JSON Encoding Rules

Draft 2015-12-30

CONTENTS

Introduction

The publications Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4 together describe Abstract Syntax Notation One (ASN.1), a notation for the definition of messages to be exchanged between peer applications.

This Technical Specification defines encoding rules that may be applied to values of ASN.1 types defined using the notation specified in the above-mentioned publications. 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 are more than one set of encoding rules that can be applied to values of ASN.1 types. This Technical Specification defines a set of JSON Encoding Rules, so called because the encodings they produce are instances of the JSON grammar specified in ECMA 404.

Clauses 7 to 29 specify the JSON/ER encoding of ASN.1 types.

Annex A is informative and contains examples of JSON/ER encodings.

Information technology – ASN.1 encoding rules: Specification of JSON Encoding Rules (JSON/ER)

1 Scope

This Technical Specification specifies a set of JSON Encoding Rules (JSON/ER) 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, 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 Technical Specification:

- 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.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Technical Specification. 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 Technical Specification 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.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.680 (2008) | ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- Recommendation ITU-T X.681 (2008) | ISO/IEC 8824-2:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- Recommendation ITU-T X.682 (2008) | ISO/IEC 8824-3:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- Recommendation ITU-T X.683 (2008) | ISO/IEC 8824-4:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- Recommendation ITU-T X.690 (2008) | ISO/IEC 8825-1:2008, *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 (2008) | ISO/IEC 8825-2:2008, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*

2.2 Additional references

- *ISO International Register of Coded Character Sets to be Used with Escape Sequences.*
- ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS).*
- ECMA Standard ECMA-404 (2013) – The JSON Data Interchange Format

3 Definitions

For the purposes of this Technical Specification, the following definitions apply.

3.1 Specification of Basic Notation

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

3.2 Information Object Specification

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

3.3 Constraint Specification

This Technical Specification 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 Technical Specification makes use of the following term defined in Rec. ITU-T X.683 | ISO/IEC 8824-4:

– variable constraint.

3.5 Basic Encoding Rules (BER)

This Technical Specification 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 Technical Specification 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 JSON/ER or which is generated by JSON/ER decoding.

3.7.2 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 JSON/ER-visible constraints (see 7.2.7).

3.7.3 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 JSON/ER-visible constraints (see 7.2.8).

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

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

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

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

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

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

3.7.9 **JSON/ER-visible constraint**: An instance of use of the ASN.1 constraint notation that affects the JSON/ER encoding of a value.

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

4 Abbreviations

For the purposes of this Technical Specification, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules of ASN.1
JSON/ER	JSON Encoding Rules of ASN.1
PER	Packed Encoding Rules of ASN.1
PDU	Protocol Data Unit

5 Encodings specified by this Technical Specification

5.1 This Technical Specification specifies a set of encoding rules which 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 JSON/ER 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 OSI presentation context identifiers.

5.3 If a type encoded with JSON/ER 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 the above character transfer syntaxes are relay-safe.

5.4 JSON/ER encodings are self-delimiting. Encodings are always a whole multiple of eight 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 JSON/ER, 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.

6 Conformance

6.1 Dynamic conformance for the JSON Encoding Rules is specified by clauses 7 to 29.

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

6.3 Alternative encodings are permitted by the JSON Encoding Rules as encoder's options. Decoders that claim conformance to JSON/ER shall support all JSON/ER encoding alternatives.

6.4 The rules in this Technical Specification 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 Technical Specification for the applicable transfer syntax.

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

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, 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 which arise through the application of JSON/ER-visible constraints;

e) whether the type of a component is an open type.

7.2 Constraints

NOTE – The fact that some ASN.1 constraints may not be JSON/ER-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 Technical Specification 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 JSON/ER-visible:

a) 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`**;

b) non-extensible size constraints on bitstring types;

c) an inner type constraint applying an JSON/ER-visible constraint (including one that restricts the mantissa, base, or exponent of a real type);

d) a contained subtype constraint in which the constraining type carries an JSON/ER-visible constraint.

7.2.2 All other constraints are not JSON/ER-visible. In particular, the following constraints are not JSON/ER-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, 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) vale and value range constraints on integer types;

m) constraints on real types except those specified in 7.2.1 (a) and (c);

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

o) an inner type constraint applied to a component of an unrestricted character string or embbeded-pdv type or external type;

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 JSON/ER-visible. If the last subtype constraint of the series of constraints is JSON/ER-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 constaints 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 JSON/ER-visible is part of an **`INTERSECTION`** construction, then the resulting constraint is JSON/ER-visible, and consists of the **`INTERSECTION`** of all the JSON/ER-visible parts (with the non-JSON/ER-visible parts ignored).

7.2.5 If a constraint that is not JSON/ER-visible is part of a **`UNION`** construction, then the resulting constraint is not JSON/ER-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 JSON/ER-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 JSON/ER-visible constraints present in the type definition and ignoring any constraints that are not JSON/ER-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 JSON/ER-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 JSON/ER-visible constraints present in the type definition and ignoring any constraints that are not JSON/ER-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. JSON/ER-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 JSON/ER 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 which 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 8 to 29 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.

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

7.4.2 This Technical Specification 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 which has been tagged.

7.4.3 An encoding prefixed type is encoded according to the type which has been prefixed.

7.4.4 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 JSON/ER-visible.

7.4.5 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 Production of a complete JSON/ER encoding

7.5.1 If an ASN.1 type is encoded using JSON/ER 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 either subclause 7.5.1.1 or subclause 7.5.1.2, as a sender's option, shall be applied to all the encodings of its abstract values.

7.5.1.1 (Wrapped JSON/ER encoding) A JSON object with a single member shall be created. The Unicode character string denoted by the name of the member shall be as follows:

a) if the "Type" being encoded is a "typereference", the "typereference";
b) if the "Type" being encoded is an "ExternalTypeReference", the "typereference" in the "ExternalTypeReference";
c) otherwise, the string "_".

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

7.5.1.2 (Unwrapped JSON/ER encoding) 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.

NOTE – A decoder can distinguish a wrapped from an unwrapped JSON/ER encoding by the fact that the former consists of a JSON object whose first (and only) member has a name beginning with an uppercase letter or the character "_". No other JSON object member produced by these encoding rules can have a name beginning with an uppercase letter or the character "_".

7.5.2 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 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.

9 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.

10 Encoding of enumerated values

The encoding of an enumerated value shall be a JSON string. 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 is required. The use of escapes is allowed in all JSON strings.

11 Encoding of real values

11.1 The encoding of a real value depends on the effective value constraints of the mantissa, base, and exponent of the real type, which shall be determined as follows:

a) if there are no JSON/ER-visible constraints, the effective value constraints of the mantissa and exponent have no finite lower and upper bounds, and the effective value constraint of the base is the integer range 2..10;

b) if there is an inner type constraint that applies JSON/ER-visible constraints to one or more components of the real type (**`mantissa`**, **`base`**, and **`exponent`**), the effective value constraint of each component is the one resulting from the inner type constraint;

c) when two or more JSON/ER-visible constraints are combined into an **`INTERSECTION`** construction, they result in an JSON/ER-visible constraint (see 7.2.4); the effective value constraint of the mantissa in that JSON/ER-visible constraint is the intersection of all the effective value constraints of the mantissas in the members of the **`INTERSECTION`** construction; the effective value constraint of the base and the effective value constraint of the exponent are determined in the same way;

d) when two or more JSON/ER-visible constraints are combined into a **`UNION`** construction, they result in an JSON/ER-visible constraint (see 7.2.5; the effective value constraint of the mantissa in that JSON/ER-visible constraint is the smallest integer range that includes all the effective value constraints of the mantissas in the members of the **`UNION`** construction; the effective value constraint of the base and the effective value constraint of the exponent are determined in the same way;

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

11.2 If all of the following are true:

a) the effective value constraint of the base is the fixed value 2;

b) the lower bound of the effective value constraint of the mantissa is greater than or equal to $-2^{53} + 1$ (–9007199254740991) and its upper bound is less than or equal to $2^{53} - 1$ (9007199254740991); and

c) the lower bound of the effective value constraint of the exponent is greater than or equal to -1074 and its upper bound is less than or equal to 971,

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

11.3 If all of the following are true:

a) the effective value constraint of the base is the fixed value 10;

b) the lower bound of the effective value constraint of the mantissa is greater than or equal to –17976931348623157 and its upper bound is less than or equal 17976931348623157; and

c) the lower bound of the effective value constraint of the exponent is greater than or equal to –323 and its upper bound is less than or equal to 292,

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

11.4 Otherwise, the real value shall be encoded as a JSON object with the following members:

a) **`"value"`**, whose value shall be a JSON number denoting the value;

b) **`"base"`**, whose value shall be a JSON number denoting the base of the real value (either 2 or 10);

c) **`"specialValue"`**, whose value shall be one of the following:

1) **`"INF"`**, denoting the special real value **`PLUS-INFINITY;`**

2) **`"-INF"`**, denoting the special real value **`MINUS-INFINITY;`**

3) **`"NaN"`**, denoting the special real value **`NOT-A-NUMBER;`**

4) **"-0"**, denoting the special real value -0 (negative 0).

Either the member **"value"** or the member **"specialValue"** (but not both) shall be present. The member **"base"** shall be present if and only if the member **"value"** is present and the effective value constraint of the base permits both the value 2 and the value 10.

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

EXAMPLES

The real type denoted by **REAL** is encoded as a JSON object as specified in 11.4 because the base is not constrained to 2 and the special values are not excluded.

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-99999..99999), base (2), exponent (-20..20)})** is encoded as a JSON number. This real type includes the real value 0 but does not include the special real values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**.

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-999999999999..999999999999), base (2), exponent (-20..20)})** is encoded as a JSON number. This real type includes the real value positive zero but does not include the special real values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**.

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-99999..99999), exponent (-20..20)})** is encoded as a JSON object as specified in 11.4 because the base is not constrained to either 2 or 10.

12 Encoding of bitstring values

12.1 General

The encoding of a bitstring value depends on the effective size constraint of the bitstring type (see 7.2.8). If the lower and upper bounds of the effective size constraint are identical, 12.2 applies, otherwise 12.3 applies.

12.2 Encoding of bitstring types with a fixed size

12.2.1 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.

12.2.2 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.

12.3 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 **12.2.1** 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.

13 Encoding of octetstring values

The encoding of an octetstring value 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 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.

14 Encoding of the null value

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

15 Encoding of sequence values

15.1 The encoding of a sequence value 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 15.2. Each member of a JSON object has a name, which is a JSON string (see ECMA 404, clause 6). The Unicode character string denoted by the name of each member of the JSON object shall be the identifier of a component of the sequence type and the value of the member shall be the JSON/ER encoding of the value of that component. The components 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.

15.2 For each optional component of the sequence type whose type is neither the null type nor an open type 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 the identifier of the component and the value of the member shall be the JSON token **`null`**.

16 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 JSON/ER encoding of the corresponding occurrence of the sequence-of value.

17 Encoding of set values

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

18 Encoding of set-of values

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

19 Encoding of choice values

The encoding of a choice value 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). The Unicode character string denoted by the name of the only member of the JSON object shall be the identifier of the chosen alternative of the choice type, and the value of the member shall be the JSON/ER encoding of the value of that alternative.

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

20 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).

21 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).

22 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).

23 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, 35.3).

24 Encoding of values of the embedded-pdv type

The encoding of a value of the embedded-pdv type shall consist of the JSON/ER 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).

25 Encoding of values of the external type

25.1 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.

25.2 Rec. ITU-T X.691 | ISO/IEC 8825-2, subclauses 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 subclauses shall be read as a reference to this Technical Specification;

b) the reference to Rec. ITU-T X.691 | ISO/IEC 8825-2, 11.2 (encoding of open type fields) present in those subclauses shall be read as a reference to clause 29 of this Technical Specification.

26 Encoding of values of the restricted character string types

26.1 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.

26.2 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.

27 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).

28 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).

29 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.

Annex A

Examples of JSON/ER encodings

(This annex does not form an integral part of this Technical Specification)

This annex illustrates the use of the JSON Encoding Rules specified in this Technical Specification by showing the representation in octets of a (hypothetical) personnel record which is defined using ASN.1. It also contains additional examples of JSON/ER encodings.

A.1 ASN.1 description of the record structure

The structure of the hypothetical personnel record is formally described below 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 JSON/ER encodings.

A.2 ASN.1 description of a record value

The value of John Smith's personnel record is formally described below 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 JSON/ER representation of this record value

The representation of the record value given above (after applying the JSON Encoding Rules defined in this Technical Specification) is shown below.

```
{
    "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 JSON/ER encodings

In the examples below, 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 Technical Specification, such as **"specialValue"** 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

{ "value" : 14, "base" : 10 }

The real value

x REAL ::= NOT-A-NUMBER

will be encoded as

{ "specialValue" : "NaN" }

The real value

x MyReal ::= 14.56

will be encoded as

14.56

because the constraint restricts the base to 10 and excludes all the four special values. 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 JSON/ER-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"