shall not be imported from an EDM if the referenced module has an exports clause and the "encodingobjectreference" does not appear as a symbol in that exports clause.

NOTE – If the referenced module has no exports clause, this is equivalent to exporting everything.

10.4.7 An "encodingobjectsetreference" is a name which is either:

a) assigned an encoding object set in an "EncodingObjectSetAssignment" (see clause 18) in an EDM; or is

b) imported into an EDM or an ELM from some other EDM in which it is either assigned an encoding object set or is imported.

10.4.8 An "encodingobjectsetreference" shall not be imported from an EDM if the referenced module has an exports clause and the "encodingobjectsetreference" does not appear as a symbol in that exports clause.

NOTE – If the referenced module has no exports clause, this is equivalent to exporting everything.

10.5 The productions for use of a built-in reference name are:

encoding class	**"BuiltinEncodingClassReference"** (see 16.1.6)
encoding object set	**"BuiltinEncodingObjectSetReference"** (see 18.2.1)

10.6 The productions for use of an external reference name are:

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

ExternalEncodingClassReference ::=
modulereference "." encodingclassreference |
modulereference "." BuiltinEncodingClassReference

ExternalEncodingObjectReference ::=
modulereference "." encodingobjectreference

ExternalEncodingObjectSetReference ::=
modulereference "." encodingobjectsetreference

10.6.1 The "modulereference" is defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 12.5, and identifies a module which is referenced in the imports list of the EDM or ELM.

10.6.2 The "ExternalEncodingClassReference" alternative that includes a "BuiltinEncodingClassReference" shall be used in the body of an EDM if and only if there is a generated encoding structure (whose name is the same as that of a "BuiltinEncodingClassReference") which is either:

a) defined implicitly in the ASN.1 module referenced by the "modulereference" (see 11.4.1); or

b) imported into another EDM referenced by the "modulereference" and exported from that module; or

c) generated in a renames clause of another EDM referenced by the "modulereference"; or

d) generated in this EDM in a renames clause, in which case the "modulereference" shall refer to this EDM.

NOTE – The "BuiltinEncodingClassReference" name can appear as a "Symbol" in the imports clause (see A.1).

10.6.3 The productions defined in 10.6 (except as specified in 10.6.2) shall be used if and only if the corresponding simple reference name has been imported from the module identified by the "modulereference", and either:

a) identical reference names have been imported from different modules, or have been generated in a renames clause in this EDM, or have been both imported and generated; or

b) the simple reference name is a "BuiltinEncodingClassReference" (see 10.5); or

c) both conditions hold.

10.7 A parameterized reference is a reference name defined in a "ParameterizedAssignment" (see C.1) and supplied with an actual parameter in accordance with the syntax of C.3. The productions involved are:

encoding classes	**"ParameterizedEncodingClassAssignment"** (see C.1) **"ParameterizedEncodingClass"** (see C.3)
encoding objects	**"ParameterizedEncodingObjectAssignment"** (See C.1) **"ParameterizedEncodingObject"** (See C.3)
encoding object sets	**"ParameterizedEncodingObjectSetAssignment"** (See C.1) **"ParameterizedEncodingObjectSet"** (See C.3)

10.8 The productions that allow all forms of identification are:

encoding classes	**"EncodingClass"** (See clause 16.1.5)
encoding objects	**"EncodingObject"** (See clause 17.1.5)
encoding object sets	**"EncodingObjectSet"** (See clause 18.1)

10.9 The productions which allow all forms except in-line definition are:

encoding classes	**"DefinedEncodingClass" and "DefinedOrBuiltinEncodingClass"**
encoding objects	**"DefinedEncodingObject"**
encoding object sets	**"DefinedEncodingObjectSet" and "DefinedOrBuiltinEncodingObjectSet"**

except that built-in encoding classes and built-in encoding object sets are not allowed by "DefinedEncodingClass" and "DefinedEncodingObjectSet".

NOTE – A further production "SimpleDefinedEncodingClass" is also used. This is defined in C.3 and allows only "encodingclassreference" and "ExternalEncodingClassReference".

10.9.1 The "DefinedEncodingClass" and "DefinedOrBuiltinEncodingClass are:

DefinedEncodingClass ::=
encodingclassreference
| ExternalEncodingClassReference
| ParameterizedEncodingClass

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

DefinedOrBuiltinEncodingClass ::=
DefinedEncodingClass
| BuiltinEncodingClassReference

10.9.2 The "DefinedEncodingObject" is:

DefinedEncodingObject ::=
encodingobjectreference
| ExternalEncodingObjectReference
| ParameterizedEncodingObject

10.9.3 The "DefinedEncodingObjectSet" and "DefinedOrBuiltinEncodingObjectSet" are:

DefinedEncodingObjectSet ::=
encodingobjectsetreference
| ExternalEncodingObjectSetReference
| ParameterizedEncodingObjectSet

DefinedOrBuiltinEncodingObjectSet ::=
DefinedEncodingObjectSet
| BuiltinEncodingObjectSetReference

11 Encoding ASN.1 types

11.1 General

11.1.1 For all ASN.1 types, there is a corresponding implicitly generated encoding structure. This encoding structure is implicitly generated for each ASN.1 type assignment, and is automatically exported from the ASN.1 module that contains that type assignment. (It does, however, have to be imported into an EDM module if it is to be used.) The name of the corresponding encoding structure is the name of the type preceded by a character "**#**". This encoding structure defines an encoding class, and is called an **implicitly generated encoding structure**.

11.1.2 There may also be one or more **explicitly generated encoding structures**. These are generated in an EDM using a renames clause.

11.1.3 The encoding of an ASN.1 type is formally defined as the result of encodings applied to precisely one of the encoding structures (implicitly or explicitly) generated from the ASN.1 type. The encodings are applied by statements in the ELM (see clause 12), using encoding objects in a combined encoding object set. An ELM shall apply encodings to at most one of the generated encoding structures corresponding to any given ASN.1 type.

11.1.4 The implicitly generated encoding structure is defined by first simplifying and expanding the ASN.1 notation (as specified in 11.3), and then by mapping ASN.1 types, type constructors and component names into corresponding built-in encoding classes, encoding constructors and encoding structure fieldnames.

11.1.5 An explicitly generated encoding structure is defined by making specified changes to the implicitly generated encoding structure using a renames clause.

11.1.6 Each field of a generated encoding structure has associated with it the abstract values of the corresponding type, and constraint-related information derived from the ASN.1 type definition (see 11.4.2). Encodings of the abstract values of the generated encoding structure are defined to be the encodings for the corresponding abstract values of the original ASN.1 type.

11.1.7 This clause 11 specifies:

a) The built-in encoding classes that are used in defining the implicitly generated encoding structures corresponding to ASN.1 types (see 11.2).

NOTE – Subclause 16.1.14 specifies additional classes that are used in the definition of user-defined encoding structures.

b) Transformations of the ASN.1 syntax (simplification and expansion) before the implicitly generated structure is produced (see 11.3).

c) The implicitly generated encoding structure for any ASN.1 type (see 11.4).

11.2 Built-in encoding classes used for implicitly generated encoding structures

11.2.1 The encoding classes used for implicitly generated encoding structures, and the ASN.1 types or constructors to which they correspond are listed in Table 2 below.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

11.2.2 Column 1 gives the ASN.1 notation which is replaced by an encoding class in the implicitly generated encoding structure. Column 2 gives the encoding class that replaces the column 1 notation. Column 3 gives the primitive class that the column 2 class is derived from.

Table 2 – Encoding classes for ASN.1 notation

ASN.1 notation	Encoding Class	Primitive Class
BIT STRING	#BIT-STRING	#BITS
BOOLEAN	#BOOLEAN	#BOOL
CHARACTER STRING	#CHARACTER-STRING	*Defined using* ***#SEQUENCE***
CHOICE	#CHOICE	#ALTERNATIVES
EMBEDDED PDV	#EMBEDDED-PDV	*Defined using* ***#SEQUENCE***
ENUMERATED	#ENUMERATED	#INT
EXTERNAL	#EXTERNAL	*Defined using* ***#SEQUENCE***
INTEGER	#INTEGER	#INT
NULL	#NULL	#NUL
OBJECT IDENTIFIER	#OBJECT-IDENTIFIER	#OBJECT-IDENTIFIER
OCTET STRING	#OCTET-STRING	#OCTETS
open type notation	#OPEN-TYPE	#OPEN-TYPE
OPTIONAL	#OPTIONAL	#OPTIONAL
REAL	#REAL	#REAL
RELATIVE-OID	#RELATIVE-OID	#OBJECT-IDENTIFIER
SEQUENCE	#SEQUENCE	#CONCATENATION
SEQUENCE OF	#SEQUENCE-OF	#REPETITION
SET	#SET	#CONCATENATION
SET OF	#SET-OF	#REPETITION
TIME	#TIME	#TIME
DATE	#DATE	#TIME
TIME-OF-DAY	#TIME-OF-DAY	#TIME
DATE-TIME	#DATE-TIME	#TIME
DURATION	#DURATION	#TIME
GeneralizedTime	#GeneralizedTime	#CHARS
UTCTime	#UTCTime	#CHARS
ObjectDescriptor	#ObjectDescriptor	#CHARS
BMPString	#BMPString	#CHARS
GeneralString	#GeneralString	#CHARS
GraphicString	#GraphicString	#CHARS
IA5String	#IA5String	#CHARS
NumericString	#NumericString	#CHARS
PrintableString	#PrintableString	#CHARS
TeletexString	#TeletexString	#CHARS
UniversalString	#UniversalString	#CHARS
UTF8String	#UTF8String	#CHARS
VideotexString	#VideotexString	#CHARS
VisibleString	#VisibleString	#CHARS
Textually present tag notation	#TAG	#TAG

11.3 Simplification and expansion of ASN.1 notation for encoding purposes

11.3.1 ECN assumes that certain ASN.1 syntactic constructs have been expanded (or reduced) into equivalent or simpler constructions.

NOTE – The types defined by the simpler constructions are capable of carrying the same set of abstract values as the original ASN.1 syntactic structures, and those abstract values are mapped to the simpler constructions.

11.3.2 The expansion or simplification of ASN.1 syntactic productions is either:

a) fully-defined in clause 11.3.4 below; or

b) referenced in those clauses as "See 11.3.2 b" and fully-defined in Rec. ITU-T X.680 | ISO/IEC 8824-1 (including Annex C) with all published amendments and technical corrigenda; or

c) referenced in those clauses as "See 11.3.2 c" and fully-defined in Rec. ITU-T X.681 | ISO/IEC 8824-2 with all published amendments and technical corrigenda.

d) referenced in those clauses as "See 11.3.2 d" and fully-defined in Rec. ITU-T X.683 | ISO/IEC 8824-4 with all published amendments and technical corrigenda.

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

11.3.3 The ASN.1 syntactic constructs removed by the expansions and simplifications below are not referenced further in this Recommendation | International Standard.

11.3.4 The following expansions and simplifications shall be applied to all ASN.1 modules:

11.3.4.1 The following transformations are not recursive and hence are applied only once:

a) All "ValueSetTypeAssignment"s shall be replaced by their equivalent "TypeAssignment"s with subtype constraints. (See 11.3.2 b.)

b) The ASN.1 **INSTANCE OF** construction shall be expanded into its equivalent sequence type. (See 11.3.2 c.)

c) "TypeFromObject" shall be replaced with the type that is referenced. (See 11.3.2 c.)

d) "ValueSetFromObjects" shall be replaced with the type that is referenced. (See 11.3.2 c.)

e) Where an instance of ASN.1 tag notation is textually followed by one or more further instances of ASN.1 tag notation, the second and subsequent instances of tag notation are discarded.

NOTE – This is similar to the rules for implicit tagging in ASN.1, but applies for all tagging environments. Multiple tagging of the same type is still possible through the use of type reference names.

11.3.4.2 The following transformations shall be applied recursively in the specified order, until a fixed-point is reached:

a) All ASN.1 parameterization shall be fully resolved by the substitution of actual parameters for dummy parameters. (See 11.3.2 d.)

NOTE – This means that where ASN.1 type notation contains an instantiation of an ASN.1 parameterized type, that instantiation becomes an inline definition.

b) All "ComponentsOf"s shall be expanded to their full form. (See 11.3.2 b.)

c) All uses of "SelectionType" shall be resolved. (See 11.3.2 b.)

11.3.4.3 The following transformations shall then be applied:

a) Named number lists in integer type definitions shall be removed. Named numbers are not visible to ECN. ECN sees a single **#INTEGER** class (possibly with bounds as specified in 11.3.4.3 c).

b) Named bit lists in bitstring definitions shall be removed. Named bits are not visible to ECN.

c) All non-PER-visible constraint notation, except the contents constraint, shall be discarded. PER-visible constraints shall be resolved to provide the following values that can be referenced in the definition of encoding rules:

i) An upper bound on integers and enumerations;

ii) A lower bound on integers and enumerations;

iii) The PER effective permitted alphabet and effective size constraints (see Rec. ITU-T X.691 | ISO/IEC 8825-2, 10.3).

d) If there is a contents constraint with a **CONTAINING** construction, then the existence of the contents constraint, its contents type, and the presence or absence of an **ENCODED BY** clause become properties associated with the abstract values of such a constrained octetstring or bitstring type, and the constraint shall then be discarded. If there is a contents constraint with no **CONTAINING** construction, then it is not visible to ECN and shall be discarded.

NOTE – When specifying encodings for values with an associated contents constraint, a separate combined encoding object set can be supplied to encode the contents type. This can be specified to override or not to override any **ENCODED BY** that is present, as a designer's option (see 11.3 and 13.2).

e) All tagging which is not textually present in the ASN.1 notation shall be ignored in the mapping to encoding structures, but (in order to model BER encodings and PER procedures) the full tag-list of a type becomes a property of the field of the encoding structure to which the corresponding values are mapped.

f) Textually present tag notation has the class of the tag removed. (See also 11.3.4.1 e.)

g) "**DEFAULT** Value" shall be replaced by "**OPTIONAL-ENCODING #OPTIONAL**" and the default value is associated with the field of the structure to which the ASN.1 component is mapped.

h) **OPTIONAL** shall be replaced by "**OPTIONAL-ENCODING #OPTIONAL**".

i) **T61String** shall be replaced by **#TeletexString**.

j) **ISO646String** shall be replaced by **#VisibleString**.

11.3.4.4 Finally, the following transformations shall then be applied:

a) Automatic allocation of values to enumerations (if applicable) shall be performed. The **ENUMERATED** syntax shall be replaced by the #**ENUMERATED** encoding class with an upper bound and lower bound set. (See 11.3.4.3 c.)

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

NOTE 1 – The **#ENUMERATED** class de-references to the **#INT** class (see 11.2.2), and the enumerations map into bounded integer values of the class. The actual names of enumerations are not visible to ECN.

b) All occurrences of "ObjectClassFieldType" (see Rec. ITU-T X.681 | ISO/IEC 8824-2, clause 14) that refer to a type field, a variable-type value field, or a variable-type value set field shall be replaced by the **#OPEN-TYPE** encoding class. (See 11.3.2 c.)

c) Extensibility markers and version brackets in sequence, set and choice constructions are removed, but (in order to model BER encodings and PER procedures) the identification of a component as part of the root or of version 1, version 2, etc. becomes a property of the component, and the existence of the extensibility marker becomes a property of the class the construction maps to.

d) The extensibility marker in constraints is removed, but the existence of the extensibility marker becomes a property of the class and whether an abstract value is in the root or is in an extension becomes a property of the abstract value.

NOTE 2 – The properties referenced in items c) and d) above can only be interrogated through non-ECN definition of encoding objects in this version of this Recommendation | International Standard. Full support for extensibility is expected to be provided in a later version of this Recommendation | International Standard.

11.3.5 With these transformations, all ASN.1 type-related constructs have corresponding encoding classes, listed in Table 2. The implicitly generated encoding structure shall be constructed by mapping the ASN.1 type-related constructs in column 1 to the classes in column 2 of Table 2 (as specified in 11.4).

11.4 The implicitly generated encoding structure

11.4.1 There is an implicitly generated structure for each ASN.1 type definition with a name constructed from the ASN.1 type reference name by the pre-fixing of a "**#**" character. Where a fully-qualified name is required for an implicitly generated encoding structure, that fully-qualified name shall include the "ModuleIdentifier" of the ASN.1 module containing the type definition. (An example of an implicitly generated structure is given in D.1.9.2.)

NOTE – An implicitly generated structure is generated and exported for each ASN.1 type in an ASN.1 module whether or not that type is listed in the **EXPORTS** clause.

11.4.2 The implicitly generated encoding structure has the same structure as the ASN.1 type definition, with:

a) ASN.1 component identifiers are mapped to encoding structure fieldnames.

b) ASN.1 notation in column 1 of Table 2 are mapped to the built-in encoding classes in column 2 of Table 2.

NOTE 1 – The first textually present tag maps into a "**[#TAG]**" construction in the implicitly generated structure. The implicitly generated structure does not contain any "**[#TAG]**" constructions for subsequent textually present tags.

c) ASN.1 "DefinedType"s are mapped to an encoding class name derived from the typereference by the addition of a character "**#**". If a type is imported into the ASN.1 module, any "ExternalEncodingClassReference" to the corresponding class in an implicitly generated structure shall reference the ASN.1 module that contains the definition of the referenced type.

NOTE 2 – If the resulting class is the name of a built-in encoding class, then all references to it in either the renames clause, or in the ELM, will use the "ExternalEncodingClassReference" notation.

d) Abstract values are mapped from a field of the type definition to the corresponding field of the encoding structure.

e) Upper and lower bounds on integer and enumerated types and all effective size constraints and effective permitted alphabet constraints (see Rec. ITU-T X.691 | ISO/IEC 8825-2, 10.3) are mapped from the type definition to the corresponding field of the encoding structure.

f) The tag number of the first textually present tag maps to the **#TAG** class.

11.4.3 Three further implicitly generated structures are produced and exported from all ASN.1 modules. These structures have names **#CHARACTER-STRING**, **#EMBEDDED-PDV** and **#EXTERNAL**, and the structures that they de-reference to are the implicitly generated structures corresponding to the associated types for **CHARACTER STRING**, **EMBEDDED PDV** and **EXTERNAL**, specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 44.5, 36.5 and 37.5 respectively.

11.4.4 All implicitly generated encoding structures can be encoded by the built-in encoding object sets (see 18.2), and will produce the same encodings as are specified by the corresponding Recommendation | International Standard for those encodings when applied to ASN.1 types.

12 The Encoding Link Module (ELM)

NOTE – There are two top-level productions in ECN, the "ELMDefinition" specified in this clause and the "EDMDefinition" specified in clause 14. These specify the syntax for defining the ELM and EDMs respectively.

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

12.1 Structure of the ELM

12.1.1 The "ELMDefinition" is:

```
ELMDefinition ::=
        ModuleIdentifier
        LINK-DEFINITIONS
        "::="
        BEGIN
        ELMModuleBody
        END
```

12.1.2 In any given application of ECN, there shall be precisely one ELM which determines the encoding of all the messages used in that application.

NOTE – The ASN.1 type(s) defining "messages" are often referred to as "top-level types".

12.1.3 The production "ModuleIdentifier" (and its semantics) is defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.1.

12.1.4 The "ModuleIdentifier" provides unambiguous identification of any module in the set of all ASN.1, ELM, and EDM modules.

12.1.5 The "ELMModuleBody" is:

```
ELMModuleBody ::=
    Imports ?
    EncodingApplicationList

EncodingApplicationList ::=
        EncodingApplication
        EncodingApplicationList ?
```

12.1.6 The production "Imports" (and its semantics) is defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.1, 13.16, and 13.17, as modified by A.1 of this Recommendation | International Standard.

12.1.7 All reference names used in the "ELMModuleBody" shall be imported into the ELM.

NOTE – This is a stronger requirement than that imposed for ASN.1 modules. In ASN.1 modules external references can be used for types and values that have not been imported. In an ELM module (and in an EDM module) external references can only be used for encoding classes that have been referenced in an imports clause. The purpose of external references is solely to resolve ambiguities between imported names and built-in names, or between two identical names imported from different modules.

12.1.8 The "Imports" makes available within the ELM:

a) implicitly generated encoding structures from an ASN.1 module;

b) explicitly generated encoding structures from an EDM module;

NOTE – When an ELM imports an explicitly generated encoding structure from an EDM, the renames clauses in other EDMs have no effect on the encoding of that structure (see 15.2.4).

c) objects and encoding object sets from an EDM module.

12.1.9 The "EncodingApplicationList" is required to contain at least one "EncodingApplication", as the sole function of an ELM is to apply encodings.

12.2 Encoding types

12.2.1 An "EncodingApplication" is:

```
EncodingApplication ::=
        ENCODE
        SimpleDefinedEncodingClass "," +
        CombinedEncodings
```

12.2.2 An "EncodingApplication" defines the encoding of the ASN.1 types corresponding to the "SimpleDefinedEncodingClass"es which shall be generated encoding structures. The encoding of the types is specified by the "CombinedEncodings" applied to the generated encoding structures as specified in 13.2.

NOTE – It will be common for an ELM to encode a single type of a single module, but where multiple types are encoded, ECN tool-vendors may (but need not) assume that this implicitly identifies top-level types needing support in generated data-structures.

12.2.3 Encodings applied to a generated encoding structure corresponding to an ASN.1 type defined in some ASN.1 module are linked solely to the use of that type as application messages. They have no implications on the encoding of

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

that type when referenced by other types or when exported from that ASN.1 module and imported into a different ASN.1 module.

12.2.4 The encoding of the type in a contents constraint is that specified by the encoding object applied to the containing class in the octetstring or bitstring category, and can be any combined encoding object set, or can be the combined encoding object set that was applied to the containing class in the octetstring or bitstring category.

12.2.5 An ELM shall not apply encodings more than once to the same ASN.1 type.

NOTE – The rules of application of encodings (specified in clause 13) mean that an "EncodingApplication" completely defines the encoding of a type unless it contains an instance of a contents constraint.

13 Application of encodings

13.1 General

13.1.1 Encodings are applied by the ELM to a generated structure (or independently to multiple generated structures) using a "CombinedEncodings" definition as specified in 13.1.3. This clause, together with 13.2, specifies the application of "CombinedEncodings" to a generated encoding structure.

13.1.2 In the ELM, the application is to the generated encoding structures identified in the "EncodingApplication". Later clauses also specify the application of encodings to all or part of an arbitrary encoding structure definition. This clause is applicable in both cases.

13.1.3 The "CombinedEncodings" is:

```
CombinedEncodings ::=
                 WITH
                 PrimaryEncodings
                 CompletionClause ?

CompletionClause ::=
                 COMPLETED BY
                 SecondaryEncodings

PrimaryEncodings ::= EncodingObjectSet

SecondaryEncodings ::= EncodingObjectSet
```

13.1.4 "EncodingObjectSet" is defined in 18.1.1.

13.1.5 The use of "CombinedEncodings" is specified in 13.2.

13.2 The combined encoding object set and its application

13.2.1 A **combined encoding object set** is formed from the "CombinedEncodings" production (see 13.1.3) as follows:

13.2.2 If there is no "CompletionClause", then the "PrimaryEncodings" form the combined encoding object set.

13.2.3 Otherwise,

a) all encoding objects in the "PrimaryEncodings" are placed in the combined encoding object set; then

b) every encoding object in the "SecondaryEncodings" is added to the combined encoding object set if (and only if) there is no encoding object already in the combined encoding object set that has the same encoding class (see 17.1.7 and 9.23.2).

13.2.4 Following this conceptual construction of the combined encoding object set, encoding commences with the "encodingclassreference" name of the encoding structures identified in the encoding application (see 13.1.2 and 17.5).

13.2.5 Where there are several encoding applications in the ELM, the rules of 12.2 ensure that applications are non-overlapping. They proceed independently. Similarly, the application of encodings to encoding structures in EDMs (specified in 13.2.10) are always non-overlapping. The following subclauses provide the rules for application to a single encoding structure.

13.2.6 Encoding objects from the combined encoding object set are applied at an **application point**. The application point is initially the "encodingclassreference" for a generated encoding structure (when application is in the ELM, as specified in 13.1.2) or is a component of an encoding structure (when application is in an EDM, as specified in 17.5).

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

13.2.7 Any encoding class in the alternatives, concatenation, and repetition categories (see 16.1.8, 16.1.9 and 16.1.10) is an encoding constructor.

13.2.8 The term "component" in the following text refers to any of the following:

a) The alternatives of a constructor that is in the alternatives category.

b) The field following a constructor that is in the repetition category.

c) The components of a constructor that is in the concatenation category.

d) A contained type (a type specified in a contents constraint).

e) The type chosen (in an instance of communication) for use with a class in the opentype category.

13.2.9 At later stages in these procedures, the application point may be on any of the following:

a) An encoding class name. This is completely encodable using the specification in an encoding object of the same class (see 17.1.7).

b) An encoding constructor (see 16.2.12). The construction procedures can be determined by the specification contained in an encoding object of the encoding constructor class, but that encoding object does not determine the encoding of the components. The specification of the encoding object that is applied may require that one or more of the components of the constructor are replaced by other (parameterized) structures before the application point passes to the components.

c) A class in the bitstring or octetstring category that has a contained type as a property associated with the values (see 11.3.4.3 d). The encoding of the contained type depends on whether there is an **ENCODED BY** present, and on the specification of the encoding object being applied (see 22.11).

d) A class in the open type category. The encoding of the component of the open type depends on whether there is an **ENCODED WITH** present, and on the specification of the encoding object being applied (see 23.10.2).

e) A component which is an encoding class (possibly preceded by one or more classes in the tag category), followed by an encoding class in the optionality category. The procedures and encodings for determining presence or absence are determined by the specification contained in an encoding object of the class in the optionality category. This encoding object may also require the replacement of the encoding class (together with all its preceding classes in the tag category) with a (parameterized) replacement structure before that class is encoded. The application point then passes to the first class in the tag category (if any), or to the component, or to its replacement.

f) An encoding class preceded by an encoding class in the tag category. The tag number associated with the class in the tag category is encoded using the specification in an encoding object of the class in the tag category, and the application point then passes to the tagged class.

g) Any other built-in encoding class. This is completely encodable using the specification contained in an encoding object of that class.

13.2.10 Encoding proceeds as follows:

13.2.10.1 If the combined encoding object set contains an encoding object of the same class (see 17.1.7) as the current application point, then that encoding object is applied. This application may cause replacement of one or more components of the class to which the encoding is being applied. If the combined encoding object set does not contain such an encoding object, then either:

a) the encoding class at the current application point is a reference to another encoding class; in this case it is de-referenced, and the procedures of 13.2.10 are recursively applied; or

b) the encoding class at the current application point is not a reference to another encoding class; in this case the ECN specification is in error.

13.2.10.2 If an encoding has been applied at the application point to the encoding class, and it is not in the optionality or tag category and does not have any components (see 13.2.7), then that application completely determines the encoding of the class and terminates these procedures.

13.2.10.3 If an encoding has been applied at the application point to an encoding class that is in the optionality category then the application point passes to the (possibly tagged) optional component.

13.2.10.4 If an encoding has been applied at the application point to an encoding class that is in the tag category then the application point passes to the tagged element, and the procedures of 13.2.10 are recursively applied.

13.2.10.5 If an encoding has been applied at the application point to an encoding class that has components which are not a contained type, then the procedures of 13.2.10 are applied recursively to each component.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

NOTE – If the encoding object being applied to a class in the open type category contains an **ENCODED WITH**, this determines the encoding object set that is applied to the component, otherwise the combined encoding object set that is being applied to this class is applied to the component (see 23.10.2).

13.2.10.6 If an encoding has been applied to an encoding class at the application point that has a component that is a class in the bitstring or octetstring category with a contained type associated with the values, then there are four cases that can occur:

a) The contents constraint contains an **ENCODED BY**, and the encoding object for this class either does not contain a specification of the encoding of the contained type, or specifies that it should not override an **ENCODED BY** (see 22.11). In this case the **ENCODED BY** specification shall be used for the contained type, and the application point passes to the contained type using this encoding specification.

b) The contents constraint contains an **ENCODED BY**, but the encoding object for this class contains a specification of the encoding of the contained type, and specifies that it should override an "ENCODED BY". In this case, the specification in the encoding object shall be applied to the contained type, and the application point passes to the contained type using this encoding specification.

c) The contents constraint does not contain an **ENCODED BY** and the encoding object for this class contains a specification of the encoding of the contained type. In this case, the specification in the encoding object is applied to the contained type, and the application point passes to the contained type using this encoding specification.

d) The contents constraint does not contain an **ENCODED BY**, and the encoding object for this class does not contain a specification of the encoding of the contained type. In this case the combined encoding object set being applied to the class shall also be applied to the contents type, and the application point passes to the contained type using this encoding specification.

13.2.10.7 If there is no encoding object in the combined encoding object set of the same class (see 17.1.7) as the current application point, and the current application point is a reference name, then it is de-referenced and these procedures are applied recursively to the new encoding structure.

13.2.10.8 Otherwise the ECN specification is in error.

13.2.11 The above algorithm can be summarized as follows: The combined encoding object set is applied in a top-down manner. If in this process an encoding structure reference name is encountered and there is an object in the combined encoding object set that can encode it, that object determines its encoding. Otherwise, the reference name is expanded by de-referencing. If at any stage an encoding is required (and does not exist) for an encoding class that cannot be de-referenced, then the ECN specification is incorrect, and the combined encoding class is said to be incomplete. When a primitive bit-field class is reached, the encoding terminates with the encoding of that class, except that if it has a contained type, encoding proceeds to the generated encoding structure corresponding to the contained type. When a type with components is reached, the process continues by applying the combined encoding object set to each component independently. When tags and optionality are involved, the optionality class is encoded first, then the encoding class in the tag category, and finally the element. When encodings are applied to constructor classes they may cause replacement of one or more components. When they are applied to an optionality class they may cause replacement of the entire element (apart from the optionality class, but including any encoding class in the tag category).

13.2.12 In the encoding process, encoding objects applied to encoding constructors (and to classes in the optionality category) may require that the encoding objects applied to the components of the constructions defined by those constructors exhibit identification handles (of a given name) to resolve alternatives, or optionality, or termination of a repetition, or order in a set-like concatenation. They may also require that the encoding objects applied to other encoding classes (following those constructions) exhibit the same identification handle, and that the handle value sets of all the involved encoding objects (exhibiting the same handle) be all disjoint. If these conditions are not satisfied, then the ECN specification is in error.

NOTE – This problem is most likely to arise if BER encoding objects are applied to encoding constructors and not to their components, as BER is heavily reliant on identification handles. PER encoding objects make no use of identification handles.

14 The Encoding Definition Module (EDM)

NOTE – There are two top-level productions in ECN, the "EDMDefinition" specified in this clause and the "ELMDefinition" specified in clause 12. These specify the syntax for defining EDMs and the ELM respectively.

14.1 The "EDMDefinition" is:

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

```
EDMDefinition ::=
        ModuleIdentifier
        ENCODING-DEFINITIONS
        "::="
        BEGIN
        EDMModuleBody
        END
```

14.2 In any given application of ECN, there are zero, one or more EDMs which define encoding objects for application in the ELM.

NOTE – If there are zero EDMs, then only built-in encoding object sets can be used in the ELM.

14.3 The production "ModuleIdentifier" (and its semantics) is defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.1.

14.4 The "ModuleIdentifier" provides unambiguous identification of any module in the set of all ASN.1, ELM, and EDM modules.

14.5 The "EDMModuleBody" is:

```
EDMModuleBody ::=
        Exports ?
        RenamesAndExports ?
        Imports ?
        EDMAssignmentList ?

EDMAssignmentList ::=
        EDMAssignment
        EDMAssignmentList ?


EDMAssignment ::=
                EncodingClassAssignment
        |       EncodingObjectAssignment
        |       EncodingObjectSetAssignment
        |       ParameterizedAssignment
```

14.6 The productions "Exports" and "Imports" (and their semantics) are defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.1, as modified by A.1 of this Recommendation | International Standard.

14.7 The "Exports" makes available for import into other EDMs (and the ELM) any reference name defined in or imported into the current EDM except that of an implicitly generated structure. The "Symbol" in the "Exports" can reference any encoding class (except a built-in encoding class or an implicitly generated structure), an encoding object, or an encoding object set. The "Symbol" shall have been defined in this EDM, or imported into it.

NOTE – When the name of an imported implicitly generated encoding structure is a built-in encoding class reference, it can be used within the EDM with a fully-qualified name. An implicitly generated encoding structure can never be exported from an EDM (however, encoding structures defined using it can, of course, be exported).

14.8 The production "RenamesAndExports" is defined in clause 15.

14.9 The "RenamesAndExports" (called the renames clause) makes available (within the EDM) explicitly generated encoding structures derived from the implicitly generated encoding structures in specified ASN.1 modules. It also makes these explicitly generated encoding structures available for import into other EDMs (and the ELM). (See clause 15.)

14.10 The "Imports" makes available (within the EDM) encoding classes, encoding objects and encoding object sets exported from other EDMs or automatically exported from ASN.1 modules.

14.11 All ASN.1 modules that define non-parameterized type reference names automatically produce and export an implicitly generated encoding structure of the same name preceded by the character "**#**". Such encoding classes can be imported into an EDM from that ASN.1 module.

NOTE – Where such names are the same as built-in encoding class names, then the external form of reference, as specified in A.1, has to be used in the body of the importing module, and in any renames clause.

14.12 Each "EDMAssignment" defines a reference name, and may make use of other reference names. Each reference name used in a module shall either be imported into that module or shall be defined precisely once within that module.

NOTE – This is a stronger requirement than that imposed for ASN.1 modules. In ASN.1 modules, external references can be used for types and values that have not been imported. In an EDM module (and in an ELM module) external references can only be used for encoding classes that have been referenced in an imports clause. The purpose of external references is solely to resolve ambiguities between imported names and built-in names, or between two identical names imported from different modules.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

14.13 There is no requirement that any reference name used in one assignment be defined (in another assignment statement) textually before its use.

14.14 The productions in "EDMAssignment" are defined in subsequent clauses as follows:

EncodingClassAssignment	Clause 16
EncodingObjectAssignment	Clause 17
EncodingObjectSetAssignment	Clause 18
ParameterizedAssignment	Subclause C.1

NOTE – The "ParameterizedAssignment" allows the parameterization of an "EncodingClassAssignment", an "EncodingObjectAssignment", and an "EncodingObjectSetAssignment", as specified in C.1.

15 The renames clause

15.1 Explicitly generated and exported structures

15.1.1 The production "RenamesAndExports" is:

```
RenamesAndExports ::=
                        RENAMES
                        ExplicitGenerationList ";"

ExplicitGenerationList ::=
                        ExplicitGeneration
                        ExplicitGenerationList ?

ExplicitGeneration ::=
                        OptionalNameChanges
                        FROM GlobalModuleReference

OptionalNameChanges ::=
                        NameChanges |  GENERATES
```

NOTE – An example of the use of the renames clause to produce explicitly generated encoding structures is given in D.3.7.

15.1.2 The production "GlobalModuleReference" is defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 13.1, and shall identify an ASN.1 module.

15.1.3 The "RenamesAndExports" is called a renames clause.

15.1.4 Each "ExplicitGeneration" generates, and exports from this module, an explicitly generated encoding structure for each of the implicitly generated encoding structures of the ASN.1 module referenced by "GlobalModuleReference". Each field of the explicitly generated encoding structure has associated with it the same abstract values as the corresponding field of the implicitly generated encoding structure (which are those associated with the corresponding field of the ASN.1 type from which it was generated).

15.1.5 If a renames clause references more than one ASN.1 module and as a result of this two explicitly generated structures have the same simple name, then neither structure is available for explicit import into an ELM or an EDM module.

NOTE – These explicitly generated structures nonetheless exist, and are likely to be implicitly referenced by other explicitly generated structures that are exported without restriction.

15.1.6 The primary purpose of the renames clause is to make available the explicitly generated structures for import into other modules, particularly the ELM. However, this clause also makes these structures available for reference within the EDM module containing the renames clause except as specified in 15.1.7. If the simple name is ambiguous, then a fully-qualified name shall be used within the EDM module containing the renames clause, as specified in 15.1.9.

NOTE – Ambiguity can arise either because of clashes with the names of built-in classes, or because of clashes of simple names between structures generated from more than one ASN.1 module, or both.

15.1.7 When a renames clause produces an explicitly generated structure from an implicitly generated structure, that implicitly generated structure cannot be imported into this EDM module using an imports clause, and the implicitly generated structure is never available in this EDM module.

15.1.8 These explicitly generated encoding structures have the same simple reference name as the implicitly generated encoding structure from which they were formed (but are distinct classes). Where a fully-qualified name is required for an explicitly generated encoding structure, that fully-qualified name shall include the "ModuleIdentifier" of the EDM module containing the renames clause, as specified in 15.1.9.

NOTE – The implicitly generated encoding structures used in their generation have the same simple reference name, but their fully-qualified name includes the "ModuleIdentifier" of the ASN.1 module in which the corresponding type was defined.

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

15.1.9 If an EDM produces explicitly generated encoding structures from more than one ASN.1 module, it is possible that some of these structures may have the same simple encoding class names. If any of these structures are referenced in the body of this EDM, then the reference shall be an "ExternalEncodingClassReference" containing the "modulereference" used as the ASN.1 module reference in the replaces clause of this EDM module.

15.1.10 The "ExternalEncodingClassReference" notation shall not be used in an imports clause except where required by clause 15.1.9.

15.1.11 If a name which has been imported using an "ExternalEncodingClassRefererence" is used in the body of a module, then the simple "encodingclassreference" can be used unless an "ExternalEncodingClassReference" is required as specified in clause 15.1.9.

15.1.12 If the "OptionalNameChanges" is **GENERATES**, then all the explicitly generated encoding structures are the same structure as the implicitly generated encoding structures used in their generation, except as specified in 15.1.14.

NOTE – (Tutorial) If, in an EDM module, there are multiple structures with the same simple reference name (whether these names arise from an imports clause or from a renames clause, or from clashes with built-in classes, or from any combination of these), then a fully-qualified name is used except for references to a built-in class. For implicitly generated structures, the fully-qualified name always uses the ASN.1 module name. For structures generated by the renames clause in an EDM module, the fully-qualified name is used. This fully-qualified name in the body of this EDM always uses the ASN.1 module name referenced by the renames clause. For structures imported from another EDM module, the fully-qualified name uses the name of that EDM module. This is always unambiguous, as importation is not permitted if an EDM module generates multiple explicitly generated structures with the same simple reference name.

15.1.13 If "OptionalNameChanges" is "NameChanges", then 15.1.14 still applies, but the explicitly generated encoding structures are further modified as specified in 15.2.

15.1.14 Consider an implicitly generated encoding structure (A say) which contains an encoding class reference to some other implicitly generated encoding structure (B say). Then:

a) If this renames clause (in any of its "ExplicitGeneration"s) produces an explicitly generated encoding structure corresponding to B (B1 say), then the corresponding reference in the explicitly generated encoding structure corresponding to A is a reference to B1.

b) If there is no explicitly generated encoding structure corresponding to B, then the reference in the generated encoding structure corresponding to A is a reference to B.

15.2 Name changes

15.2.1 The "NameChanges" production is:

```
NameChanges ::=
        NameChange
        NameChanges ?

NameChange ::=
        OriginalClassName
        AS
        NewClassName
        IN
        NameChangeDomain

OriginalClassName ::= SimpleDefinedEncodingClass | BuiltinEncodingClassReference

NewClassName ::= encodingclassreference
```

15.2.2 Each "NameChanges" specifies that, in the generation of explicitly generated encoding structures, all occurrences of "OriginalClassName" within "NameChangeDomain" in the implicitly generated encoding structures are to be renamed as the class "NewClassName". "NameChangeDomain" is specified in 15.3, and identifies one or more implicitly generated encoding structures (or components of those structures) from the ASN.1 module referenced by the "GlobalModuleReference" in the "ExplicitGeneration".

NOTE 1 – This enables different encodings to be applied to some occurrences of a class from that applied to other occurrences.

NOTE 2 – This implies that "OriginalClassName" can only be a name implicitly generated from an ASN.1 type, that is, the name of a user-defined ASN.1 type (preceded by "**#**"), or one of the class names listed in column 2 of Table 2.

15.2.3 References by "OriginalClassName" to fields of the implicitly generated encoding structure which correspond to use of "ExternalTypeReference" in the ASN.1 type definition shall use the "SimpleDefinedEncodingClass" notation with the same "modulereference" as the "ExternalTypeReference". Otherwise, if the "DefinedType" (preceded by a "**#**") is not a "BuiltinEncodingClassReference", a simple "encodingclassreference" shall be used. If a "typereference" (preceded by a "**#**") is a "BuiltinEncodingClassReference" then the "SimpleDefinedEncodingClass" notation shall be used with the same "modulereference" as the ASN.1 module that generated the implicitly generated encoding structure.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

15.2.4 When an ELM imports an explicitly generated encoding structure from an EDM, renames clauses in other EDMs have no effect on the encoding of that structure.

NOTE – This means in practice that all the "coloring" (see 9.16.4) needed for any particular message has to be done in a single EDM.

15.2.5 The "NewClassName" shall be defined in an encoding class assignment statement (see clause 16) of the form:

```
<NewClassName> ::= <OriginalClassName>
```

where "*<NewClassName>*" and "*<OriginalClassName>*" are the names of the new and original classes appearing in the "NameChanges" production. The assignment shall be in the EDM module with the renames clause.

NOTE – The "*<OriginalClassName>*" is required to reference a built-in encoding class or an externally generated encoding structure produced by the renames clause in this module. In case of ambiguity, this will require the use of an external reference in "*<OriginalClassName>*".

15.3 Specifying the region for name changes

15.3.1 The production "NameChangeDomain" is:

```
NameChangeDomain ::=
        IncludedRegions
        Exception ?

Exception ::=
        EXCEPT
        ExcludedRegions

IncludedRegions ::=
        ALL | RegionList

ExcludedRegions ::= RegionList

RegionList ::=
        Region "," +

Region ::=
        SimpleDefinedEncodingClass |
        ComponentReference

ComponentReference ::=
        SimpleDefinedEncodingClass
        "."
        ComponentIdList

ComponentIdList ::=
        identifier "." +
```

15.3.2 Each "SimpleDefinedEncodingClass" shall be the name of an implicitly generated encoding structure from the ASN.1 module referenced by the "GlobalModuleReference" in the "ExplicitGeneration". When used in "Region", it identifies the whole of that encoding structure definition.

NOTE – The "ExternalEncodingClassReference" form of "SimpleDefinedEncodingClass" is used if the referenced class is derived from a "typereference" name which (when preceded by "**#**") is a "BuiltinEncodingClassReference" (see 15.2.3).

15.3.3 Each "identifier" shall be the "identifier" in a "NamedField" of the implicitly generated encoding structure identified by the "encodingclassreference" in the "ComponentReference". The "ComponentReference" identifies the entire definition of the identified component of that encoding structure.

15.3.4 The first "identifier" of the "ComponentIdList" shall be an "identifier" in a "NamedField" of the implicitly generated encoding structure identified by the "encodingclassreference" in the "ComponentReference", and identifies the entire definition of that component of the encoding structure. Each subsequent "identifier" of the "ComponentIdList" shall be an "identifier" in a "NamedField" of the implicitly generated encoding structure identified by the previous part of the "ComponentIdList", and identifies the entire definition of that component.

15.3.5 The definitions identified by different "Region"s in "RegionList" shall be disjoint. A definition is identified by "RegionList" if and only if it is identified by a "Region" in "RegionList".

15.3.6 If "IncludedRegions" is **ALL**, it identifies all parts of all the implicitly generated encoding structures from the ASN.1 module referenced by the "GlobalModuleReference" in the "ExplicitGeneration".

15.3.7 The definitions identified by the "ExcludedRegions" shall be a proper subset of the definitions identified by the "IncludedRegions".

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021

15.3.8 The "NameChangeDomain" specification identifies the definitions in which the name changes are to be made. The definitions in the "NameChangeDomain" are the definitions identified by the "IncludedRegions" which are not also identified by "ExcludedRegions".

16 Encoding class assignments

16.1 General

16.1.1 The "EncodingClassAssignment" is:

```
EncodingClassAssignment ::=
                    encodingclassreference
                    "::="
                    EncodingClass
```

16.1.2 The "EncodingClassAssignment" assigns the "EncodingClass" to the "encodingclassreference".

NOTE – Any "EncodingObject" notation that was valid with "EncodingClass" as a governor is valid with "encodingclassreference" as a governor.

16.1.3 An encoding class is in one of the following categories:

a) A category in the bit-field group of categories (see 16.1.7).

b) The alternatives category (see 16.1.8).

c) The concatenation category (see 16.1.9).

d) The repetition category (see 16.1.10).

e) The optionality category (see 16.1.11).

f) The tag category (see 16.1.12).

g) A category in the encoding procedure group of categories (see 16.1.13).

NOTE – The term encoding constructor is used for any class in the alternatives, concatenation, and repetition categories. These are also called the encoding constructor group of categories.

16.1.4 The category of each built-in encoding class is specified in 16.1.14.

NOTE – If an encoding class is a tagged class (see 16.2.1), or has bounds (see 16.2.6), then the category of the class is the category of the class with the tag and the bounds removed.

16.1.5 The "EncodingClass" is:

```
EncodingClass ::=
                    BuiltinEncodingClassReference
        |           EncodingStructure
```

16.1.6 The "BuiltinEncodingClassReference" is:

```
BuiltinEncodingClassReference ::=
                    BitfieldClassReference
        |           AlternativesClassReference
        |           ConcatenationClassReference
        |           RepetitionClassReference
        |           OptionalityClassReference
        |           TagClassReference
        |           EncodingProcedureClassReference
```

16.1.7 The "BitfieldClassReference" is:

```
BitfieldClassReference ::=
```


© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 8825-3:2021