INTERNATIONAL STANDARD

ISO/IEC 8825-3

First edition
2002-12-15

AMENDMENT 1
2005-11-15

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

AMENDMENT 1: Extensibility support

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

AMENDEMENT 1: Prise en charge de l'extensibilité

Reference number
ISO/IEC 8825-3:2002/Amd.1:2005(E)

CONTENTS

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

Amendment 1 to International Standard ISO/IEC 8825-3:2002 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 Amendment 1 to ITU-T Rec. X.692.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

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

Amendment 1

Extensibility support

NOTE – All new or changed text in this amendment is underlined in clauses being replaced. When new clauses with a heading are inserted, only the heading is underlined. Deleted text is present but marked with a strike-through. When merging all such text into the base Recommendation | International standard, the underlining is to be removed and struck-through text taken out.

1 Subclause 3.2.8

Replace 3.2.8 with:

3.2.8 conditional encoding: An encoding which is to be applied only if some specified condition ~~bounds condition or size range condition~~ is satisfied.

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

2 New subclauses 9.25 *bis* and 9.25 *ter*

Insert new clauses 9.25 bis and 9.25 ter and update the contents:

9.25 *bis* Other conditions for applying encodings

9.25 *bis*.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.

9.25 *bis*.2 It is also possible to require that all of a given list of conditions are to be satisfied.

9.25 *bis*.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.

9.25 *bis*.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.

9.25 *ter* Encoding control for the open type

9.25 *ter*.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).

9.25 *ter*.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.

9.25 *ter*.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.

9.25 *ter.*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.

9.25 *ter.*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.

9.25 *ter.*6 The application may, however, say "unknown" (see 9.25 *ter.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.

3 Subclause 13.2.9

Replace subclause 13.2.9 with:

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 23.9 *bis*.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.

4 Subclause 13.2.10.5

Replace the Note in 13.2.10.5 with:

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 23.9 *bis*.2).

5 Subclause 17.5.15

Replace 17.5.15 with:

17.5.15 If a **REFERENCE** is needed as an actual parameter of any of the encoding objects or encoding object sets used in this production, then it can either be supplied as a dummy parameter of the encoding object that is being defined, or it can be supplied as a "ComponentIdList" (see 15.3.1 for the syntax of the "ComponentIdList" – the meaning of the "ComponentIdList" in this context is specified below).

17.5.15 *bis* If the governor is not a constructor in the repetition category, then the first (or only) "identifier" in the "ComponentIdList" shall be the "identifier" of a textually present "NamedType" (at some level of nesting – see 17.5.15 *ter*) of the construction that is obtained by de-referencing the governor. It identifies the entire definition of that "NamedType" component, whether that definition is textually present or not.

17.5.15 *ter* If there is more than one such matching identifier, then the chosen matching identifier shall be determined by the first match in a scan (in textual order) of the outer-level identifiers, then by a scan (in textual order) of the second level identifiers, then by a scan (in textual order) of the third-level identifiers, and so on.

17.5.15 *quat* Each subsequent "identifier" of the "ComponentIdList" (if any) shall be an "identifier" in a "NamedType" of the structure identified by the previous part of the "ComponentIdList", and identifies the entire definition of that "NamedType" component, whether it is textually present or not in the definition of the structure identified by the previous part of the "ComponentIdList".

17.5.15 *quin* If the governor is a constructor in the repetition category, then the actual parameter for the **REFERENCE** shall be a "ComponentIdList" whose first "identifier" identifies a component that is textually present in the "EncodingStructure" in the "RepetitionStructure" obtained by de-referencing the repetition (see 17.5.15 *ter*). Subclauses 17.5.15 *ter* and 17.5.15 *quat* then apply.

17.5.15 *sex* If the **REFERENCE** is required to identify a container, it can also be supplied as:

a) **STRUCTURE** (provided the constructor for the structure being encoded is not an alternatives category) when it refers to that structure;

b) **OUTER** when it refers to the container of the complete encoding.

NOTE – The "EncodeStructure" is the only production in which **REFERENCE**s can be supplied, except through the use of dummy parameters or the use of **OUTER**, or where references are in support of **flag-to-be-used** or **flag-to-be-set** in the definition of an encoding object for a class in the repetition category which uses replacement.

6 Subclause 18.2.6

Replace the Note in 18.2.6 with:

NOTE – The combined encoding object set applied by these encoding objects to the type chosen for use with the **#OPEN-TYPE** class is always the same as the combined encoding object set applied to the **#OPEN-TYPE** class as these encoding objects do not contain an **ENCODED WITH** (see 13.2.10.5 and 13.2.9 d).

7 Subclause 21.11.1

Replace 21.11.1 with:

21.11.1 The "**RangeCondition**" type is:

```
RangeCondition ::= ENUMERATED
                 {unbounded-or-no-lower-bound,
                  semi-bounded-with-negatives,
                  bounded-with-negatives,
                  semi-bounded-without-negatives,
                  bounded-without-negatives,
                  test-lower-bound,
                  test-upper-bound,
                  test-range}
```

8 Subclause 21.11.4

Replace 21.11.4 with:

21.11.4 The predicate is satisfied for each of the first five enumeration values of 21.11.1 if and only if the following conditions are satisfied by the bounds on the encoding class in the integer category:

a) **unbounded-or-no-lower-bound**: either there are no bounds, or else there is only an upper bound but no lower bound.

b) `semi-bounded-with-negatives`: there is a lower bound that is less than zero, but no upper bound.

c) `bounded-with-negatives`: there is a lower bound that is less than zero, and an upper bound.

d) `semi-bounded-without-negatives`: there is a lower bound that is greater than or equal to zero, but no upper bound.

e) `bounded-without-negatives`: there is a lower bound that is greater than or equal to zero, and an upper bound.

NOTE – For any given set of bounds, exactly one predicate will be satisfied.

9 New subclause 21.11.5

Add a new subclause 21.11.5:

21.11.5 If the last three enumeration values of 21.11.1 are used, a value of the "`Comparison`" type (see 21.11 *bis*) shall be provided, together with an integer `comparator` value. If the other enumeration values are used, these shall not be provided.

10 New subclause 21.11 *bis*

Add a new subclause 21.11 bis after 21.11 and add to the contents list:

<u>21.11 *bis* The `Comparison` type</u>

21.11 *bis*.1 The "`Comparison`" type is:

```
Comparison ::= ENUMERATED
             {equal-to,
              not-equal-to,
              greater-than,
              less-than,
              greater-than-or-equal-to,
              less-than-or-equal-to}
```

21.11 *bis*.2 There is no default value for an encoding property of this type.

21.11 *bis*.3 An encoding property of type "`Comparison`" is used to test an identified property of a class against an integer value (the `comparator`).

21.11 *bis*.4 The predicate using a "`Comparison`" is satisfied for each enumeration value if and only if the identified property satisfies the following conditions:

a) `equal-to`: its value equals that of the specified integer `comparator` value.

b) `not-equal-to`: its value is different from that of the specified integer `comparator` value.

c) `greater-than`: its value is greater than that of the specified integer `comparator` value.

d) `less-than`: its value is less than that of the specified integer `comparator` value.

e) `greater-than-or-equal-to`: its value is greater than or equal to that of the specified integer `comparator` value.

f) `less-than-or-equal-to`: its value is less than or equal to that of the specified integer `comparator` value.

11 Subclause 21.12.1

Replace 21.12.1 with:

21.12.1 The "`SizeRangeCondition`" type is:

```
SizeRangeCondition ::= ENUMERATED
             {no-ub-with-zero-lb,
              ub-with-zero-lb,
              no-ub-with-non-zero-lb,
              ub-with-non-zero-lb,
              fixed-size,
              test-lower-bound,
              test-upper-bound,
              test-range}
```

12 Subclause 21.12.4

Replace 21.12.4 with:

21.12.4 The predicate is satisfied for each of the first five enumeration values of 21.12.1 if and only if the effective size constraint satisfies the following conditions:

a) **`no-ub-with-zero-lb`**: there is no upper bound on the size and the lower bound is zero.

b) **`ub-with-zero-lb`**: there is an upper bound on the size and the lower bound is zero.

c) **`no-ub-with-non-zero-lb`**: there is no upper bound on the size and the lower bound is non-zero.

d) **`ub-with-non-zero-lb`**: there is an upper bound on the size and the lower bound is non-zero.

e) **`fixed-size`**: the lower bound and the upper bound on the size are the same value.

NOTE – Only the "**`fixed-size`**" case overlaps with other predicates.

13 New subclause 21.12.5

Add a new subclause 21.12.5 after 21.12.4:

21.12.5 If the last three enumeration values of 21.12.1 are used, a value of the "**`Comparison`**" type (see 21.11 *bis*) shall be provided, together with an integer **`comparator`** value. If the other enumeration values are used, these shall not be provided.

14 New subclause 21.16

Add a new subclause 21.16:

21.16 The `IntegerMapping` type

21.16.1 The "**`IntegerMapping`**" type is:

```
IntegerMapping ::= SET OF SEQUENCE {
               source  SET OF INTEGER,
               result INTEGER} (CONSTRAINED BY {/* the intersection of the source
                                                  components shall be empty */})
```

21.16.2 The "**`IntegerMapping`**" is used to specify explicitly an ints-to-ints transform.

15 Subclause 23.2.3.8

Replace 23.2.3.8 with:

23.2.3.8 If an encoding object in the "**`REPETITION-ENCODINGS`**" ordered list is defined using "**`IF`**" or "**`IF-ALL`**", then all preceding encoding objects in that list shall be defined using "**`IF`**" or "**`IF-ALL`**".

16 Subclause 23.4.3.8

Replace 23.4.3.8 with:

23.4.3.8 If an encoding object in the "**`REPETITION-ENCODINGS`**" ordered list is defined using "**`IF`**" or "**`IF-ALL`**", then all preceding encoding objects in that list shall be defined using "**`IF`**" or "**`IF-ALL`**".

17 Subclause 23.6.2.3

Replace 23.6.2.3 with:

23.6.2.3 If an encoding object in the "`ENCODINGS`" ordered list is defined using "`IF`" or "`IF-ALL`", then all preceding encoding objects in that list shall be defined using "`IF`" or "`IF-ALL`".

18 Subclause 23.7.1

Replace 23.7.1 with:

23.7.1 The defined syntax

The syntax for defining encoding objects for the `#CONDITIONAL-INT` class is defined as:

```
#CONDITIONAL-INT ::= ENCODING-CLASS {

             -- Condition (see 21.11)
             &range-condition                  RangeCondition  OPTIONAL,
             &comparison                       Comparison OPTIONAL,
             &comparator                       INTEGER OPTIONAL,
             &Range-conditions                 RangeCondition ORDERED OPTIONAL,
             &Comparisons                      Comparison ORDERED OPTIONAL,
             &Comparators                      INTEGER ORDERED OPTIONAL,

             -- Structure-only replacement specification (see 22.1)
             &#Replacement-structure           OPTIONAL,
             &replacement-structure-encoding-object  &#Replacement-structure OPTIONAL,

             -- Pre-alignment and padding specification (see 22.2)
             &encoding-space-pre-alignment-unitUnit (ALL EXCEPT repetitions)
                                               DEFAULT bit,
             &encoding-space-pre-padding       Padding DEFAULT zero,
             &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT
                                                      different:any) DEFAULT bits:'0'B,

             -- Start pointer specification (see 22.3)
             &start-pointer                    REFERENCE     OPTIONAL,
             &start-pointer-unit               Unit (ALL EXCEPT repetitions)
                                               DEFAULT bit,
             &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

             -- Encoding space specification (see 22.4)
             &encoding-space-size              EncodingSpaceSize
                                               DEFAULT self-delimiting-values,
             &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                               DEFAULT bit,
             &encoding-space-determination     EncodingSpaceDetermination
                                               DEFAULT field-to-be-set,
             &encoding-space-reference         REFERENCE OPTIONAL,
             &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
             &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

             -- Value encoding
             &Transform                        #TRANSFORM ORDERED OPTIONAL,
             &encoding                         ENUMERATED
                                               {positive-int, twos-complement,
                                                reverse-positive-int,
                                                reverse-twos-complement}
                                               DEFAULT twos-complement,

             -- Value padding and justification (see 22.8)
             &value-justification              Justification DEFAULT right:0,
             &value-pre-padding                Padding DEFAULT zero,
             &value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
             &value-post-padding               Padding DEFAULT zero,
             &value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
             &unused-bits-determination        UnusedBitsDetermination
                                               DEFAULT field-to-be-set,
             &unused-bits-reference            REFERENCE OPTIONAL,
             &Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
             &Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

             -- Identification handle specification (see 22.9)
&exhibited-handle                  PrintableString OPTIONAL,
&Handle-positions                  INTEGER (0..MAX) OPTIONAL,
&Handle-value                      HandleValue DEFAULT tag:any,

-- Bit reversal specification (see 22.12)
&bit-reversal                      ReversalSpecification
                                   DEFAULT no-reversal
}
WITH SYNTAX {
[IF &range-condition [&comparison  &comparator]]
[IF-ALL &Range-conditions [&Comparisons &Comparators]]
[ELSE]
[REPLACE
      [STRUCTURE]
      WITH &#Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object]]
[ALIGNED TO
      [NEXT]
      [ANY]
            &encoding-space-pre-alignment-unit
            [PADDING &encoding-space-pre-padding
            [PATTERN &encoding-space-pre-pattern]]]
[START-POINTER   &start-pointer
      [MULTIPLE OF     &start-pointer-unit]
      [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
ENCODING-SPACE
      [SIZE &encoding-space-size
            [MULTIPLE OF &encoding-space-unit]]
      [DETERMINED BY &encoding-space-determination]
      [USING &encoding-space-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
[TRANSFORMS &Transforms]
[ENCODING &encoding]
[VALUE-PADDING
      [JUSTIFIED &value-justification]
      [PRE-PADDING &value-pre-padding
            [PATTERN &value-pre-pattern]]
      [POST-PADDING &value-post-padding
            [PATTERN &value-post-pattern]]
      [UNUSED BITS
            [DETERMINED BY &unused-bits-determination]
            [USING &unused-bits-reference
                  [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                  [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
      [AS &handle-value]]
[BIT-REVERSAL &bit-reversal]
}
```

19 Subclause 23.7.2.2

Replace 23.7.2.2 with:

23.7.2.2 The syntax allows the specification of a single condition on the bounds of the integer for this encoding to be applied (use of "**IF**"). It also allows the specification that all of a set of conditions are to be satisfied (use of "**IF-ALL**"). It also allows the specification that there is no condition. The use of "**ELSE**", or omission of ~~both~~ "**IF**", "**IF-ALL**" and "**ELSE**" specifies that there is no condition. "**IF-ALL**" shall be used with three lists if one or more of the size-range-conditions require a comparison, and shall be used with one list otherwise. When using three lists, size-range-conditions that do not require a comparison or comparator (if any) shall follow all those that require a comparison, and shall have no corresponding entry in the second and third lists. In using "**IF-ALL**" with three lists, the lists shall be interpreted as a list of predicates using the values in corresponding positions in the three lists.

NOTE – It is recommended that the three lists be formatted to provide a condition in each column.

EXAMPLE:

```
IF-ALL {test-lower-bound, test-range    , bounded-with-negatives }
             {greater-than    , less-than-or-equal-to }
             {-10             , 20                    }
```

20 Subclause 23.7.2.4

Replace 23.7.2.4 with:

23.7.2.4 At most one of "**IF**", "**IF-ALL**" and "**ELSE**" shall be present.

21 Subclauses 23.7.2.6, 23.7.2.7 and 23.7.2.8

Replace 23.7.2.6, 23.7.2.7 and 23.7.2.8 with:

23.7.2.6 It is an ECN specification or application error if any transform in the "**TRANSFORMS**" is not reversible for the abstract value to which it is applied. The first transform of "**TRANSFORMS**", if present, shall have a source that is integer and the last transform shall have a result that is integer.

NOTE – The tests for the "**IF**" and "**IF-ALL**" conditions take~~s~~ place on the bounds of the original value, and are~~is~~ not affected by these transforms.

23.7.2.7 The "**INT-TO-INT**" transform with the value "**subtract:lower-bound**" shall be included only if the "**IF**" or "**IF-ALL**" condition restricts the application of this encoding to classes of the integer category with a lower bound, and (if present) shall be the first transform in the list.

23.7.2.8 The "**ENCODING-SPACE SIZE**" shall not be "**fixed-to-max**" unless the "**IF**" or "**IF-ALL**" condition restricts the encoding to a class with both an upper and a lower bound.

22 Subclause 23.9.3.8

Replace 23.9.3.8 with:

23.9.3.8 If an encoding object in the "**REPETITION-ENCODINGS**" ordered list is defined using "**IF**" or "**IF-ALL**", then all preceding encoding objects in that list shall be defined using "**IF**" or "**IF-ALL**".

23 New subclause 23.9 *bis*

Insert the new subclause 23.9 bis after 23.9 and update the contents:

23.9 *bis* Defining encoding objects for classes in the open type category

23.9 *bis*.1 The defined syntax

The syntax for defining encoding objects for classes in the open type category is defined as:

```
#OPEN-TYPE ::= ENCODING-CLASS {

             -- Structure-only replacement specification (see 22.1)
             &#Replacement-structure           OPTIONAL,
             &replacement-structure-encoding-object
                                               &#Replacement-structure OPTIONAL,

             -- Pre-alignment and padding specification (see 22.2)
             &encoding-space-pre-alignment-unit
                                               Unit (ALL EXCEPT repetitions)
                                               DEFAULT bit,
             &encoding-space-pre-padding       Padding DEFAULT zero,
             &encoding-space-pre-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                               DEFAULT bits:'0'B,

             -- Start pointer specification (see 22.3)
             &start-pointer                    REFERENCE OPTIONAL,
             &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
             &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

             -- Encoding space specification (see 22.4)
             &encoding-space-size              EncodingSpaceSize
                                               DEFAULT self-delimiting-values,
             &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                               DEFAULT bit,
             &encoding-space-determination     EncodingSpaceDetermination
```

```
                                  DEFAULT field-to-be-set,
&encoding-space-reference         REFERENCE OPTIONAL,
&Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,

-- Open-type encoding
&Known-structure-encodings        #ENCODINGS OPTIONAL,
&Unknown-structure                OPTIONAL,
&Unknown-structure-encodings      #ENCODINGS OPTIONAL,

-- Value padding and justification (see 22.8)
&value-justification              Justification DEFAULT right:0,
&value-pre-padding                Padding DEFAULT zero,
&value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
&value-post-padding               Padding DEFAULT zero,
&value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
&unused-bits-determination        UnusedBitsDetermination
                                  DEFAULT field-to-be-set,
&unused-bits-reference            REFERENCE OPTIONAL,
&Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
&Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

-- Bit reversal specification (see 22.12)
&bit-reversal                     ReversalSpecification
                                  DEFAULT no-reversal
}
WITH SYNTAX {
[REPLACE
      [STRUCTURE]
      WITH &#Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object]]
[ALIGNED TO
      [NEXT]
      [ANY]
            &encoding-space-pre-alignment-unit
            [PADDING &encoding-space-pre-padding
            [PATTERN &encoding-space-pre-pattern]]]
[START-POINTER &start-pointer
      [MULTIPLE OF     &start-pointer-unit]
      [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
ENCODING-SPACE
      [SIZE &encoding-space-size
            [MULTIPLE OF &encoding-space-unit]]
      [DETERMINED BY &encoding-space-determination]
      [USING &encoding-space-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
[ENCODED WITH &Known-structure-encodings]
[UNKNOWN IS &Unknown-structure
      [ENCODED WITH &Unknown-structure-encodings]]
[VALUE-PADDING
      [JUSTIFIED &value-justification]
      [PRE-PADDING &value-pre-padding
            [PATTERN &value-pre-pattern]]
      [POST-PADDING &value-post-padding
            [PATTERN &value-post-pattern]]
      [UNUSED BITS
            [DETERMINED BY &unused-bits-determination]
            [USING &unused-bits-reference
                  [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                  [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
      [AS &handle-value]
[BIT-REVERSAL &bit-reversal]
}
```

23.9 *bis*.2 Model for the encoding of classes in the open type category

23.9 *bis*.2.1 The model of open type encodings is:

a) The class in the open type category can be replaced by another structure to provide length delimitation if required.

b) The encoding object defined for this category applies the "`ENCODED WITH`" encoding object set to the type whose value is to be encoded for the open type. If there is no "`ENCODED WITH`", then the current combined encoding object set is used.

c) The decoder will request the application for identification of the type encoded into the open type. The application will either respond with identification of the type, which is then decoded, or will state that the type encoded in the open type cannot be determined (an "unknown" response).

d) If the response is "unknown" and the "`UNKNOWN IS`" is present, then the decoder will use the "`UNKNOWN IS`" structure and the "`ENCODED WITH`" within the "`UNKNOWN IS`" (if present) to determine the end of the encoding space.

e) If the response is "unknown" and the "`UNKNOWN IS`" is absent, then the encoding space size can be determined by the "`ENCODING-SPACE`" (see 23.9 *bis.3.3*), and the decoder will return to the application all the bits contained in the defined encoding space except for value pre- and post-padding.

23.9 *bis*.2.2 In the case of an unknown decoding, the decoder will pass the bits forming the unknown encoding to the application as the value of the open type.

23.9 *bis*.3 Purpose and restrictions

23.9 *bis*.3.1 This syntax is used to define the way an open type is encoded, and the means that a decoder uses to determine the end of the encoding of an unknown type in an open type.

23.9 *bis*.3.2 If "`REPLACE STRUCTURE`" is set no other parameters shall be set.

23.9 *bis*.3.3 If "`ENCODING-SPACE SIZE`" is "`self-delimiting`" then "`UNKNOWN IS`" shall be set.

23.9 *bis*.4 Encoder actions

23.9 *bis*.4.1 For any encoding property group that is set, the encoder shall perform the encoder actions specified in clause 22, in the following order and in accordance with the encoding object definition:

a) replacement;

b) pre-alignment and padding;

c) start pointer;

d) encoding space (see 23.9 *bis*.4.3);

e) open-type encoding (see 23.9 *bis*.4.2);

f) value padding and justification (see 23.9 *bis*.4.5);

g) bit reversal.

23.9 *bis*.4.2 The encoder shall encode the value of the type supplied by the application using the "`ENCODED WITH`" encoding object set if this is present, otherwise the current combined encoding object set shall be used.

23.9 *bis*.4.3 If "`ENCODING-SPACE SIZE`" is "`variable-with-determinant`" or "`encoder-option-with-determinant`", it shall be the minimum number of "`MULTIPLE OF`" units needed to contain the pattern ("s", say), subject to 23.9 *bis*.4.5.

23.9 *bis*.4.4 An encoder (as an encoder's option) may increase "s" (as determined in 23.9 *bis*.4.3) in "`MULTIPLE OF`" units (subject to any restrictions that the range of values of any "`added-field`" or "`asn1-field`" imposes) if "`ENCODING-SPACE SIZE`" is set to "`encoder-option-with-determinant`".

23.9 *bis*.4.5 If the number of unused bits is not zero, then "`VALUE-JUSTIFICATION`" shall be applied using either the set values or the default values.

23.9 *bis*.5 Decoder actions

23.9 *bis*.5.1 For any encoding property group that is set, the decoder shall perform the decoder actions specified in clause 22, in the following order and in accordance with the encoding object definition:

a) pre-alignment and padding;

b) start pointer;

c) encoding space;

d) bit-reversal;

e) value padding and justification;

f) open-type decoding (see 23.9 *bis*.5.2).

23.9 *bis*.5.2 For open type decoding, the decoder shall query the application for the type which has been encoded and shall decode a value of that type or of the "**UNKNOWN IS**" structure in accordance with the "**ENCODED WITH**" specifications in the "**UNKNOWN IS**".

23.9 *bis*.5.3 If the decoding was of an unknown type, the bits forming the unknown encoding (without pre-padding bits and without value pre- and post-padding bits, if any) shall be passed to the application as the value of the open type.

24 Subclause 23.12.2.3

Replace 23.12.2.3 with:

23.12.2.3 If an encoding object in the "**REPETITION-ENCODINGS**" ordered list is defined using "**IF**" or "**IF-ALL**", then all preceding encoding objects in that list shall be defined using "**IF**" or "**IF-ALL**".

25 Subclause 23.13.1

Replace 23.13.1 with:

23.13.1 The defined syntax

The syntax for defining encoding objects for the **#CONDITIONAL-REPETITION** class is defined as:

```
#CONDITIONAL-REPETITION ::= ENCODING-CLASS {

              -- Condition (see 21.12)
              &size-range-condition              SizeRangeCondition  OPTIONAL,
              &comparison                        Comparison OPTIONAL,
              &comparator                        INTEGER OPTIONAL,
              &Size-range-conditions             SizeRangeCondition ORDERED OPTIONAL,
              &Comparisons                       Comparison ORDERED OPTIONAL,
              &Comparators                       INTEGER ORDERED OPTIONAL,

              -- Structure or component replacement specification (see 22.1)
              &#Replacement-structure            OPTIONAL,
              &replacement-structure-encoding-object  &#Replacement-structure OPTIONAL,
              &#Head-end-structure               OPTIONAL,

              -- Pre-alignment and padding specification (see 22.2)
              &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
              &encoding-space-pre-padding        Padding DEFAULT zero,
              &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                                 DEFAULT bits:'0'B,

              -- Start pointer specification (see 22.3)
              &start-pointer                     REFERENCE    OPTIONAL,
              &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
              &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

              -- Repetition space specification (see 22.7)
              &repetition-space-size             EncodingSpaceSize
                                                 DEFAULT self-delimiting-values,
              &repetition-space-unit             Unit DEFAULT bit,
              &repetition-space-determination    RepetitionSpaceDetermination
                                                 DEFAULT field-to-be-set,
              &main-reference                    REFERENCE OPTIONAL,
              &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
              &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,
              &handle-id                         PrintableString
                                                 DEFAULT "default-handle",
              &termination-pattern               Non-Null-Pattern (ALL EXCEPT
                                                 different:any) DEFAULT '0'B,

              -- Repetition alignment
              &repetition-alignment              ENUMERATED {none, aligned}
                                                 DEFAULT none,
-- Value padding and justification (see 22.8)
&value-justification              Justification DEFAULT right:0,
&value-pre-padding                Padding DEFAULT zero,
&value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
&value-post-padding               Padding DEFAULT zero,
&value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
&unused-bits-determination        UnusedBitsDetermination
                                  DEFAULT field-to-be-set,
&unused-bits-reference            REFERENCE OPTIONAL,
&Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
&Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL,

-- Identification handle specification (see 22.9)
&exhibited-handle                 PrintableString OPTIONAL,
&Handle-positions                 INTEGER (0..MAX) OPTIONAL,
&Handle-value                     HandleValue DEFAULT tag: any,

-- Bit reversal specification (see 22.12)
&bit-reversal                     ReversalSpecification
                                  DEFAULT no-reversal
}
WITH SYNTAX {
[IF &size-range-condition [&comparison  &comparator]]
[IF-ALL &Size-range-conditions [&Comparisons &Comparators]]
[ELSE]
[REPLACE
      [STRUCTURE]
      [COMPONENT]
      [ALL COMPONENTS]
      WITH &Replacement-structure
      [ENCODED BY &replacement-structure-encoding-object
            [INSERT AT HEAD &#Head-end-structure]]]
[ALIGNED TO
      [NEXT]
      [ANY]
            &encoding-space-pre-alignment-unit
            [PADDING &encoding-space-pre-padding
            [PATTERN &encoding-space-pre-pattern]]]
[START-POINTER   &start-pointer
      [MULTIPLE OF     &start-pointer-unit]
      [ENCODER-TRANSFORMS    &Start-pointer-encoder-transforms]]
REPETITION-SPACE
      [SIZE &repetition-space-size
            [MULTIPLE OF &repetition-space-unit]]
      [DETERMINED BY &repetition-space-determination
            [HANDLE &handle-id]]
      [USING &main-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
      [PATTERN &termination-pattern]
[ALIGNMENT &repetition-alignment]
[VALUE-PADDING
      [JUSTIFIED &value-justification]
      [PRE-PADDING &value-pre-padding
            [PATTERN &value-pre-pattern]]
      [POST-PADDING &value-post-padding
            [PATTERN &value-post-pattern]]
      [UNUSED BITS
            [DETERMINED BY &unused-bits-determination]
            [USING &unused-bits-reference
                  [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                  [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
[AS &handle-value]]
[BIT-REVERSAL &bit-reversal]
}
```