INTERNATIONAL STANDARD

ISO/IEC 8825-8

Second edition
2021-06

Information technology — ASN.1 encoding rules —

Part 8: Specification of JavaScript Object Notation Encoding Rules (JER)

Reference number
ISO/IEC 8825-8:2021(E)

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

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.697 (02/2021).

This second edition cancels and replaces the first edition (ISO/IEC 8825-8:2018), 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.

CONTENTS

Page

1 Scope 1

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

3 Definitions 2
3.1 Specification of Basic Notation 2
3.2 Information Object Specification 2
3.3 Constraint Specification 2
3.4 Parameterization of ASN.1 Specification 2
3.5 Basic Encoding Rules (BER) 2
3.6 Packed Encoding Rules (PER) 2
3.7 Additional definitions 2

4 Abbreviations 3

5 Encodings specified by this Recommendation | International Standard 3

6 Conformance 4

7 General provisions 4
7.1 Use of the type notation 4
7.2 Constraints 5
7.3 Type and value model used for encoding 6
7.4 Types to be encoded 6
7.5 Encoding instructions 6
7.6 Production of a complete JER encoding 7

8 Notation, lexical items and keywords used in JER encoding instructions 7

9 Specifying JER encoding instructions 8

10 Assigning a JER encoding instruction using a type prefix 9

11 Assigning a JER encoding instruction using a JER encoding control section 9

12 Identification of the targets for a JER encoding instruction 9
12.1 General rules 9
12.2 Types defined in the module 10
12.3 Built-in types 10
12.4 Types imported from another module 10

13 Multiple assignment of JER encoding instructions 10
13.1 Order in which multiple assignments are considered 10
13.2 Effect of assigning a negating encoding instruction 11
13.3 Multiple assignment of JER encoding instructions of the same category 11

14 The ARRAY encoding instruction 11
14.1 General 11

14.2 Restrictions 11

15 The BASE64 encoding instruction 11
15.1 General 11
15.2 Restrictions 12

16 The NAME encoding instruction 12
16.1 General 12
16.2 Restrictions 13

17 The OBJECT encoding instruction 13
17.1 General 13
17.2 Restrictions 13

18 The TEXT encoding instruction 13
18.1 General 13
18.2 Restrictions 14

19 The UNWRAPPED encoding instruction 14
19.1 General 14
19.2 Restrictions 14

20 Encoding of boolean values 14

21 Encoding of integer values 15

22 Encoding of enumerated values 15

23 Encoding of real values 15
23.1 General 15
23.2 Encoding of the special real values 15
23.3 Encoding as a JSON number 16
23.4 Encoding as a JSON object 16

24 Encoding of bitstring values 16
24.1 General 16
24.2 Encoding of bitstring types with a fixed size 16
24.3 Encoding of bitstring types with a variable size 16
24.4 Alternative encoding of bitstring types with a JER-visible contents constraint 16

25 Encoding of octetstring values 17
25.1 General 17
25.2 Encoding of an octetstring value as a JSON string containing a Base64 encoding 17
25.3 Encoding of an octetstring value as a JSON string containing a hexadecimal encoding 17
25.4 Alternative encoding of an octetstring type with a JER-visible contents constraint 17

26 Encoding of the null value 17

27 Encoding of sequence values 17
27.1 General 17
27.2 Array-based encoding 17
27.3 Object-based encoding 17

28 Encoding of sequence-of values 18
29 Encoding of set values 18
30 Encoding of set-of values 18
30.1 General 18
30.2 Array-based encoding 18
30.3 Object-based encoding 18
31 Encoding of choice values 19
31.1 General 19
31.2 Unwrapped encoding 19
31.3 Wrapped encoding 19
32 Encoding of object identifier values 19
33 Encoding of relative object identifier values 19
34 Encoding of values of the internationalized resource reference type 19
35 Encoding of values of the relative internationalized resource reference type 19
36 Encoding of values of the embedded-pdv type 19
37 Encoding of values of the external type 20
38 Encoding of values of the restricted character string types 20
39 Encoding of values of the unrestricted character string type 20
40 Encoding of values of the time types 20
41 Encoding of open type values 20
42 Object identifier values referencing the encoding rules 20
Annex A Examples of JER encodings 21
A.1 ASN.1 description of the record structure 21
A.2 ASN.1 description of a record value 21
A.3 Example JER representation of this record value 21
A.4 Additional examples of JER encodings 22
Annex B Examples of JER encoding instructions and their effect on the encodings 25
B.1 ASN.1 description of the record structure 25
B.2 ASN.1 description of a record value 25
B.3 JER representation of this record value 26
B.4 Additional examples of JER encodings 26
B.5 Examples of JER encodings of choice types 28

Introduction

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 and Rec. ITU-T X.683 | ISO/IEC 8824-4 together describe Abstract Syntax Notation One (ASN.1), a notation for the definition of messages to be exchanged between peer applications.

This Recommendation | International Standard defines encoding rules that may be applied to values of ASN.1 types defined using the notation specified in the publications listed in the previous paragraph. Application of these encoding rules produces a transfer syntax for such values. It is implicit in the specification of these encoding rules that they are also to be used for decoding.

There is more than one set of encoding rules that can be applied to values of ASN.1 types. This Recommendation | International Standard defines a set of JavaScript Object Notation Encoding Rules (JER), so called because the encodings they produce are instances of the JSON grammar specified in ECMA-404.

This Recommendation | International Standard specifies the syntax and semantics of JER encoding instructions that modify the JSON text produced by the application of JER to certain ASN.1 types.

Clauses 8 to 12 list the JER encoding instructions and specify the syntax for their assignment to an ASN.1 type or component using either a JER type prefix (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 31.3) or a JER encoding control section (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 54).

Clause 13 defines the order of precedence if JER encoding instructions are present in both a JER type prefix and in a JER encoding control section.

Clauses 14 to 19 specify:

a) the syntax of each JER encoding instruction used in a type prefix or a JER encoding control section;

b) restrictions on the JER encoding instructions that can be associated with a particular ASN.1 type (resulting from inheritance and multiple assignments).

Clauses 20 to 41 specify the JER encoding of ASN.1 types, referencing earlier clauses that define the JER encoding instructions.

Annex A is informative and contains examples of JER encodings where JER encoding instructions are not used.

Annex B is informative and contains examples of JER encoding instructions and their effect on the JER encodings.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – ASN.1 encoding rules: Specification of JavaScript Object Notation Encoding Rules (JER)

1 Scope

This Recommendation | International Standard specifies a set of JavaScript Object Notation Encoding Rules (JER) that may be used to derive a transfer syntax for values of types defined in 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 and Rec. ITU-T X.683 | ISO/IEC 8824-4. It is implicit in the specification of these encoding rules that they are also to be used for decoding.

The encoding rules specified in this Recommendation | International Standard:

– are used at the time of communication;
– are intended for use in circumstances where interoperability with applications using JSON is the major concern in the choice of encoding rules;
– allow the extension of an abstract syntax by addition of extra values for all forms of extensibility described in Rec. ITU-T X.680 | ISO/IEC 8824-1.

This Recommendation | International Standard also specifies the syntax and semantics of JER encoding instructions, as well as the rules for their assignment and combination. JER encoding instructions can be used to control JER encoding for specific Abstract Syntax Notation One (ASN.1) types.

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 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 and the Unicode standard version 3.2.0:2002. It cannot be applied using later versions of these two standards.

2.1 Identical Recommendations | International Standards

– Recommendation. ITU-T X.226 (1994) | ISO/IEC 8823-1:1994, *Information technology – Open Systems Interconnection – Connection-oriented Presentation protocol: Protocol specification*.
– 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).*

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

2.2 Additional references

– ECMA Standard ECMA-404 (2017), *The JSON Data Interchange Syntax*.

– IETF RFC 2045 (1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.*

–

– ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet coded character set (UCS).*

3 Definitions

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

3.1 Specification of basic notation

For the purposes of this Recommendation | International Standard, all the definitions in Rec. ITU-T X.680 | ISO/IEC 8824-1 apply.

3.2 Information object specification

For the purposes of this Recommendation | International Standard, all the definitions in Rec. ITU-T X.681 | ISO/IEC 8824-2 apply.

3.3 Constraint specification

This Recommendation | International Standard makes use of the following terms defined in Rec. ITU-T X.682 | ISO/IEC 8824-3:

a) component relation constraint;

b) table constraint.

3.4 Parameterization of ASN.1 specification

This Recommendation | International Standard makes use of the following term defined in Rec. ITU-T X.683 | ISO/IEC8824-4:

– variable constraint.

3.5 Basic Encoding Rules (BER)

This Recommendation | International Standard makes use of the following terms defined in Rec. ITU-T X.690 | ISO/IEC 8825-1:

a) data value;

b) dynamic conformance;

c) encoding (of a data value);

d) receiver;

e) sender;

f) static conformance.

3.6 Packed Encoding Rules (PER)

This Recommendation | International Standard makes use of the following terms defined in Rec. ITU-T X.691 | ISO/IEC 8825-2:

a) composite type;

b) composite value;

c) outermost type;

d) relay-safe encoding;

e) simple type;

f) textually dependent.

3.7 Additional definitions

3.7.1 abstract syntax value: A value of an abstract syntax (defined as a set of values of a single ASN.1 type) which is to be encoded by JER or which is generated by JER decoding.

3.7.2 **associated encoding instruction (for a type)**: A set of JER encoding instructions associated with a type.

3.7.3 **effective value constraint** (of an integer type): The smallest integer range that includes all the values of the integer type that are permitted by the JER-visible constraints (see clause 7.2.7).

3.7.4 **effective size constraint** (of a bitstring type): The smallest integer range that includes the lengths of all the values of the string type that are permitted by the JER-visible constraints (see clause 7.2.8).

3.7.5 **final encoding instructions (for a type)**: The set of JER encoding instructions associated with a type as a result of the complete ASN.1 specification, and which are applied in producing encodings of that type.

3.7.6 **inherited encoding instructions**: A set of JER encoding instructions that are associated with the type identified by a type reference.

3.7.7 **JSON array**: A series of JSON tokens that constitute an array structure as specified in ECMA-404, clause 7.

3.7.8 **JSON member name string** (of a component of a sequence, set or choice type that is encoded as a JSON object): The Unicode character string denoted by the name of the member of the JSON object identifying the component in the JER encoding.

3.7.9 **JSON number**: A JSON token that is a number as specified in ECMA-404, clause 8.

3.7.10 **JSON object**: A series of JSON tokens that constitute an object structure as specified in ECMA-404, clause 6.

3.7.11 **JSON string**: A JSON token that is a string as specified in ECMA-404, clause 9.

NOTE – A JSON string is part of a JER encoding, it begins and ends with a quotation mark, may contain escapes, and is distinct from the Unicode character string that it denotes.

3.7.12 **JSON token**: A Unicode character string that is one of the several kinds of tokens specified in ECMA-404, clause 4.

3.7.13 **JER encoding instruction**: Notation used to change the JER encoding of a type.

3.7.14 **JER-visible constraint**: An instance of use of the ASN.1 constraint notation that affects the JER encoding of a value.

3.7.15 **octet**: A group of eight consecutive bits, numbered from bit 8 (the most significant bit) to bit 1 (the least significant bit).

3.7.16 **prefixed encoding instruction**: A JER encoding instruction that is assigned to a type using a type prefix.

NOTE – Prefixed encoding instructions can delete, replace or add to the associated encoding instructions of a type.

3.7.17 **targeted encoding instruction**: A JER encoding instruction that is assigned to multiple types using a target list in a JER encoding control section.

NOTE – Targeted encoding instructions can delete, replace or add to the associated encoding instructions of multiple types.

4 Abbreviations

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

ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
JER	JavaScript Object Notation Encoding Rules
OSI	Open Systems Interconnection
PER	Packed Encoding Rules
UTF-8	Unicode Transformation Format 8 bit

5 Encodings specified by this Recommendation | International Standard

5.1 This Recommendation | International Standard specifies a set of encoding rules that can be used to encode and decode the values of an abstract syntax defined as the values of a single (known) ASN.1 type. This clause describes their applicability and properties.

5.2 JER encodings are always relay-safe provided the abstract values of the types **`EXTERNAL`**, **`EMBEDDED PDV`**, and **`CHARACTER STRING`** are constrained to prevent the carriage of open systems interconnection (OSI) presentation context identifiers.

5.3 If a type encoded with JER contains **EXTERNAL**, **EMBEDDED PDV**, or **CHARACTER STRING** types, then the outer encoding ceases to be relay-safe unless the transfer syntax used for all the **EXTERNAL**, **EMBEDDED PDV**, or **CHARACTER STRING** types is relay-safe.

NOTE – The character transfer syntaxes supporting all character abstract syntaxes of the form **{iso standard 10646 level-1(1)** …**}** are canonical. Those supporting **{iso standard 10646 level-2(2)** …**}** and **{iso standard 10646 level-3(3)** …**}** are not always canonical. All these character transfer syntaxes are relay-safe.

5.4 JER encodings are self-delimiting. Encodings are always a whole multiple of 8 bits. When carried in an **EXTERNAL** type, they shall be carried in the **OCTET STRING** choice alternative, unless the **EXTERNAL** type itself is encoded in JER, in which case the value may be encoded as a single ASN.1 type (i.e., an open type). When carried in an OSI presentation protocol, the "full encoding" (as defined in Rec. ITU-T X.226 | ISO/IEC 8823-1) with the **OCTET STRING** alternative shall be used.

5.5 This Recommendation | International Standard also specifies the syntax and semantics of JER encoding instructions (see clauses 14 to 19).

5.6 ASN.1 forms a basic JSON schema notation. The ASN.1 schema is used to define the content and structure of data using ASN.1 and the JavaScript Object Notation Encoding Rules. It can be used without JER encoding instructions.

5.7 JER encoding instructions provide wider flexibility in the JSON texts that can be specified.

5.8 JER encoding instructions are assigned to ASN.1 type definitions or to type references using either or both JER type prefixes (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 31.3) and a JER encoding control section (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 54). If encoding instructions are associated with a type definition, they are carried with the ASN.1 type (through its type reference) into other type definitions and other ASN.1 modules. The final encoding instructions of a type are applied when the type is encoded in JER and modify the JSON text produced.

6 Conformance

6.1 Dynamic conformance for the JavaScript Object Notation Encoding Rules is specified in clauses 7 to 41.

6.2 Static conformance is specified by those standards that specify the application of these encoding rules.

6.3 Alternative encodings are permitted by the JavaScript Object Notation Encoding Rules as encoder's options. Decoders that claim conformance to JER shall support all JER encoding alternatives.

6.4 The rules in this Recommendation | International Standard are specified in terms of an encoding procedure. Implementations are not required to mirror the procedure specified, provided the octet string produced as the complete encoding of an abstract syntax value is identical to one of those specified in this Recommendation | International Standard for the applicable transfer syntax.

6.5 Implementations performing decoding are required to produce the abstract syntax value corresponding to any received octet string that could be produced by a sender conforming to the encoding rules identified in the transfer syntax associated with the material being decoded.

6.6 If an ASN.1 specification assigns JER encoding instructions in accordance with clauses 8 to 13 such that an ASN.1 type or component has final encoding instructions that violate the restrictions specified in clauses 14 to 19, then that ASN.1 specification is not in conformity with this Recommendation | International Standard, even if (without the encoding instructions) it would conform to all the requirements of Rec. ITU-T X.680 | ISO/IEC 8824-1.

NOTE – It is only occasionally invalid to assign an encoding instruction to a "type", as it can be negated (removed from the set of associated encoding instructions) by a further assignment. It is the final encoding instructions that determine conformity of the specification.

7 General provisions

7.1 Use of the type notation

7.1.1 These encoding rules make specific use of the ASN.1 type notation as specified in 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 and Rec. ITU-T X.683 | ISO/IEC 8824-4, and can only be applied to encode the values of a single ASN.1 type specified using that notation.

7.1.2 In particular, but not exclusively, they are dependent on the following information being retained in the ASN.1 type and value model underlying the use of the notation:

a) the identifiers of the components of a sequence or set type and of the alternatives of a choice type;

b) the identifiers of the enumeration items of an enumerated type;

c) whether a set or sequence type component has a default value or not;

d) the restricted range of values of a type that arises through the application of JER-visible constraints;

e) whether the type of a component is open.

7.2 Constraints

NOTE – The fact that some ASN.1 constraints may not be JER-visible for the purposes of encoding and decoding does not in any way affect the use of such constraints in the handling of errors detected during decoding, nor does it imply that values violating such constraints are allowed to be transmitted by a conforming sender. However, this Recommendation | International Standard makes no use of such constraints in the specification of encodings.

7.2.1 In general, the constraint on a type will consist of individual constraints combined using some or all of set arithmetic, contained subtype constraints and serial application of constraints.

The following constraints are JER-visible:

a) non-extensible size constraints on bitstring types;

b) non-extensible single value constraints on real types where the single value is either plus zero or minus zero or one of the special real values PLUS-INFINITY, MINUS-INFINITY and NOT-A-NUMBER;

c) non-extensible single value constraints and value range constraints on the base of a real type;

d) an inner type constraint that applies a non-extensible single value constraint or value range constraint to the base of a real type;

e) a contents constraint with CONTAINING but without ENCODED BY;

f) a contained subtype constraint in which the constraining type carries a JER-visible constraint.

7.2.2 All other constraints are not JER-visible. In particular, the following constraints are not JER-visible:

a) constraints that are expressed in human-readable text or in ASN.1 comment;

b) variable constraints (see Rec. ITU-T X.683 | ISO/IEC 8824-4, clauses 10.3 and 10.4);

c) user-defined constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3, 9.1);

d) table constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3);

e) component relation constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3, 10.7);

f) constraints whose evaluation is textually dependent on a table constraint or a component relation constraint (see Rec. ITU-T X.682 | ISO/IEC 8824-3);

g) extensible subtype constraints;

h) size constraints applied to a character string or octet string type;

i) single value subtype constraints applied to a character string type;

j) permitted alphabet constraints;

k) pattern constraints;

l) value and value range constraints on integer types;

m) constraints on real types except those specified in clause 7.2.1 b) and c);

n) constraints on the time type and on the useful and defined time types;

o) inner type constraints except those specified in clause 7.2.1 d);

p) constraints on the useful types.

7.2.3 If a type is specified using a serial application of constraints, each of those constraints may or may not be individually JER-visible. If the last subtype constraint of the series of constraints is JER-visible and contains an extension marker, then that subtype constraint is extensible for the purposes of these encoding rules. Any other constraint is not extensible for the purposes of these encoding rules, even if it contains an extension marker.

NOTE – In a serial application of constraints, each subtype constraint removes the extensibility specified in earlier constraints of the series of constraints (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 50.8).

7.2.4 If a constraint that is JER-visible is part of an **`INTERSECTION`** construction, then the resulting constraint is JER-visible, and consists of the **`INTERSECTION`** of all the JER-visible parts (with the non-JER-visible parts ignored).

7.2.5 If a constraint that is not JER-visible is part of a **`UNION`** construction, then the resulting constraint is not JER-visible.

7.2.6 If a constraint has an **`EXCEPT`** clause, the **`EXCEPT`** keyword and the following value set is completely ignored, whether the value set following the **`EXCEPT`** keyword is JER-visible or not.

7.2.7 The effective value constraint of an integer type is an integer range determined as follows, taking into account all the JER-visible constraints present in the type definition and ignoring any constraints that are not JER-visible:

a) the lower bound of the effective value constraint is the least permitted value of the integer type, if such a value exists; otherwise, the effective value constraint has no finite lower bound;

b) the upper bound of the effective value constraint is the greatest permitted value of the integer type, if such a value exists; otherwise, the effective value constraint has no finite upper bound.

NOTE – The only integer types that can have an effective value constraints with a finite lower or upper bound are the type of the components of a real type, to which a value or value range constraint is applied by using an inner type constraint. Value constraints on all other integer types are not JER-visible, and therefore the effective value constraint of those types has no finite lower or upper bound.

7.2.8 The effective size constraint of a bitstring type is a single integer range determined as follows, taking into account all the JER-visible constraints present in the type definition and ignoring any constraints that are not JER-visible:

a) the lower bound of the effective size constraint is the length of the shortest permitted value of the string type (possibly zero);

b) the upper bound of the effective size constraint is the length of the longest permitted value of the string type, if such length is finite; otherwise, the effective size constraint has no finite upper bound.

7.3 Type and value model used for encoding

7.3.1 An ASN.1 type is either a simple type or a type built using other types. The notation permits the use of type references and tagging of types. For the purpose of these encoding rules, the use of type references and tagging have no effect on the encoding and are invisible in the model. The notation also permits the application of constraints and of error specifications. JER-visible constraints are present in the model as a restriction of the values of a type. Other constraints and error specifications do not affect encoding and are invisible in the JER type and value model.

7.3.2 A value to be encoded can be considered as either a simple value or as a composite value built using the structuring mechanisms from components that are either simple or composite values, paralleling the structure of the ASN.1 type definition.

7.4 Types to be encoded

7.4.1 Clauses 20 to 41 specify the encoding of the following types: boolean, integer, enumerated, real, bitstring, octetstring, null, sequence, sequence-of, set, set-of, choice, object identifier, relative object identifier, internationalized resource reference, relative internationalized resource reference, embedded-pdv, external, restricted character string, unrestricted character string, time, and open types.

7.4.2 The selection type shall be encoded as an encoding of the selected type.

7.4.3 This Recommendation | International Standard does not contain specific provisions for the encoding of tagged types as tagging is not visible in the type and value model used for these encoding rules. Tagged types are thus encoded according to the type that has been tagged.

7.4.4 An encoding prefixed type is encoded according to the type that has been prefixed.

7.4.5 The useful types **`GeneralizedTime`**, **`UTCTime`**, and **`ObjectDescriptor`** shall be encoded as if they had been replaced by their definitions given in Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 45. Constraints on the useful types are not JER-visible.

7.4.6 A type defined using a value set assignment shall be encoded as if the type had been defined using the production specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 16.8.

7.5 Encoding instructions

7.5.1 JER encoding instructions modify the JSON text produced by the application of the JER to a type.

7.5.2 All occurrences of ASN.1 "Type" notation have an associated set (possibly empty) of JER encoding instructions (the final encoding instructions). Encoding instructions are associated with a "Type" through:

a) (inherited encoding instructions) the presence of associated encoding instructions on the "Type" used in the definition of a "typereference" used as a "Type";

b) (targeted encoding instructions) assignment of one or more JER encoding instructions to multiple types using a JER encoding control section;

c) (prefixed encoding instructions) assignment of one or more JER encoding instructions to an occurrence of "Type" using JER type prefixes; and

d) (import-list encoding instructions) assignment of one or more encoding instructions to all type references imported from an identified ASN.1 module.

7.5.3 The effect of assigning a JER encoding instruction is to add, delete or replace associated encoding instructions.

7.5.4 The order (or manner) in which encoding instructions become part of (or are removed from) the set of associated encoding instructions is not significant in the application of the final encoding instructions.

7.5.5 The final encoding instructions affect the JER encoding of types. They have no other impact, and in particular are not associated with any value reference defined using the type, nor do they affect value mappings, nor do they affect other encoding rules.

7.6 Production of a complete JER encoding

7.6.1 If an ASN.1 type is encoded using JER and the encoding is contained in:

a) an ASN.1 bitstring type or octetstring type; or

b) any part of an ASN.1 external or embedded-pdv type; or

c) any carrier protocol that is not defined using ASN.1,

then that ASN.1 type is defined as an outermost type, and clause 7.6.2 shall be applied to all the encodings of its abstract values.

7.6.2 The series of JSON tokens that constitute the encoding of an abstract value of the outermost type shall be encoded in UTF-8 into an octet string, which is the complete encoding of the abstract value of the outermost type.

7.6.3 The use of any of the escapes specified in ECMA-404, clause 9, is permitted in any JSON string produced by these encoding rules.

8 Notation, lexical items and keywords used in JER encoding instructions

8.1 The notation used in specifying the syntax of an "EncodingInstruction" in a JER type prefix (see clause 10), and in an "EncodingInstructionAssignmentList" in a JER encoding control section (see clause 11) is that defined by Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 5.

8.2 Rec. ITU-T X.680 | ISO/IEC 8824-1, clauses 11 and 12.1 also apply to a JER "EncodingInstruction" and to a JER "EncodingInstructionAssignmentList".

NOTE – In particular, arbitrary ASN.1 white-space characters and ASN.1 comments can appear between lexical items in both of these syntactic constructs.

8.3 The following lexical items are used in this Recommendation | International Standard:

comment	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.6)
identifier	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.3)
modulereference	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.5)
number	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.8)
typereference	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.2)
"{"	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.37)
"}"	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.37)
":"	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.37)
"["	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.37)
"]"	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.37)
","	(see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 12.37)

8.4 The keywords specified in clause 8.5 are used in either or both JER "EncodingInstruction"s and JER "EncodingInstructionAssignmentList"s (in addition to some ASN.1 reserved words), and can appear in such syntactic constructs only with the meaning assigned to them in the following clauses of this Recommendation | International Standard.

8.5 The keywords are:

```
ARRAY
AS
BASE64
CAPITALIZED
IN
LOWERCAMELCASED
LOWERCASED
NAME
NOT
OBJECT
TEXT
UNCAPITALIZED
UNWRAPPED
UPPERCAMELCASED
UPPERCASED
```

9 Specifying JER encoding instructions

9.1 The JER "EncodingInstruction" production is:

EncodingInstruction::=
PositiveInstruction
| NegatingInstruction

PositiveInstruction ::=
Instruction

NegatingInstruction ::=
NOT **Instruction**

Instruction ::=
ArrayInstruction
| Base64Instruction
| NameInstruction
| ObjectInstruction
| TextInstruction
| UnwrappedInstruction

9.2 JER encoding instructions can be assigned to ASN.1 types using either the "EncodingInstruction" production in a JER type prefix or the "EncodingInstructionAssignmentList" production in a JER encoding control section. Assignment using a type prefix is specified in clause 10. Assignment using a JER encoding control section is specified in clause 11.

9.3 An encoding instruction in a type prefix or in a JER encoding control section can be a positive instruction, used to add or to replace an encoding instruction (use of "PositiveInstruction"), or a negating instruction used to cancel one or more associated encoding instructions (use of "NegatingInstruction").

9.4 An "Instruction" consists of three parts:

a) a keyword identifying the category of the encoding instruction;

b) syntax, specific to each encoding instruction category, providing details of the encoding instruction.

NOTE – When used in a negating instruction, this is always empty. It is also empty in some JER encoding instructions for which the keyword is a sufficient definition.

9.5 Some JER encoding instructions require the specification of one or more abstract values of a type. This specification uses the "Value" production (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 17.7). If a "valuereference" is used as "Value", then this "valuereference" shall be defined in (or imported into) the ASN.1 module containing the JER encoding instruction.

NOTE – This means that the value can be specified either directly using basic ASN.1 value notation or by a value reference.

9.6 Table 1 lists in column 1 the alternatives in the "Instruction" production. Column 2 gives the clauses that specify the requirements for use of these encoding instructions and their effects on the encodings.

Table 1 – Encoding instructions and their defining clauses

Encoding instruction	Defining clause
ArrayInstruction	Clause 14
Base64Instruction	Clause 15
NameInstruction	Clause 16
ObjectInstruction	Clause 17
TextInstruction	Clause 18
UnwrappedInstruction	Clause 19

9.7 Each of the alternatives of the "Instruction" production is in a defined category of encoding instruction. In this Recommendation | International Standard, the category of each encoding instruction is denoted by the name of the corresponding production.

9.8 An ASN.1 type can never have more than one associated JER encoding instruction of a given category, no matter how they are assigned. The result of multiple assignments of a JER encoding instruction of a given category is specified in clause 13.3.

9.9 If the "Type" in a "TypeAssignment" (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 16.1) has final encoding instructions, all uses of the corresponding "typereference" (in the module containing the "TypeAssignment" or in some other module) inherit its final associated encoding instructions, except that any final **NAME** encoding instruction is not inherited.

10 Assigning a JER encoding instruction using a type prefix

Each use of an "EncodingInstruction" in a type prefix assigns that JER encoding instruction to the occurrence of "Type" associated with the type prefix.

11 Assigning a JER encoding instruction using a JER encoding control section

11.1 The JER "EncodingInstructionAssignmentList" production is:

```
EncodingInstructionAssignmentList ::=
    TargetedEncodingInstruction
        EncodinginstructionAssignmentList ?

TargetedEncodingInstruction ::=
        "[" EncodingInstruction "]" TargetList
```

11.2 The "EncodingInstruction" production is defined in clause 9.

11.3 Each use of a "TargetedEncodingInstruction" in a JER encoding control section assigns the JER encoding instruction to the occurrences of "Type" that are identified in the "TargetList" of the targeted encoding instruction. The "TargetList" production and the targets it identifies are specified in clause 12.

12 Identification of the targets for a JER encoding instruction

12.1 General rules

12.1.1 The target of a prefixed encoding instruction is always the type associated with the prefix.

12.1.2 The target of a targeted encoding instruction is determined as follows.

12.1.3 The "TargetedEncodingInstruction" specifies the JER encoding instruction that is being assigned, and the target(s) for that assignment within the ASN.1 module, specified by the production "TargetList". All targets are an occurrence of the "Type" production within the ASN.1 module.

12.1.4 The "TargetList" production is:

TargetList ::=
Targets "," +

Targets ::=
TypeIdentification
| BuiltInTypeIdentification
| ImportedTypesIdentification

12.1.5 The JER encoding instruction is assigned to all the types identified by the "TargetList" as specified in clauses 12.2 to 12.4.

12.2 Types defined in the module

12.2.1 The "TypeIdentification" production is:

TypeIdentification ::=
ALL

12.2.2 A use of this production identifies all "Type"s in "TypeAssignment"s in the module.

12.3 Built-in types

12.3.1 The "BuiltInTypeIdentification" production is:

BuiltInTypeIdentification ::=
CHOICE
| ENUMERATED
| OCTET STRING
| SEQUENCE
| SET OF

12.3.2 A use of this production identifies all textual occurrences within the module of the corresponding built-in type or of a type defined using the corresponding constructor.

12.4 Types imported from another module

12.4.1 The "ImportedTypesIdentification" production is:

ImportedTypesIdentification ::=
ALL IMPORTS FROM **modulereference**

12.4.2 The "modulereference" shall be one of the "modulereference"s used in one of the "GlobalModuleReferences" of the imports clause of the module.

12.4.3 The JER encoding instruction is assigned to each of the "typereference"s in the corresponding "SymbolList", after the final encoding instructions produced by assignment in the exporting module have been assigned.

12.4.4 If an imported "typereference" is exported from this module, the final encoding instructions inherited by that "typereference" in a module that imports it are those inherited in this importing module, and are not affected by assignment of encoding instructions using an "ImportedTypesIdentification". This assignment affects only the use of the type reference within this module.

13 Multiple assignment of JER encoding instructions

13.1 Order in which multiple assignments are considered

13.1.1 A "Type" which is not a "typereference" initially has an empty set of associated encoding instructions.

13.1.2 A "Type" which is a "typereference" (which may be imported) initially has the set of final encoding instructions of the "Type" which was assigned to it when it was defined (possibly modified by encoding instructions assigned to it in the imports list of an importing module – see clause 12.4).

13.1.3 Targeted encoding instructions for a "Type" (using a JER encoding control section) are assigned next, in the order in which the targeted encoding instructions appear in the JER encoding control section.

13.1.4 Prefixed encoding instructions (using a type prefix) assigned to a type are considered next, with the rightmost (the innermost) prefixed encoding instruction considered first, and the leftmost (the outermost) prefixed encoding instruction considered last.

13.1.5 Each assignment of an encoding instruction produces a new set of associated encoding instructions, as specified in clauses 13.2 to 13.3.

13.2 Effect of assigning a negating encoding instruction

An assignment of a negating encoding instruction (use of "NegatingInstruction") results in the removal (from the set of associated encoding instructions) of any encoding instruction of the same category.

NOTE 1 – A negating encoding instruction never becomes part of the set of associated encoding instructions.

NOTE 2 – When an "Instruction" occurs as part of a "NegatingInstruction", the "Instruction" consists only of a keyword (for example, **NAME** rather than **NAME AS** "a").

13.3 Multiple assignment of JER encoding instructions of the same category

NOTE – Multiple assignment of JER encoding instructions of the same category is expected to be rare, except where a JER encoding instruction is assigned globally, and an overriding (possibly negating) encoding instruction is assigned to specific types or components.

13.3.1 Assignments of positive encoding instructions (use of "PositiveInstruction") result in the addition (to the set of associated encoding instructions) of that JER encoding instruction if there are no other associated encoding instructions of the same category.

13.3.2 If there is an encoding instruction of the same category in the set of associated encoding instructions, then that encoding instruction is removed from the set, and the assigned JER encoding instruction is added.

13.3.3 If a type that appears in a "ContentsConstraint", in a "TypeConstraint", or in a "TableConstraint" is to be encoded by JER, then the final encoding instructions (as determined by the rules in 13.3.1 and 13.3.2) are used to determine the encoding of that type. If a type appears in any other ASN.1 constraint, then all associated encoding instructions are discarded.

14 The ARRAY encoding instruction

14.1 General

14.1.1 The "ArrayInstruction" is:

ArrayInstruction ::=
ARRAY

14.1.2 Application of this final encoding instruction to a sequence type causes the type to be encoded as a JSON array instead of as a JSON object (see clause 27).

14.2 Restrictions

If the final encoding instructions for an ASN.1 type contain an **ARRAY** encoding instruction, then the type shall be a sequence type. Any component of the sequence type that is either:

a) a component marked **OPTIONAL** or **DEFAULT**, or

b) an extension addition that is a "ComponentType", or

c) a component contained in an extension addition group

shall not be an open type, an extensible choice type with a final **UNWRAPPED** encoding instruction, or a type that produces the JSON token **null** for one of its abstract values.

15 The BASE64 encoding instruction

15.1 General

15.1.1 The "Base64Instruction" is:

Base64Instruction ::=
BASE64

15.1.2 Application of this final encoding instruction to an octetstring type causes the type to be encoded as a Base64 string as specified in IETF RFC 2045, 6.8 (see clause 25).

15.2 Restrictions

If the final encoding instructions for an ASN.1 type contain a **BASE64** encoding instruction then the type shall be an octetstring type.

16 The NAME encoding instruction

16.1 General

16.1.1 The "NameInstruction" is:

NameInstruction ::=
NAME AS **NewNameOrKeyword**
| NAME

NewNameOrKeyword ::=
NewName
| **Keyword**

NewName ::=
RestrictedCharacterStringValue

Keyword ::=
CAPITALIZED
| UPPERCAMELCASED
| UPPERCASED
| LOWERCAMELCASED
| LOWERCASED

16.1.2 The first alternative of "NameInstruction" shall be used when the "NameInstruction" occurs in a "PositiveInstruction" and the second alternative shall be used when the "NameInstruction" occurs in a "NegatingInstruction".

16.1.3 This encoding instruction is normally assigned to the type of a component of a sequence, set, or choice type. Since this encoding instruction is not inherited (see clause 9.9), its presence among the final encoding instructions of a type has no effect on the JER encodings unless the type is the "Type" in the "NamedType" (possibly a "Type" that textually occurs within one or more nested "PrefixedType"s) of a sequence, set, or choice component.

16.1.4 Application of this final encoding instruction to the type of a component of a sequence, set, or choice type changes the Unicode string to be denoted by the name of the member identifying the component in the JER encoding. The new name can be specified either as a replacement string (use of "NewName") or as a case change operation (use of "Keyword") to be applied to the identifier of the component.

16.1.5 When the "Keyword" alternative is used, the new name shall be derived from the identifier by changing the case of some of its characters as specified in the following subclauses.

16.1.5.1 If the "Keyword" is **CAPITALIZED**, then the first character of the identifier (a lower-case letter) shall be replaced by its upper-case equivalent.

16.1.5.2 If the "Keyword" is **UPPERCASED**, then all characters of the identifier that are lower-case letters shall be replaced by their upper-case equivalent. Other characters are unchanged.

16.1.5.3 If the "Keyword" is **UPPERCAMELCASED**, then

a) the first character of the identifier (a lower-case letter) shall be replaced by its upper-case equivalent;

b) all characters of the identifier that are lower-case letters and are preceded by a hyphen shall be replaced by their upper-case equivalent; and

c) any hyphens present in the identifier shall be removed.

16.1.5.4 If the "Keyword" is **LOWERCASED**, then all characters of the identifier that are upper-case letters shall be replaced by their lower-case equivalent. Other characters are unchanged.

16.1.5.5 If the "Keyword" is **LOWERCAMELCASED**, then

a) all characters of the identifier that are lower-case letters and are preceded by a hyphen shall be replaced by their upper-case equivalent; and

b) any hyphens present in the identifier shall be removed.

16.2 Restrictions

When this final encoding instruction is associated with the types of one or more components of a sequence, set or choice type, the final set of Unicode character strings consisting of the unchanged strings and the new strings identifying the components of the type shall not contain two identical strings.

17 The OBJECT encoding instruction

17.1 General

17.1.1 The "ObjectInstruction" is:

ObjectInstruction ::=
    OBJECT

17.1.2 Application of this final encoding instruction to a set-of type whose component is a sequence type causes the set-of type to be encoded as a JSON object instead of as a JSON array (see clause 30). The JSON object will contain one member for each item of the set-of value.

NOTE – A typical use of this encoding instruction is to produce a JSON object that represents an unordered set of associations between a key (a value of an ASN.1 type encoded as a JSON string) and a value of an arbitrary ASN.1 type. Such a set is often called a "map".

17.2 Restrictions

If the final encoding instructions for an ASN.1 type contain an **OBJECT** encoding instruction, then the type shall be a set-of type whose component type is a sequence type with two components and without an extension marker. The first component of the sequence type shall be of one of the following types: **IA5String**, **ISO646String**, **VisibleString**, **NumericString**, **PrintableString**, **BMPString**, **UniversalString**, **UTF8String**, or an enumerated type. Neither component of the sequence type shall be marked **OPTIONAL** or **DEFAULT**.

18 The TEXT encoding instruction

18.1 General

18.1.1 The "TextInstruction" is:

TextInstruction ::=
    TEXT **TextChangeList**
    | TEXT

TextChangeList ::=
    TextChange "," +

TextChange ::=
    IdentifierOrAll AS **NewTextOrKeyword**

NewTextOrKeyword ::=
    NewText
    | **Keyword**

NewText ::=
    RestrictedCharacterStringValue

IdentifierOrAll ::=
    identifier
    | ALL

18.1.2 The first alternative of "TextInstruction" shall be used when the "TextInstruction" occurs in a "PositiveInstruction" and the second alternative shall be used when the "TextInstruction" occurs in a "NegatingInstruction".

18.1.3 The "Keyword" is defined in clause 16.

18.1.4 Application of this final encoding instruction to an enumerated type changes the Unicode character strings that identify one or more enumeration items. Each new string can be specified either as a replacement string (use of "NewText") or as a case change operation (use of "Keyword") to be applied to the identifier of the enumeration item as specified in clause 16.1.5.

18.1.5 When the first alternative of "IdentifierOrAll" ("identifier") is used, the change specified by the "NewTextOrKeyword" applies to the enumeration item with that identifier. When the second alternative of "IdentifierOrAll" (**ALL**) is used, the change applies to all the enumeration items whose identifiers do not appear in this **TEXT** encoding instruction.

18.2 Restrictions

18.2.1 If the final encoding instructions for an ASN.1 type contain a **TEXT** encoding instruction then the type shall be an enumerated type.

18.2.2 Each enumeration identifier shall occur at most once in a **TEXT** encoding instruction. The second alternative of "IdentifierOrAll" (**ALL**) shall occur at most once in a **TEXT** encoding instruction. When the "IdentifierOrAll" in a "TextChange" is **ALL**, the "NewTextOrKeyword" shall be a "Keyword".

18.2.3 The final set of Unicode character strings consisting of the unchanged strings and the new strings identifying the enumeration items shall not contain two identical strings.

19 The UNWRAPPED encoding instruction

19.1 General

19.1.1 The "UnwrappedInstruction" is:

UnwrappedInstruction ::=
UNWRAPPED

19.1.2 Application of this final encoding instruction to a choice type causes the values of the choice type to be encoded as unwrapped (see clause 31).

19.2 Restrictions

19.2.1 If the final encoding instructions for an ASN.1 type contain an **UNWRAPPED** encoding instruction, then the type shall be a choice type that satisfies the conditions in the following clauses, taking account of any final encoding instructions present on the types of the alternatives or on their components and ignoring any constraints that are not JER-visible.

19.2.2 For each of the following kinds of JSON values:

a) the JSON token **null**

b) the JSON token **false**

c) the JSON token **true**

d) a JSON number

e) a JSON string

f) a JSON array

there shall be at most one alternative that produces a JSON value of that kind for one or more abstract values that are permitted by JER-visible constraints.

19.2.3 If the choice type has two or more alternatives that produce a JSON object for one or more abstract values that are permitted by JER-visible constraints, then those alternatives shall be sequence or set types without an extension marker. Each of those types may be either a sequence type or a set type. For any two such types, one of them shall have at least one component not marked **OPTIONAL** or **DEFAULT** whose JSON member name string differs from the JSON member name strings of all the components of the other.

NOTE – The text in this clause excludes any other ASN.1 type that can produce a JSON object for one or more of its abstract values (certain real types, certain bit string types, set-of types with a final **OBJECT** encoding instruction, extensible sequence or set types, choice types without a final **UNWRAPPED** encoding instruction, and certain choice types with a final **UNWRAPPED** encoding instruction).

19.2.4 None of the alternatives of the choice type shall be an open type or an extensible choice type with a final **UNWRAPPED** encoding instruction.

20 Encoding of boolean values

The encoding of a boolean value shall be one of the two JSON tokens **false** and **true**, denoting the boolean values **FALSE** and **TRUE**, respectively.

NOTE – The use of quotation marks around **false** or **true** is forbidden.

21 Encoding of integer values

The encoding of an integer value shall be a JSON number denoting the value, with no fractional part and no exponent.

NOTE – The use of quotation marks around the number is forbidden. Superfluous leading zeros are forbidden.

22 Encoding of enumerated values

22.1 The encoding of an enumerated value shall be a JSON string.

22.2 If the enumerated type has a final **TEXT** encoding instruction and the instruction changes the string assigned to the chosen enumeration item, the Unicode character string denoted by the JSON string shall be the string produced by the instruction; otherwise, the Unicode character string denoted by the JSON string shall be the identifier of the chosen enumeration item.

NOTE – The use of quotation marks around the identifier or the string produced by the **TEXT** encoding instruction is required. The use of escapes is allowed in all JSON strings.

23 Encoding of real values

23.1 General

23.1.1 If the real value is one of the special values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**, it shall be encoded as specified in clause 23.2.

23.1.2 If the real value is 0 or the base of the real value is 2, then the real value shall be encoded as specified in clause 23.3.

23.1.3 If neither 23.1.1 nor 23.1.2 applies, then the encoding of a real value depends on the effective value constraint of the base of the real type, which shall be determined as follows:

a) if there are no JER-visible constraints, the effective value constraint of the base is an integer range that includes both the value 2 and the value 10;

b) if there is an inner type constraint on the real type, the effective value constraint of the base is the one resulting from the JER-visible constraint that the inner type constraint applies to the **base** component;

c) when two or more JER-visible constraints are combined into an **INTERSECTION** construction, they result in a JER-visible constraint (see clause 7.2.4); the effective value constraint of the base is the intersection of all the effective value constraints of the bases in the members of the **INTERSECTION** construction;

d) when two or more JER-visible constraints are combined into a **UNION** construction, they result in a JER-visible constraint (see clause 7.2.5); the effective value constraint of the base is the smallest integer range that includes all the effective value constraints of the bases in the members of the **UNION** construction;

e) when an **EXCEPT** clause is present, it is ignored.

23.1.4 If the effective value constraint of the base includes only the value 10, then the real value shall be encoded as specified in clause 23.3.

23.1.5 Otherwise, the real value shall be encoded as specified in clause 23.4.

23.2 Encoding of the special real values

The real value shall be encoded as a JSON string. The Unicode character string denoted by the JSON string shall be the one specified in Table 2.

Table 2 – Encoding of the real special values

Value	Unicode character string
-0	"-0"
MINUS-INFINITY	"-INF"
PLUS-INFINITY	"INF"
NOT-A-NUMBER	"NaN"

23.3 Encoding as a JSON number

The real value shall be encoded as a JSON number denoting the value.

23.4 Encoding as a JSON object

The real value shall be encoded as a JSON object with a single member. The Unicode character string denoted by the name of the member shall be **"base10Value"** and the value of the member shall be a JSON number denoting the value.

NOTE – The use of escapes is allowed in all JSON strings.

EXAMPLES

Any base-2 value of the real type denoted by **REAL** is encoded as a JSON number.

Any base-10 value of the real type denoted by **REAL** is encoded as a JSON object as specified in clause 23.4.

The value **MINUS-INFINITY** of the real type denoted by **REAL** is encoded as the JSON string **"-INF"**. Note that the use of escapes is allowed in all JSON strings.

Any value of the real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-99999..99999), base (2), exponent (-55..55)})** is encoded as a JSON number. This real type includes a set of base-2 values, including the real value 0, but does not include the special real values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**.

Any value of the real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-999999999999..999999999999), base (10), exponent (-20..20)})** is encoded as a JSON number. This real type includes a set of base-10 values, including the real value positive zero, but does not include the special real values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**.

24 Encoding of bitstring values

24.1 General

The encoding of a bitstring value depends on the JER-visible constraints of the bitstring type as follows:

a) if the lower and upper bounds of the effective size constraint are identical, then the value shall be encoded as specified in clause 24.2;

b) otherwise, if the bitstring type has a JER-visible contents constraint, then the value shall be encoded either as specified in clause 24.3 or as specified in clause 24.4 as a sender's option;

c) otherwise, the value shall be encoded as specified in clause 24.3.

24.2 Encoding of bitstring types with a fixed size

24.2.1 The encoding of a bitstring value with a fixed size shall be a JSON string. The Unicode character string denoted by the JSON string shall consist of an even number of the hexadecimal digits 0123456789abcdefABCDEF, with each consecutive pair of digits encoding one group of eight consecutive bits in the bitstring value. If the length of the bitstring value is not a multiple of 8 bits, the bitstring value shall be encoded as if it contained extra bits, up to the next multiple of 8, all set to zero. If the bitstring value is empty, the JSON string shall be empty.

NOTE – The use of escapes is allowed in all JSON strings.

24.2.2 When Rec. ITU-T X.680 | ISO/IEC 8824-1, 22.7, applies (i.e., the bitstring type is defined with a "NamedBitList"), the bitstring value shall be encoded after trailing 0 bits have been added or removed as necessary to satisfy the effective size constraint.

24.3 Encoding of bitstring types with a variable size

The encoding of a bitstring value with a variable size shall be a JSON object with the following members:

a) **"value"**, whose value shall be a JSON string encoding the bitstring value as specified in clause 24.2.1 for a bitstring type with a fixed size;

b) **"length"**, whose value shall be a JSON number indicating the length of the bitstring value (in bits). The JSON number shall have no fractional part and no exponent.

24.4 Alternative encoding of bitstring types with a JER-visible contents constraint

The bitstring value shall be encoded as a JSON object with a single member. The Unicode character string denoted by the name of the member shall be "containing". The value of the member shall be the JER encoding of the contained value.

25 Encoding of octetstring values

25.1 General

The encoding of an octetstring value depends on the JER-visible constraints and the final encoding instructions of the octetstring type as follows:

a) if the octetstring type has a final **BASE64** encoding instruction, then the value shall be encoded as specified in clause 25.2;

b) otherwise, if the octetstring type has a JER-visible contents constraint, then the value shall be encoded either as specified in clause 25.3 or as specified in clause 25.4 as a sender's option;

c) otherwise, the value shall be encoded as specified in clause 25.3.

25.2 Encoding of an octetstring value as a JSON string containing a Base64 encoding

(This clause applies only to the values of an octetstring type with a final **BASE64** encoding instruction.) The octetstring value shall be encoded as a JSON string. The Unicode character string denoted by the JSON string shall be the Content-Transfer-Encoding, specified in IETF RFC 2045, 6.8, of the octetstring value, except that the 76-character limit does not apply.

NOTE – The use of escapes is allowed in all JSON strings.

25.3 Encoding of an octetstring value as a JSON string containing a hexadecimal encoding

The octetstring value shall be encoded as a JSON string. The Unicode character string denoted by the JSON string shall consist of an even number of the hexadecimal digits 0123456789abcdefABCDEF, with each consecutive pair of digits encoding one octet in the octetstring value. If the octetstring value is empty, the JSON string shall be empty.

NOTE – The use of escapes is allowed in all JSON strings.

25.4 Alternative encoding of an octetstring type with a JER-visible contents constraint

The octetstring value shall be encoded as a JSON object with a single member. The Unicode character string denoted by the name of the member shall be "containing". The value of the member shall be the JER encoding of the contained value.

26 Encoding of the null value

The encoding of the null value shall be the JSON token **null**.

27 Encoding of sequence values

27.1 General

The encoding of a sequence value depends on the presence of a final **ARRAY** encoding instruction on the sequence type. If an **ARRAY** encoding instruction is present, clause 27.2 applies, otherwise clause 27.3 applies.

27.2 Array-based encoding

27.2.1 The encoding of a value of a sequence type having a final **ARRAY** encoding instruction shall be a JSON array having one element for each component of the sequence type, except as specified in clause 27.2.2. First, an element shall be added to the JSON array for each component of the extension root in textual order, except as specified in clause 27.2.2; then, an element shall be added for each extension addition that is a "ComponentType" and for each component contained in an extension addition group, in textual order, except as specified in clause 27.2.2. Each array element corresponding to a component that is present in the sequence value shall be the JER encoding of that component's value. Each array element corresponding to a component that is absent in the sequence value shall be the JSON token **null**.

27.2.2 Any number of instances of the JSON token **null** may be omitted from the end of the JSON array, as a sender's option.

27.3 Object-based encoding

27.3.1 The encoding of a value of a sequence type not having a final **ARRAY** encoding instruction shall be a JSON object that has one member for each component of the sequence value that is present and that may have additional members as specified in clause 27.3.4. Each member of a JSON object has a name, which is a JSON string (see ECMA-404, clause 6).

27.3.2 For each component that is present:

a) if the type of the component has a final **NAME** encoding instruction, the Unicode character string denoted by the name of the member of the JSON object shall be the name produced by the instruction; otherwise, the Unicode character string denoted by the name of the member shall be the identifier of the component;

b) the value of the member shall be the JER encoding of the value of that component.

27.3.3 The components of the sequence value may be added to the encoding in any order.

NOTE – The use of quotation marks around each component identifier is required. The use of escapes is allowed in all JSON strings.

27.3.4 For each component marked **OPTIONAL** or **DEFAULT** whose type is not an open type, an extensible choice type with a final **UNWRAPPED** encoding instruction or a type that produces the JSON token **null** for one of its abstract values and which is absent in the sequence value, an additional member may be included in the encoding of the sequence value. The Unicode character string denoted by the name of the member shall be determined as for a component that is present in the sequence value, and the value of the member shall be the JSON token **null**.

NOTE – The types that produce the JSON token **null**, besides certain open types, are the null type and choice types with a final **UNWRAPPED** encoding instruction having an alternative that produces the JSON token **null** for one of its abstract values.

28 Encoding of sequence-of values

The encoding of a sequence-of value shall be a JSON array having one element for each occurrence of the component of the sequence-of type in the sequence-of value, in the same order. Each element of the JSON array shall be the JER encoding of the corresponding item of the sequence-of value.

29 Encoding of set values

A value of a set type shall be encoded as if the type had been declared a sequence type.

NOTE – The object-based encoding (see clause 27.3) is always used for a set value because a set type is not allowed to have a final **ARRAY** encoding instruction.

30 Encoding of set-of values

30.1 General

The encoding of a set-of value depends on the presence of a final **OBJECT** encoding instruction on the set-of type. If an **OBJECT** encoding instruction is present, clause 30.3 applies, otherwise clause 30.2 applies.

30.2 Array-based encoding

The encoding of a value of a set-of type not having a final **OBJECT** encoding instruction shall be a JSON array having one element for each occurrence of the component of the set-of type in the set-of value, in any order. Each element of the JSON array shall be the JER encoding of one item of the set-of value.

30.3 Object-based encoding

NOTE – A set-of type can have a final **OBJECT** encoding instruction only if the component type of the set-of type is a sequence type with two components, a "key" type (an ASN.1 type encoded as a JSON string) and a "value" type (an arbitrary ASN.1 type), in this order (see clause 17). The identifiers of the two components do not matter.

30.3.1 The encoding of a value of a set-of type having a final **OBJECT** encoding instruction shall be a JSON object that has one member for each occurrence of the component of the set-of type in the set-of value. Each member of a JSON object has a name, which is a JSON string (see ECMA-404, clause 6).

30.3.2 For each item of the set-of value (a sequence value):

a) the Unicode character string denoted by the name of the member shall be the JER encoding of the value of the first component of the sequence value;

NOTE – If the type of the first component is an enumerated type with a final **TEXT** encoding instruction, the Unicode character string denoted by the name of the member is the one that results from the application of that instruction.

b) the value of the member shall be the JER encoding of the second component of the sequence value.

30.3.3 The items of the set-of value may be added to the encoding in any order.

NOTE – The use of escapes is allowed in all JSON strings.

31 Encoding of choice values

31.1 General

The encoding of a choice value depends on the presence of a final **UNWRAPPED** encoding instruction on the choice type. If an **UNWRAPPED** encoding instruction is present, clause 31.2 applies, otherwise clause 31.3 applies.

31.2 Unwrapped encoding

The encoding of a value of a choice type having a final **UNWRAPPED** encoding instruction shall be the encoding of the chosen alternative.

NOTE – In the unwrapped encoding, the choice type is encoded by omitting the left brace ("{"), the name of the chosen alternative, the colon (":"), and the final right brace ("}").The unwrapped encoding relies on the decoder's ability to identify the alternative that was encoded by examining the JER encoding of the alternative since there is no explicit indication of which alternative was encoded.

31.3 Wrapped encoding

31.3.1 The encoding of a value of a choice type not having a final **UNWRAPPED** encoding instruction shall be a JSON object having exactly one member. Each member of a JSON object has a name, which is a JSON string (see ECMA-404, clause 6).

31.3.2 The only member of the JSON object shall be as follows:

a) if the type of the chosen alternative has a final **NAME** encoding instruction, the Unicode character string denoted by the name of the member of the JSON object shall be the name produced by the instruction; otherwise, the Unicode character string denoted by the name of the member shall be the identifier of the chosen alternative;

b) the value of the member shall be the JER encoding of the value of the chosen alternative.

NOTE – The use of quotation marks around the identifier is required.

32 Encoding of object identifier values

The encoding of an object identifier value shall be a JSON string. The Unicode character string denoted by the JSON string shall be an instance of the "XMLObjectIdentifierValue" production denoting the value (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 32.3).

33 Encoding of relative object identifier values

The encoding of a relative object identifier value shall be a JSON string. The Unicode character string denoted by the JSON string shall be an instance of the "XMLRelativeOIDValue" production denoting the value (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 33.3).

34 Encoding of values of the internationalized resource reference type

The encoding of a value of the internationalized resource reference type shall be a JSON string. The Unicode character string denoted by the JSON string shall be an instance of the "XMLIRIValue" production denoting the value (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 34.3).

35 Encoding of values of the relative internationalized resource reference type

The encoding of a value of the relative internationalized resource reference type shall be a JSON string. The Unicode character string denoted by the JSON string shall be an instance of the "XMLRelativeIRIValue" production denoting the value (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 35.3).

36 Encoding of values of the embedded-pdv type

The encoding of a value of the embedded-pdv type shall consist of the JER encoding of the sequence type specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 36.5. The value of the **data-value** component (of type **OCTET STRING**) shall be the octets which form the complete encoding of the single data value referenced in Rec. ITU-T X.680 | ISO/IEC 8824-1, 36.3 a).

37 Encoding of values of the external type

37.1 The encoding of a value of the external type shall consist of the encoding of the sequence type specified in Rec. ITU-T X.691 | ISO/IEC 8825-2, 29.1.

37.2 Rec. ITU-T X.691 | ISO/IEC 8825-2, clauses 29.2 to 29.11, apply, with the following modifications:

a) the reference to "This Recommendation | International Standard" (meaning Rec. ITU-T X.691 | ISO/IEC 8825-2) present in those clauses shall be read as a reference to this Recommendation | International Standard;

b) the reference to Rec. ITU-T X.691 | ISO/IEC 8825-2, 11.2 (encoding of open type fields) present in those clauses shall be read as a reference to clause 41 of this Recommendation | International Standard.

38 Encoding of values of the restricted character string types

38.1 The encoding of a character string value of one of the types **`IA5String`**, **`ISO646String`**, **`VisibleString`**, **`NumericString`**, **`PrintableString`**, **`BMPString`**, **`UniversalString`**, and **`UTF8String`**, shall be a JSON string. The Unicode character string denoted by the JSON string shall be the character string value.

NOTE – The use of escapes is allowed in all JSON strings.

38.2 A value of one of the remaining restricted character string types (**`TeletexString`**, **`T61String`**, **`VideotexString`**, **`GraphicString`**, and **`GeneralString`**) shall be encoded as if it were an octetstring value consisting of the octets specified in Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.23.5.

39 Encoding of values of the unrestricted character string type

The encoding of a value of the **`CHARACTER STRING`** type shall consist of the encoding of the type defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 44.5. The value of the **`string-value`** component (of type **`OCTET STRING`**) shall be the octets which form the complete encoding of the character string value referenced in Rec. ITU-T X.680 | ISO/IEC 8824-1, 44.3 a).

40 Encoding of values of the time types

The encoding of a time value shall be a JSON string. The Unicode character string denoted by the JSON string shall be an instance of the "XMLTimeValue" production denoting the value (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 38.3.2).

41 Encoding of open type values

NOTE – An open type is an ASN.1 type that can take any abstract value of any ASN.1 type. Each value of an open type consists of:

a) a contained type; and

b) a value of the contained type.

The encoding of an open type value shall be the encoding of the value of the contained type.

42 Object identifier values referencing the encoding rules

42.1 The encoding rules specified in this Recommendation | International Standard can be referenced and applied whenever there is a need to specify an unambiguous character string representation for the values of a single identified ASN.1 type.

42.2 The following object identifier and object descriptor values are assigned to identify the encoding rules specified in this Recommendation | International Standard:

For JER:

```
{joint-iso-itu-t asn1 (1) jer-encoding (7) }
"JER encoding of a single ASN.1 type"
```

Annex A

Examples of JER encodings

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

This annex illustrates the use of the JavaScript Object Notation Encoding Rules specified in this Recommendation | International Standard by showing the representation in octets of a (hypothetical) personnel record which is defined using ASN.1. It also contains additional examples of JER encodings.

A.1 ASN.1 description of the record structure

The structure of the hypothetical personnel record is formally described as follows using ASN.1 specified in Rec. ITU-T X.680 | ISO/IEC 8824-1. This is identical to the example defined in Rec. ITU-T X.690 | ISO/IEC 8825-1, Annex A.

```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
      name            Name,
      title           [0] VisibleString,
      number          EmployeeNumber,
      dateOfHire      [1] Date,
      nameOfSpouse    [2] Name,
      children        [3] IMPLICIT
            SEQUENCE OF ChildInformation DEFAULT {} }
ChildInformation ::= SET
      { name               Name,
        dateOfBirth        [0] Date}
Name ::= [APPLICATION 1] IMPLICIT SEQUENCE
      { givenName               VisibleString,
        initial            VisibleString,
        familyName         VisibleString}
EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER
Date ::= [APPLICATION 3] IMPLICIT VisibleString  -- YYYYMMDD
```

NOTE – Tags are used in this example only because it was felt appropriate to use the identical example to that which appeared in the earliest version of Rec. ITU-T X.680 | ISO/IEC 8824-1. The tags used in this example have no effect on the JER encodings.

A.2 ASN.1 description of a record value

The value of John Smith's personnel record is formally described as follows using the basic ASN.1 value notation:

```
{     name            {givenName "John", initial "P", familyName "Smith"},
      title                "Director",
      number               51,
      dateOfHire      "19710917",
      nameOfSpouse         {givenName "Mary", initial "T", familyName "Smith"},
      children
                           {{name {givenName "Ralph", initial "T", familyName "Smith"},
                                dateOfBirth "19571111"},
                            {name {givenName "Susan", initial "B", familyName "Jones"},
                                dateOfBirth "19590717"}}}
```

A.3 Example JER representation of this record value

A possible representation of the record value given in A.2 (after applying the JavaScript Object Notation Encoding Rules defined in this Recommendation | International Standard) is as follows.

```
{
      "name" : {
            "givenName" : "John",
            "initial" : "P",
            "familyName" : "Smith"
      },
      "title" : "Director",
      "number" : 51,
      "dateOfHire" : "19710917",
      "nameOfSpouse" : {
            "givenName" : "Mary",
```

```
                "initial" : "T",
                "familyName" : "Smith"
          },
          "children" : [
                {
                      "name" : {
                            "givenName" : "Ralph",
                            "initial" : "T",
                            "familyName" : "Smith"
                      },
                      "dateOfBirth": "19571111"
                },
                {
                      name : {
                            "givenName" : "Susan",
                            "initial" : "B",
                            "familyName" : "Jones"
                      },
                      "dateOfBirth" : "19590717"
                }
          ]
    }
```

A.4 Additional examples of JER encodings

In the following examples, it is understood that whenever a JSON string appears in an encoding, any other JSON string denoting the same Unicode character string (e.g., by the use of escapes) can appear in its place. This also applies to the fixed JSON strings that are specified in this Recommendation | International Standard, such as "**value**" and "**NaN**". It is also understood that whenever a JSON object appears in an encoding, its members can occur in an arbitrary order.

Consider the following ASN.1 definitions:

```
MyInteger ::= INTEGER (0..1500)

MyEnumerated ::= ENUMERATED { red, yellow, green }

MyReal ::= REAL (0 |
     WITH COMPONENTS { mantissa (-999999999999.. 999999999999), base (10), exponent (-
100..100)})

MyBitString1 ::= BIT STRING (SIZE (10))

MyBitString2 ::= BIT STRING (SIZE (10), ...)

MyOctetString ::= OCTET STRING (SIZE (4))

MySequence1 ::= SEQUENCE {
     a     INTEGER OPTIONAL,
     b     BOOLEAN,
     c     UTF8String
}

MySequence2 ::= SEQUENCE {
     x     MyReal,
     y     MySequence1,
     ...
}

MySequenceOf1 ::= SEQUENCE (SIZE (1..16)) OF INTEGER

MySequenceOf2 ::= SEQUENCE OF MySequence1

MyChoice ::= CHOICE {
     a     MySequence1,
     b     UniversalString
}
```

The boolean value

```
     x BOOLEAN ::= TRUE
```


will be encoded as

```
true
```

The integer values

```
x1 INTEGER ::= 100
x2 MyInteger ::= 100
```

will both be encoded as

```
100
```

The enumerated value

```
x MyEnumerated ::= red
```

will be encoded as

```
"red"
```

The real value

```
x REAL ::= 14
```

will be encoded as

```
{ "base10Value" : 14 }
```

because the value denoted by **x** is a base-10 real abstract value and the real type is unconstrained.

The real value

```
x REAL ::= { mantissa 14, base 2, exponent 0 }
```

will be encoded as

```
14
```

because the value denoted by **x** is a base-2 real abstract value (14×2^0).

The real value

```
x REAL ::= NOT-A-NUMBER
```

will be encoded as

```
"NaN"
```

The real value

```
x MyReal ::= 14.56
```

will be encoded as

```
14.56
```

because the constraint restricts the base to 10. This value can also be encoded as any other JSON number denoting the same numeric value (e.g., **0.145600e2**).

The value

```
x MyBitString1 ::= '0101010101'B
```

will be encoded as

```
"5540"
```

because the length (10) is implied by the size constraint.

The value

```
x BIT STRING ::= '0101010101'B
```

will be encoded as

```
{ "length" : 10, "value" : "5540" }
```

The value

```
x MyBitString2 ::= '0101010101'B
```

will be encoded as

```
{ "length" : 10, "value" : "5540" }
```

because the size constraint contains an extension marker and therefore is not JER-visible.

The values

```
x1 OCTET STRING ::= 'EABC001E'H
x2 MyOctetString ::= 'EABC001E'H
```

will both be encoded as

```
"EABC001E"
```

The value
```
x NULL ::= NULL
```
will be encoded as
```
null
```

The value
```
x MySequence1 ::= { a 123, b TRUE, c "Hello" }
```
will be encoded as
```
{ "a" : 123, "b" : true, "c" : "Hello" }
```

The value
```
x MySequence1 ::= { b TRUE, c "Hello" }
```
will be encoded as
```
{ "b" : true, "c" : "Hello" }
```
or as the same JSON object with its members encoded in a different order (**{ "c" : "Hello", "b" : true }**).

The value
```
x MySequence2 ::= {  x -3.1415, y { b TRUE, c "Hello" }}
```
will be encoded as
```
{ "x" : -3.1415, "y" : { "b" : true, "c" : "Hello" }}
```
(an extension marker in a sequence type has no effect on the encoding).

The value
```
x MySequenceOf1 ::= {  1, 2, 3 }
```
will be encoded as
```
[ 1, 2, 3 ]
```

The value
```
x MySequenceOf2 ::= {{ b TRUE, c "one" }, { a 99, b FALSE, c "two" }}
```
will be encoded as
```
[{ "b" : true, "c" : "one" }, { "a" : 99, "b" : false, "c" : "two" }]
```

The value
```
x MyChoice ::= b : "mouse"
```
will be encoded as
```
{ "b" : "mouse" }
```

The values
```
x1 OBJECT IDENTIFIER ::= { iso standard 8571 application-context (1) }
x2 OBJECT IDENTIFIER ::= { 1 0 8571 1 }
```
will both be encoded as
```
"1.0.8571.1"
```

The values
```
x1 VisibleString ::=      "ABCDEabcde12345(/)"
x2 IA5String ::=          "ABCDEabcde12345(/)"
x3 BMPString ::=          "ABCDEabcde12345(/)"
x4 UTF8String ::=         "ABCDEabcde12345(/)"
x5 UniversalString ::=    "ABCDEabcde12345(/)"
x6 PrintableString ::=    "ABCDEabcde12345(/)"
```
will all be encoded as
```
"ABCDEabcde12345(/)"
```

The value
```
x TIME ::= "2014-12-31T23:59:59"
```
will be encoded as
```
"2014-12-31T23:59:59"
```


