Network Working Group
Internet-Draft
Updates: 7049 (if approved)
Intended status: Standards Track
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
S. Leonard
Penango, Inc.

July 08, 2016

Concise Binary Object Representation (CBOR) Tags and Techniques for Object Identifiers, Enumerations, Binary Entities, Regular Expressions, and Sets

draft-bormann-cbor-tags-oid-04

Abstract

The Concise Binary Object Representation (CBOR, RFC 7049) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

Useful tags and techniques have emerged since the publication of RFC 7049; the present document makes use of CBOR's built-in major types to define and refine several useful constructs, without changing the wire protocol. This document adds object identifiers (OIDs) to CBOR with CBOR tags <<O>> and <<R>> [values TBD]. It is intended as the reference document for the IANA registration of the CBOR tags so defined. Useful techniques for enumerations and sets are presented (without new tags). As the documentation for MIME entities (tag 36) and regular expressions (tag 35) RFC 7049 left much out, this document provides more comprehensive specifications.

Status of This Memo

Table of Contents

1. Introduction

The Concise Binary Object Representation (CBOR, [RFC7049]) provides for the interchange of structured data without a requirement for a pre-agreed schema. RFC 7049 defines a basic set of data types, as well as a tagging mechanism that enables extending the set of data types supported via an IANA registry.

Useful tags and techniques have emerged since the publication of [RFC7049]. This document makes use of CBOR's built-in major types to provide for several useful constructs without changing the wire protocol.

The original focus of this work was to add support for object identifiers (OIDs, [X.680]), which many IETF protocols carry. The ASN.1 Basic Encoding Rules (BER, [X.690]) specify the binary encodings of both object identifiers and relative object identifiers. The contents of these encodings can be carried in a CBOR byte string. This document defines two CBOR tags that cover the two kinds of ASN.1 object identifiers encoded in this way. The tags can also be applied to arrays and maps for more articulated identification purposes. It is intended as the reference document for the IANA registration of the tags so defined. To promote the use and usefulness of OIDs in CBOR, a new arc is also proposed.

This document covers several useful techniques that have been or are being developed as implementers are applying CBOR to practical problems. Enumerations have found wide utility in CBOR, despite CBOR's lack of a native enumerated type. A section covers the advantages of choosing built-in types, with additional consideration for using the newly-defined object identifier types in enumerations. CBOR also lacks a native set type (in the mathematical sense of an arbitrary unordered collection of items), but has a more powerful alternative in its native map type. A section covers how to adapt the map type to express set and multiset semantics.

Finally, this document covers the semantics of existing tags in [RFC7049] that were somewhat underspecified. "Tag 36 is for MIME messages", but the reference [RFC2045] actually defines a different construct, the MIME entity, that finds expression in a variety of message-oriented Internet protocols. Similarly, "Tag 35 is for regular expressions", but the references to Perl Compatible Regular Expressions (PCRE) and JavaScript syntax (ECMA-262) are not compatible with each other. Two sections cover the subtleties of items tagged with these tags, and so update [RFC7049] without changing the basic CBOR wire protocol.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The terminology of RFC 7049 applies; in particular the term "byte" is used in its now customary sense as a synonym for "octet".

2. Object Identifiers

The International Object Identifier tree [X.660] is a hierarchically managed space of identifiers, each of which is uniquely represented as a sequence of unsigned integers ("sub-identifiers") [X.680]. While these sequences can easily be represented in CBOR arrays of unsigned integers, a more compact representation can often be achieved by adopting the widely used representation of object identifiers defined in BER; this representation may also be more amenable to processing by other software making use of object identifiers.

BER represents the sequence of unsigned integers by concatenating self-delimiting [RFC6256] representations of each of the sub-identifiers in sequence.

ASN.1 distinguishes absolute object identifiers (ASN.1 Type "OBJECT IDENTIFIER"), which begin at a root arc ([X.660] Clause 3.5.21), from relative object identifiers (ASN.1 Type "RELATIVE-OID"), which begin relative to some object identifier known from context ([X.680] Clause 3.8.63). As a special optimization, BER combines the first two integers in an absolute object identifier into one numeric identifier by making use of the property of the hierarchy that the first arc has only three integer values (0, 1, and 2), and the second arcs under 0 and 1 are limited to the integer values between 0 and 39. (The root arc "joint-iso-itu-t(2)" has no such limitations on its second arc.) If X and Y are the first two integers, the single integer actually encoded is computed as:

```
X * 40 + Y
```

The inverse transformation (again making use of the known ranges of X and Y) is applied when decoding the object identifier.

Since the semantics of absolute and relative object identifiers differ, this specification defines two tags:

Tag <<O>> (value TBD): tags a byte string as the [X.690] encoding of an absolute object identifier (simply "object identifier" or "OID").

Tag <<R>> (value TBD): tags a byte string as the [X.690] encoding of a relative object identifier (also "relative OID").

2.1. Requirements on the byte string being tagged

A byte string tagged by <<O>> or <<R>> MUST be a syntactically valid BER representation of an object identifier. Specifically:

- its first byte, and any byte that follows a byte that has the most significant bit unset, MUST NOT be 0x80 (this requirement excludes expressing the sub-identifiers with anything but the shortest form)

- its last byte MUST NOT have the most significant bit set (this requirement excludes an incomplete final sub-identifier)

If either of these invalid conditions are encountered, they MUST be treated as decoding errors. Comparing two OIDs or relative OIDs for equality in a byte-for-byte fashion may not be safe before these checks succeed on at least one of them (this includes the case where one of them is a local constant); a process implementing an exclusion list MUST check for decoding errors first.

[X.680] restricts RELATIVE-OID values to have at least one sub-identifier (array element). This specification permits empty relative object identifiers; they may still be excluded by application semantics.

[RFC7049] permits byte strings to be indefinite-length, with chunks divided at arbitrary byte boundaries. This contrasts with text strings, where each chunk in an indefinite-length text string is required be well-formed UTF-8 on its own: splitting the octets of a UTF-8 character encoding between chunks is not allowed.

By analogy to this principle and to Clauses 8.9.1 and 8.20.1 of [X.690], the byte strings carrying the OIDs and relative OIDs are also to be treated as indivisible units: They MUST be encoded in definite-length form; indefinite-length form is treated as an encoding error (and the same considerations as above apply). (An added convenience is that CBOR encodings can be searched through efficiently for specific object identifiers without initiating the decoding process.)

We provide "binary regular expression" forms for implementation convenience. Unlike typical regular expressions that operate on character sequences, the following regular expressions take bytes as their domain, so they can be applied directly to CBOR byte strings.

For byte strings with tag <<O>>:

```
/^((?:[\x81-\xFF][\x80-\xFF]*)?[\x00-\x7F])+$/
```

For byte strings with tag <<R>>:

```
/^((?:[\x81-\xFF][\x80-\xFF]*)?[\x00-\x7F])*$/
```

Putative CBOR data that fails these tests SHALL be rejected as improperly coded.

Another (possibly more efficient) way to validate the byte strings is to hunt for prohibited patterns.

For byte strings with tag <<O>>:

```
/^$|(?:^|[\x00-\x7F])\x80|[\x80-\xFF]$/
```

or with lookbehind:

```
/^$|^\x80|(?<[\x00-\x7F])\x80|(?<[\x80-\xFF])$/
```

For byte strings with tag <<R>>:

```
/(?:^|[\x00-\x7F])\x80|[\x80-\xFF]$/
```

or with lookbehind:

```
/^\x80|(?<[\x00-\x7F])\x80|(?<[\x80-\xFF])$/
```

Putative CBOR data that passes these tests SHALL be rejected as improperly coded.

(It is worth pointing out that these tests, when optimally implemented, ought to be markedly faster than UTF-8 validation.)

3. Examples

In the following examples, we are using tag number 6 for <<O>> and tag number 7 for <<R>>. See Section 16.2.

3.1. Encoding of the SHA-256 OID

ASN.1 Value Notation
{ joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) sha256(1) }

Dotted Decimal Notation (also XML Value Notation)
2.16.840.1.101.3.4.2.1

```
06                                  # UNIVERSAL TAG 6
   09                               # 9 bytes, primitive
      60 86 48 01 65 03 04 02 01    # X.690 Clause 8.19
#      |   840  1  |  3  4  2  1     show component encoding
#   2.16         101
```

Figure 1: SHA-256 OID in BER

```
C6                                  # 0b110_00110: mt 6, tag 6
   49                               # 0b010_01001: mt 2, 9 bytes
      60 86 48 01 65 03 04 02 01    # X.690 Clause 8.19
```

Figure 2: SHA-256 OID in CBOR

3.2. Encoding of a UUID OID

```
UUID
8b0d1a20-dcc5-11d9-bda9-0002a5d5c51b

ASN.1 Value Notation
{ joint-iso-itu-t(2) uuid(25)
geomicaGPAS(184830721219540099336690027854602552603) }

Dotted Decimal Notation (also XML Value Notation)
2.25.184830721219540099336690027854602552603

06                                  # UNIVERSAL TAG 6
   14                               # 20 bytes, primitive
      69 82 96 8D 8D 88 9B CC A8 C7 B3 BD D4 C0 80 AA AE D7 8A 1B
#      |                  184830721219540099336690027854602552603
#   2.25
```

Figure 3: UUID in an object identifier, in BER

```
C6                                  # 0b110_00110: mt 6, tag 6
   54                               # 0b010_10100: mt 2, 20 bytes
      69 82 96 8D 8D 88 9B CC A8 C7 B3 BD D4 C0 80 AA AE D7 8A 1B
```

Figure 4: UUID in an object identifier, in CBOR

3.3. Encoding of a MIB Relative OID

Given some OID (e.g., "lowpanMib", assumed to be "1.3.6.1.2.1.226" [RFC7388]), to which the following is added:

```
ASN.1 Value Notation (not suitable for diagnostic notation)
{ lowpanObjects(1) lowpanStats(1) lowpanOutTransmits(29) }
```

```
Dotted Decimal Notation (diagnostic notation; see Section 5)
.1.1.29

0D                              # UNIVERSAL TAG 13
   03                           # 3 bytes, primitive
      01 01 1D                  # X.690 Clause 8.20
#      1  1 29                    show component encoding
```

Figure 5: MIB relative object identifier, in BER

```
C7                              # 0b110_00110: mt 6, tag 7
   43                           # 0b010_01001: mt 2 (bstr), 3 bytes
      01 01 1D                  # X.690 Clause 8.20
```

Figure 6: MIB relative object identifier, in CBOR

This relative OID saves seven bytes compared to the full OID encoding.

4. Discussion

Staying close to the way object identifiers are encoded in ASN.1 BER makes back-and-forth translation easy. Object identifiers in IETF protocols are serialized in dotted decimal form or BER form, so there is an advantage in not inventing a third form. Also, expectations of the cost of encoding object identifiers are based on BER; using a different encoding might not be aligned with these expectations. If additional information about an OID is desired, lookup services such as the OID Resolution Service (ORS) [X.672] and the OID Repository [OID-INFO] are available.

This specification allocates two numbers out of the single-byte tag space. This use of code point space is justified by the wide use of object identifiers in data interchange. For most common OIDs in use (namely those whose contents encode to less than 24 bytes), the CBOR encoding will match the efficiency of [X.690]. (This preliminary conclusion is likely to generate some discussion, see Section 16.2.)

5. Diagnostic Notation

Implementers will likely want to see OIDs and relative OIDs in their "natural forms" (as sequences of decimal unsigned integers) for diagnostic purposes. Accordingly, this section defines additional syntactic elements that can be used in conjunction with the diagnostic notation described in Section 6 of [RFC7049].

An object identifier may be written in ASN.1 value notation (with enclosing braces and secondary identifiers, ObjectIdentifierValue of

Clause 32.3 of [X.680]), or in dotted decimal notation with at least three arcs. Both examples are shown in Section 3. The surrounding tag notation is not to be used, because the tag is implied. The ASN.1 value notation for OIDs does not overlap with JSON object notation for CBOR maps, because at least two arcs are required for a valid OID.

A relative object identifier may be written in dotted decimal notation or in ASN.1 value notation, in both cases prefixed with a dot as shown in Section 3.3. The surrounding tag notation is not to be used, because the tag is implied.

The notation in this section may be employed in addition to the basic notation, which would be a tagged binary string.

RFC 7049 diagnostic notation	6(h'2b0601')	7(h'0601')
Dotted decimal notation	1.3.6.1	.6.1
ASN.1 value notation	{1 3 6 1}	.{6 1}

Table 1: Examples for extended diagnostic notation

6. A New Arc for Concise OIDs

Object identifiers in [X.690] form are remarkably compact. Nevertheless, for some applications (and engineers), they are simply not compact enough, at least when compared to certain alternatives such as very small unsigned integers (see Section 7). The shortest object identifier under the IETF's control is 1.3.6.1 (4 bytes), although an assignment directly under that arc has not happened since 1999 [RFC2506], and no assignments directly under that arc have ever been assigned directly to protocol elements. The shortest IETF-controlled, First-Come, First-Served OID arc is 8 bytes by getting a Private Enterprise Number from IANA, an OID for which is assigned under 1.3.6.1.4.1. To promote object identifier usage in CBOR and to make OIDs as competitive as possible, (the authors / the IETF / ISOC) have secured a very short arc "{ x y z }" that only occupies (1, 2, 3) byte(s).

[[NB: Registration procedures under that arc.]]

The history of OIDs suggests that the human mind tends to excessive taxonomy around them. Unlike assignments in the 1.3.6.1 range, this document suggests that registrants acquire OIDs under this short arc "laterally" rather than hierarchically, in keeping with CBOR's design goal to have concise serializations.

7. Enumerations in CBOR

This section provides a roadmap to using enumerated items in CBOR, including design considerations for choosing between OIDs, integers, and UTF-8 strings.

CBOR does not have an ENUMERATED type like ASN.1 to identify named values in a protocol element with three or more states (Clause 20 and Clause G.2.3 of [X.680]). ASN.1 ENUMERATED turns out to be superfluous because ASN.1 INTEGER values can get named (and have historically been used for finite, multistate variables, such as version numbers), while ASN.1 ENUMERATED types can be defined to be extensible with the ellipsis lexical item. Practically, the named integers are not serialized in the binary encodings anyway; they merely serve as a semantic hints for designers and debuggers.

CBOR expects that protocol designers will use one of the basic major types for multistate variables, assigning semantics to particular values using higher-level schemas. The obvious choices for the basic types are integers (particularly unsigned integers) and UTF-8 strings. However, these major types are not without drawbacks.

Integers are compact for small values, but have a flat namespace so there are mis-assignment and collision risks that can only be mitigated with protocol-specific registries. Arrays of integers are possible, but arrays require more processing logic for equality comparisons, and the JSON conversion is not intuitive when the enumerated value serves as a key in a map.

UTF-8 strings are less compact when the strings are supposed to resemble their semantics, and there are normalization issues if the strings contain characters beyond the ASCII range. UTF-8 strings also comprise a flat namespace like integers unless the higher-level schema employs delimiters, which makes the string even larger. If conciseness is a design goal, other perceived advantages of a string as an identifier are pretty much blown out the moment one has to tack "https://" onto the front.

This section provides a novel alternative in OIDs.

7.1. Factors Favoring OID Enumerations

A protocol designer might choose OIDs or relative OIDs for an enumerated item in view of the following observations:

1. OIDs and relative OIDs are quite compact: a single-arc relative OID encoded according to this specification occupies just two bytes for primary integer values 0-127 (excluding the semantic

tag <<R>>), and three bytes for primary integer values 128-16383. (In contrast, an unsigned integer requires one byte for 0-23, two bytes for 24-255, and three bytes for 256-65535.)

2. OIDs and relative OIDs (with base) are persistent and globally unambiguous.

3. OIDs and relative OIDs have built-in semantics for designers and debuggers. Specifically, the advent of universal OID repositories such as [OID-INFO] makes it easy for a designer or debugger to pull up useful information about the object of interest (Clause 3.5.10 of [X.660]). This useful information (for humans) does not have to bleed into the encoded representation (for machines).

4. OIDs and relative OIDs are always compared for exact equality: no need to deal with case folding, case sensitivity, or other normalization issues. ("Overlong" encodings are PROHIBITED; therefore overlong encodings MUST be treated as coding errors.)

5. OIDs and relative OIDs have a built-in hierarchy, so if implementers want to extend an enumeration without assigning new values "horizontally", they have the option of assigning new values "vertically", possibly with more or less stringent assignment rules.

6. Because OIDs and relative OIDs (with base) are part of the so-called International Object Identifier tree [X.660], any other protocol specification can reuse the enumeration if the designers find it useful.

7. OIDs and relative OIDs have natural JSON representations in the dotted decimal notations prescribed in Section 5. OIDs and relative OIDs can be distinguished from each other by the presence or absence of the leading dot ".". As the resulting JSON string is entirely numeric in the ASCII range, case and normalization are irrelevant to the comparison. (An object identifier also has a semantic string representation in the form of an OID-IRI [X.680], for those who really want that type of thing.)

8. OIDs and relative OIDs are human language-neutral. A protocol designer working in US-English might name an enumerated value "sig" for "signature", but "sig" could also stand for "significand", "signal", or "special interest group". In Swedish and Norwegian, "sig" is a pronoun that means "himself, herself, itself, one, them", etc.--an entirely different meaning.

7.2. Factors Favoring Integer Enumerations

A protocol designer might choose integers for an enumerated item in view of the following observations:

1. The CBOR encoding of unsigned integers 0-23 is the most compact, occupying exactly one byte (excluding any semantic tags).

2. A protocol designer may wish to prohibit extensibility as a matter of course. Integers comprise a single flat namespace: there is no hierarchy.

3. If greater range is desired while sticking to one byte, a protocol designer may double the range of possible values by allowing negative integers. However, enumerating values using negative integers may have unintended side-effects, because some programming environments (e.g., C/C++) make implementation-defined assumptions about the number of bits needed for an enumerated type.

7.3. Factors Favoring UTF-8 String Enumerations

A protocol designer might choose UTF-8 strings for an enumerated item in view of the following observations:

1. A specification can practically limit the content of UTF-8 strings to the ASCII range (or narrower), mitigating some normalization problems.

2. UTF-8 strings are easier to read on-the-wire for humans.

3. UTF-8 strings can contain arbitrary textual identifiers, which can be hierarchical, e.g., URIs.

7.4. OID Enumeration Example

An enumerated item indicates the revision level of a data format. Revision levels are issued by year, such as 2011, 2012, etc. However, in the year 2013, two revisions were issued: the first one and an important update in June that needs to be distinguished. The revision levels are assigned to some OID arc:

"{2 25 6464646464 revs(4)}"

In this arc, the following sub-arcs are assigned:

Sub-Arc
{v2011(1)}
{v2012(2)}
{v2013(3)}
{v2013(3) june(6)}
{v2014(4)}
{v2015(5)}

Table 2: Example Sub-Arcs

In CBOR, the enumeration is encoded as a relative OID. The schema specifies the base OID arc, which is omitted:

```
c7         # tag(7)
   41 03   # .3

c7         # tag(7)
   42 0306 # .3.6
```

Figure 7: Enumerated Items in CBOR

```
.3
.{v2013(3) june(6)}
```

Figure 8: Enumerated Items in CBOR Diagnostic Notation

```
".3"
".3.6"
```

Figure 9: Enumerated Items in JSON (possibility 1)

```
"v2013"
"v2013/june"
```

Figure 10: Enumerated Items in JSON (possibility 2)

8. Tag Factoring and Tag Stacking with OID Arrays and Maps

A common use of object identifiers in ASN.1 is to identify the kind of data in an open type (Clause 3.8.57 of [X.680]), using information object classes [X.681]. CBOR is schema-neutral, and (although not fully discussed in [RFC7049]) semantic tagging was originally intended to identify items in a global, context-free way (i.e., where a specification would not repurpose a tag with different semantics

than its IANA registration). Therefore, using OIDs to identify contextual data in a similar fashion to [X.681] is RECOMMENDED.

8.1. Tag Factoring

<<O>> and <<R>> can tag CBOR arrays and maps. The idea is that the tag is factored out from each individual byte string; the tag is placed in front of the array or map instead. The tags <<O>> and <<R>> are left-distributive.

When the <<O>> or <<R>> tag is applied to an array, it means that the respective tag is imputed to all items in the array. For example, when the array is tagged with <<O>>, every array item that is a binary string is an OID.

When the <<O>> or <<R>> tag is applied to a map, it means that the respective tag is imputed to all keys in the map. The values in the map are not considered specially tagged.

Array and map stacking is permitted. For example, a 3-dimensional array of OIDs can be composed by using a single <<O>> tag, followed by an array of arrays of arrays of binary strings. All such binary strings are considered OIDs.

8.2. Switching OID and Relative OID

If an individual item in a <<O>> or <<R>> tagged array, or an individual key in a <<O>> or <<R>> tagged map, is tagged with the opposite tag (<<R>> or <<O>>) of the array or map itself, that tag cancels and replaces the outer tag for that item. Like tags MUST NOT be used on such individual items; such tagging is a coding error. For example, if <<R>> is the outer tag on an array and <<O>> is the inner tag on a binary string, semantically the inner item is treated as a regular OID, not as a relative OID.

The purpose is to create more compact and flexible identifier spaces, especially when object identifiers are used as enumerated items. Examples:

<<R>> outside, <<O>> inside: An implementation that strives for a compact representation, does not have to emit base OID arcs repeatedly for each item. At the same time, if a private organization or standards body separate from the specification needs to identify something that the specification maintainers disagree with, the separate body does not need to request registration of an identifier under a controlled arc (i.e., the base arc of the relative OIDs).

<<O>> outside, <<R>> inside: A collection of OIDs is supposed to be open to all-comers, but a certain set of OIDs issued under a particular arc is foreseeable for the majority of implementations. For example, an OID protocol slot may identify cryptographic algorithms: anyone can write (and has written) an algorithm with an arbitrary OID. However, the protocol slot designer may wish to privilege certain algorithms (and therefore OIDs) that are well-known in that field of use.

8.3. Tag Stacking

CBOR permits tag stacking (tagging a tagged item), although this technique has not been used much yet. This specification anticipates that OIDs and relative OIDs will be associated with values with uniform semantics. This section provides specific semantics when tags are "stacked", that is, a CBOR item starts with tag <<O>> or <<R>>, followed by one or more arbitrary tags ("subsequent tags"), followed by a map or array.

8.3.1. Map

The overall gist is that the first tag applies to the keys in a map; the subsequent tags apply to the values in a map.

When <<O>> or <<R>> is the first tag in a stack of tags, followed by a map:

- The <<O>> or <<R>> tag indicates that the keys of the map are byte string OIDs, byte string relative OIDs, or tag-factored arrays or maps of the same.

- The subsequent tags uniformly apply to all of the values.

For example, if tag 32 (URL) is the subsequent tag, then all values in the map are treated semantically as if tag 32 is applied to them individually. See Figure 11.

It is possible that individual values can be tagged. Semantically, these tags cumulate with the outer subsequent tags; inner value tags do not cancel or replace the outer tags.

8.3.2. Array

The overall gist is that the first tag applies to the ordered "keys" in the array (even-numbered items, assuming that the index starts at 0); the subsequent tags apply to the ordered "values" in the array (odd-numbered items). This tagging technique creates an ordered

associative array. [[NB: Some call this the FORTRAN approach. need to cite]]

When <<O>> or <<R>> is the first tag in a stack of tags, followed by an array:

- The <<O>> or <<R>> tag indicates that alternating items, starting with the first item, are byte string OIDs, byte string relative OIDs, or tag-factored arrays or maps of the same.

- The subsequent tags uniformly apply to the alternating items, starting with the second item.

- The array MUST have an even number of items; an array that has an odd number of items is a coding error.

To create an ordered associative array wherein the values (even elements) are arbitrarily tagged, stack tag 55799, self-describe CBOR (Section 2.4.5 of [RFC7049]), after the <<O>> or <<R>> tag. Tag 55799 imparts no special semantics, so it is an effective placeholder. (This sequence is mainly provided for completeness: it is a more compact alternative to an array of duple-arrays that each contain an OID or relative OID, and an arbitrary value.)

8.4. Diagnostic Notation for OID Arrays and Maps

There are no syntactic changes to diagnostic notation beyond Section 5. Using <<O>> or <<R>> with arrays and maps, however, leads to some sublime results.

When an array or map is tagged, that item is embraced with the usual tag format: "<<O>>(<item>)" or "<<R>>(<item>)". This syntax indicates the presence of the tag on the outer item. Inner items in the array or keys in the map are noted in Section 5 form, but are not individually tagged on-the-wire when the tag is the same as the outer tag, because like-tagging is a coding error.

An array or map that involves a stack of tags is notated the usual way. For example, the CBOR diagnostic notation of a map of OIDs to URIs is:

```
6(32({0.9.2342.7776.1: "http://example.com/",
      0.9.2342.7776.2: "ftp://ftp.example.com/pub/"}))
```

Figure 11: Map of OIDs to URIs, in CBOR Diagnostic Diagnostic Notation

9. Applications and Examples of OIDs

9.1. GPU Farm

Consider a 3-dimensional OID array, indicating certain operations to perform on a matrix of values in a GPU farm. Default operations are under the OID arc 0.9.2342.7777 (such as .1, .2, .124, etc.); the arc 0.9.2342.7777 itself represents the identity operation. Certain cryptographic operations like SHA-256 hashing (2.16.840.1.101.3.4.2.1) are also permitted. The resulting notation would be:

```
7([[[.1,   .2,   .3],
    [.1,   .2,   .3],
    [.1,   .2,   .3]],
   [[.124, .125, .126],
    [.95,  .96,  .97 ],
    [.11,  .12,  .13 ]],
   [[h'',  .6,   .4.2],
    [.6,   h'',  .4.2],
    [.6,   2.16.840.1.101.3.4.2.1, h'']]])
```

Figure 12: GPU Farm Matrix Operations, in CBOR Diagnostic Notation

```
c7                                  # tag(7)
   83                               # array(3)
      83                            # array(3)
         83                         # array(3)
            41 01                   # .1 (2)
            41 02                   # .2 (2)
            41 03                   # .3 (2)
         83                         # array(3)
            41 01                   # .1 (2)
            41 02                   # .2 (2)
            41 03                   # .3 (2)
         83                         # array(3)
            41 01                   # .1 (2)
            41 02                   # .2 (2)
            41 03                   # .3 (2)
      83                            # array(3)
         83                         # array(3)
            41 7c                   # .124 (2)
            41 7d                   # .125 (2)
            41 7e                   # .126 (2)
         83                         # array(3)
            41 5f                   # .95 (2)
            41 60                   # .96 (2)
            41 61                   # .97 (2)
         83                         # array(3)
            41 0b                   # .11 (2)
            41 0c                   # .12 (2)
            41 0d                   # .13 (2)
      83                            # array(3)
         83                         # array(3)
            40                      # (empty) (1)
            41 06                   # .6 (2)
            42 0402                 # .4.2 (3)
         83                         # array(3)
            41 06                   # .6 (2)
            40                      # (empty) (1)
            42 0402                 # .4.2 (3)
         83                         # array(3)
            41 06                   # .6 (2)
            c6 49 608648016503040201 # 2.16.840.1.101.3.4.2.1 (10)
            40                      # (empty) (1)
```

Figure 13: GPU Farm Matrix Operations, in CBOR (76 bytes)

9.2. X.500 Distinguished Name

Consider the X.500 distinguished name:

Attribute Types	Attribute Values
c (2.5.4.6)	US
l (2.5.4.7)	Los Angeles
s (2.5.4.8)	CA
postalCode (2.5.4.17)	90013
street (2.5.4.9)	532 S Olive St
businessCategory (2.5.4.15)	Public Park
buildingName (0.9.2342.19200300.100.1.48)	Pershing Square

Table 3: Example X.500 Distinguished Name

Table 3 has four RDNs. The country and street RDNs are single-valued. The second and fourth RDNs are multi-valued.

The equivalent representations in CBOR diagnostic notation and CBOR are:

```
6([{ 2.5.4.6: "US" },
   { 2.5.4.7: "Los Angeles", 2.5.4.8: "CA", 2.5.4.17: "90013" },
   { 2.5.4.9: "532 S Olive St" },
   { 2.5.4.15: "Public Park",
     0.9.2342.19200300.100.1.48: "Pershing Square" }])
```

Figure 14: Distinguished Name, in CBOR Diagnostic Notation

```
6([{ h'550406': "US" },
   { h'550407': "Los Angeles", h'550408': "CA", h'550411': "90013" },
   { h'550409': "532 S Olive St" },
   { h'55040f': "Public Park",
     h'0992268993f22c640130': "Pershing Square" }])
```

Figure 15: Distinguished Name, in CBOR Diagnostic Notation (RFC 7049 only)

```
c6                                      # tag(6)
   84                                   # array(4)
      a1                                # map(1)
         43 550406                      # 2.5.4.6 (4)
         62                             # text(2)
            5553                        # "US"
      a3                                # map(3)
         43 550407                      # 2.5.4.7 (4)
         6b                             # text(11)
            4c6f7320416e67656c6573      # "Los Angeles"
         43 550408                      # 2.5.4.8 (4)
         62                             # text(2)
            4341                        # "CA"
         43 550411                      # 2.5.4.17 (4)
         65                             # text(5)
            3930303133                  # "90013"
      a1                                # map(1)
         43 550409                      # 2.5.4.9 (4)
         6e                             # text(14)
            3533322053204f6c697665205374   # "532 S Olive St"
      a2                                # map(2)
         43 55040f                      # 2.5.4.15 (4)
         6b                             # text(11)
            5075626c6963205061726b      # "Public Park"
         4a 0992268993f22c640130    # 0.9.2342.19200300.100.1.48 (11)
         6f                             # text(15)
            5065727368696e6720537175617265 # "Pershing Square"
```

Figure 16: Distinguished Name, in CBOR (108 bytes)

(This example encoding assumes that all attribute values are UTF-8 strings, or can be represented as UTF-8 strings with no loss of information.)

For reference, the [RFC4514] LDAP string encoding of such data would be:

```
buildingName=Pershing Square+businessCategory=Public Park,
street=532 S Olive St,l=Los Angeles+postalCode=90013+st=CA,c=US
```

Figure 17: Distinguished Name, in LDAP String Encoding (121 bytes)

10. Binary Internet Messages and MIME Entities

Section 2.4.4.3 of [RFC7049] assigns tag 36 to "MIME messages (including all headers)" [RFC2045], and prescribes UTF-8 strings, without further elaboration. Actually MIME encircles several

different formats, and is not limited to UTF-8 strings. This section updates tag 36.

10.1. CBOR Byte String and Binary MIME

Tag 36 is to be used with byte strings. When the tagged item is a byte string, any octet can be used in the content. Arbitrary octets are supported by [RFC2045] and can be supported in protocols such as SMTP using BINARYMIME [RFC3030].

A conforming implementation that purports to process tag 36-tagged items, MUST accept byte strings as well as UTF-8 strings. Byte strings, rather than UTF-8 strings, SHOULD be considered the default. (While binary Content-Transfer-Encoding is not particularly common as of this writing, 8-bit encoding is, and it is foreseeable that many 8-bit encoded messages will still have charsets other than UTF-8.)

10.2. Internet Messages, MIME Messages, and MIME Entities

Definitions: "MIME message" is not explicitly defined in [RFC2045], but a careful read suggests that a MIME message is: "either a (complete or "top-level") RFC 822 message being transferred on a network, or a message encapsulated in a body of type "message/rfc822" or "message/partial"," that also contains MIME header fields, namely, MIME-Version field, which MUST be present (Section 4 of [RFC2045]. Other MIME header fields such as Content-Type and Content-Transfer-Encoding are assumed to be their [RFC2045] default values, if not present in the data.

When the contents have a From field (a type of "originator address field") and a Date field (the lone "origination date field") (Section 3.6 of [RFC5322]), the item is concluded to have a Content-Type of message/rfc822 or message/global, as appropriate, except as otherwise specified in this section.

(TBD: Do we need a separate tag for a MIME entity?) (Alternate proposal: When the tagged data does not include a MIME-Version field or other fields required by RFC822 (5322) (e.g., no From field), it is presumed to be a MIME entity, rather than a MIME message. Therefore, it has no top-level content-type: instead it is simply a "MIME entity", consisting of one element, whose Content-Type is the content of the Content-Type header field, if present, or the [RFC2045] default of "text/plain; charset=us-ascii", if absent. Content-Transfer-Encoding SHALL be assumed to be 8bit when the CBOR item is a UTF-8 string, and SHALL be assumed to be binary when the CBOR item is a byte string. (Or should all be considered CTE: binary?) And, when the tagged data has RFC822 required fields but no

MIME-Version, shall we assume it's a MIME entity, or shall we assume it's an Internet message that does not conform to MIME?)

Content that has no headers whatsoever is valid, and implementations that process tag 36 MUST permit this case: in such a case, the data starts with CRLF CRLF, followed by the body. In such a case, the content is assumed to be a MIME entity of Content-Type "text/plain; charset=us-ascii", and not an RFC822 (RFC5322) Internet message. (TBD: Confirm.)

10.3. Netnews, HTTP, and SIP Messages

Other message types that are MIME-related are message/news, message/http, and message/sip.

[RFC5537] specifies that message/news is deprecated (marked as obsolete) and that message/rfc822 SHOULD be used in its place; presumably this also extends to message/global over time. Netnews Article Format [RFC5536] is a strict subset of Internet Message Format; it can be detected by the presence of the six mandatory header fields: Date, From, Message-ID, Newsgroups, Path, and Subject. (Newsgroups and Path fields are specific to Netnews.)

message/http [RFC7230] is the media type for HTTP requests and responses. It can be detected by analyzing the first line of the body, which is an HTTP Start Line (Section 3.1 of [RFC7230]): it does not conform to the syntax of an Internet Message Format header field. The optional parameter "msgtype" can be inferred from the Start Line. Implementers need to be aware that the default character encoding for message/http is ISO-8859-1, not UTF-8. Therefore, implementations SHOULD NOT encode HTTP messages with CBOR UTF-8 strings.

Similarly, message/sip [RFC3261] is the media type of SIP request and response messages. It can be detected by analyzing the first line of the body, which is a SIP start-line (Section 7.1 of [RFC3261]): it does not conform to the syntax of an Internet Message Format header field. The optional parameter can be inferred from the start-line.

10.4. Other Messages

The CBOR binary or UTF-8 string MAY contain other types of messages. An implementation MAY send such a message as a MIME entity with the Content-Type field appropriately set, or alternatively, MAY send the message at the top-level directly. However, if a purported message type is ambiguous with a message/rfc822 (or message/global) message, a receiver SHALL treat the message as message/rfc822 (or message/global). If a purported message type is ambiguous with a MIME entity

(and unambiguously not message/rfc822 or message/global), a receiver SHALL treat the message as a MIME entity.

11. Applications and Examples of Messages and Entities

Tag 36 is the RECOMMENDED way to convey data with MIME-related metadata, including messages (which may or may not actually be MIME-enabled) and MIME entities.

Example 1: A legacy RFC822 message is encoded as a UTF-8 string or byte string with tag 36. The contents have From, To, Date, and Subject header fields, two CRLFs, and a single line "Hello World!", terminated with a CRLF.

Example 2a: A [RFC5280] certificate is encoded as a byte string with tag 36. The contents are comprised of "Content-Type: application/pkix-cert", two CRLFs, and the DER encoding of the certificate. (The "Content-Transfer-Encoding: binary" header is not necessary.)

Example 2b: A [RFC5280] certificate is encoded as a UTF-8 string or byte string with tag 36. The contents are comprised of "Content-Type: application/pkix-cert", a CRLF, "Content-Transfer-Encoding: base64", two CRLFs, and the base64 encoding of the DER encoding of the certificate, conforming to Section 6.8 of [RFC2045]. In particular, base64 lines are limited to 76 characters, separated by CRLF, and the final line is supposed to end with CRLF. Needless to say, this is not nearly as efficient as Example 2a.

12. X.690 Series Tags

[[NB: Carsten probably won't like this. Plan on removing this section. It is mainly provided to contrast with Section 10.]]

It is foreseeable that CBOR applications will need to send and receive ASN.1 data, for example, for legacy or security applications. While a native representation in CBOR is preferred, preserving the data in an ASN.1 encoding may be necessary, for example, to preserve cryptographic verification. A tag <<X>> is allocated for this purpose.

When the tagged item is a byte string, the byte string contents are encoded according to [X.690], i.e., BER, CER, or DER. CBOR implementations are not required to validate conformance of the contained data to [X.690].

When the tagged item is an array with 3 items:

1. The first item SHALL be an OID (with tag <<O>> omitted; it SHALL NOT be a relative OID), indicating the ASN.1 module containing the type of the PDU. [[NB: this is a good example of a non-trivial structure in which an element is well-defined to be an OID, which has a tag. Is the CBOR philosophy to tag the item, or omit the tag on the item, when the item's semantics are already fixed by the outer tag? Similar situations can apply to tag 32 (URI), etc.]]

2. The second item SHALL be a UTF-8 string indicating the ASN.1 value's _type reference name_ (Clause 3.8.88 of [X.680]) conforming to the "typereference" production (Clause 12.2 of [X.680]).

3. The third item SHALL be a byte string, whose contents are encoded per the prior paragraph.

(TBD: Use of tagged UTF-8 string is reserved for ASN.1 textual formats such as XER and ASN.1 value notation? Probably not necessary. Just omit.)

Implementation note: DER-encoded items are always definite-length, so there is very little reason to use CBOR byte string indefinite encoding when encoding such DER-encoded items.

Example: A [RFC5280] certificate can be encoded:

1. as a byte string with tag <<X>>, or

2. as an array with tag <<X>>, with three elements:

   (1) a byte string "h'2B 06 01 05 05 07 00 12'", which is the BER encoding of 1.3.6.1.5.5.7.0.18,

   (2) a UTF-8 string "Certificate", and

   (3) a byte string containing the DER encoding of the certificate.

13. Regular Expression Clarification

(TODO: better specify conformance to actual regular expression standards with tag 35. PCRE and JavaScript/ECMAScript regular expressions are very different; [RFC7049] is not specific enough about this.)

14. Set and Multiset Technique

CBOR has no native type for a set, which is an arbitrary unordered collection of items. The following technique is RECOMMENDED to express set and multiset semantics concisely in native CBOR data.

In computer science, a _set_ is a collection of distinct items; there is no ordering to the items. Thus, implementations can optimize set storage in many ways that are not available with ordered elements in arrays. Sets can be stored in hashtables, bit fields, trees, or other abstract data types.

In computer science, a _multiset_ allows multiple instances of a set's elements. Put another way, each distinct item has a cardinality property indicating the number of these items in the multiset.

To store items in a set or multiset, it is RECOMMENDED to store the CBOR items as keys in a map; the values SHALL all be positive integers (major type 0, value/additional information greater than or equal to 1). In the special case of a set, the values SHALL be the integer 1. This technique has no special tag associated with it. As with arrays that schemas classify as "records" (i.e., arrays with positionally defined elements), schemas are likewise free to classify maps as sets in particular instances.

15. Fruits Basket Example

Consider a basket of fruits. The basket can contain any number of fruits; each fruit of the same species is considered identical. This basket has two apples, four bananas, six pears, and one pineapple:

```
{"\u{1F34E}": 2, "\u{1F34C}": 4,
 "\u{1F350}": 6, "\u{1F34D}": 1}
```

Figure 18: Fruits Basket in CBOR Diagnostic Notation

```
A4                      # map(4)
   64                   # text(4)
      f09f8d8e          # "\u{1F34E}"
   02                   # unsigned(2)
   64                   # text(4)
      f09f8d8c          # "\u{1F34C}"
   04                   # unsigned(4)
   64                   # text(4)
      f09f8d90          # "\u{1F350}"
   06                   # unsigned(6)
   64                   # text(4)
      f09f8d8d          # "\u{1F34D}"
   01                   # unsigned(1)
```

Figure 19: Fruits Basket in CBOR (33 bytes)

[[TODO: Consider a Merkle Tree example: set of sets of sets of sets of things. ???]]

16. IANA Considerations

(This section to be edited by the RFC editor.)

16.1. CBOR Tags

IANA is requested to assign the CBOR tags in Table 4, with the present document as the specification reference.

Tag	Data Item	Semantics
6<<TBD>>	multiple	object identifier (BER encoding)
7<<TBD>>	multiple	relative object identifier (BER encoding)

Table 4: Values for New Tags

16.2. Discussion

(This subsection to be removed by the RFC editor.)

The space for single-byte tags in CBOR (0..23) is severely limited. It is not clear that the benefits of encoding OIDs/relative OIDs with one less byte per instance outweigh the consumption of two values in this code point space.

Procedurally, this space is also reserved for standards action.

An alternative would be to go for the specification required space, e.g. tag number 40 for <<O>> and tag number 41 for <<R>>. As an example this would change Figure 2 into:

```
d8 28                           # tag(40)
   49                           # bytes(9)
      60 86 48 01 65 03 04 02 01 #
```

Figure 20: SHA-256 OID in cbor (using specification required tag)

16.3. Pre-Existing Tags

(TODO: complete.) IANA is requested to modify the registrations for the following CBOR tags:

Tag	Data Item	Semantics
35	<<TBD>>	regular expression <<TBD>>
36	multiple	message or MIME entity

Table 5: Values for Existing Tags

16.4. New Tags

(TODO: complete.)

17. Security Considerations

The security considerations of RFC 7049 apply.

The encodings in Clauses 8.19 and 8.20 of [X.690] are extremely compact and unambiguous, but MUST be followed precisely to avoid security pitfalls. In particular, the requirements set out in Section 2.1 of this document need to be followed; otherwise, an attacker may be able to subvert a checking process by submitting alternative representations that are later taken as the original (or even something else entirely) by another decoder supposed to be protected by the checking process.

OIDs and relative OIDs can always be treated as opaque byte strings. Actually understanding the structure that was used for generating them is not necessary, and, except for checking the structure requirements, it is strongly NOT RECOMMENDED to perform any processing of this kind (e.g., converting into dotted notation and back) unless absolutely necessary. If the OIDs are translated into other representations, the usual security considerations for non-

trivial representation conversions apply; the integers of the sub-identifiers need to be handled as unlimited-range integers (cf. Figure 4).

17.1. Conversions Between BER and Dotted Decimal Notation

[PKILCAKE] uncovers exploit vectors for the illegal values above, as well as for cases in which conversion to or from the dotted decimal notation goes awry. Neither [X.660] nor [X.680] place an upper bound on the range of unsigned integer values for an arc; the integers are arbitrarily valued. An implementation SHOULD NOT attempt to convert each component using a fixed-size accumulator, as an attacker will certainly be able to cause the accumulator to overflow. Compact and efficient techniques for such conversions, such as the double dabble algorithm [DOUBLEDABBLE] are well-known in the art; their application to this field is left as an exercise to the reader.

18. References

18.1. Normative References

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <http://www.rfc-editor.org/info/rfc3261>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[RFC5536] Murchison, K., Ed., Lindsey, C., and D. Kohn, "Netnews Article Format", RFC 5536, DOI 10.17487/RFC5536, November 2009, <http://www.rfc-editor.org/info/rfc5536>.

[RFC5537] Allbery, R., Ed. and C. Lindsey, "Netnews Architecture and Protocols", RFC 5537, DOI 10.17487/RFC5537, November 2009, <http://www.rfc-editor.org/info/rfc5537>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[X.660] International Telecommunications Union, "Information technology -- Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree", ITU-T Recommendation X.660, July 2011.

[X.680] International Telecommunications Union, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, August 2015.

[X.690] International Telecommunications Union, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015.

18.2. Informative References

[DOUBLEDABBLE]
Gao, S., Al-Khalili, D., and N. Chabini, "An improved BCD adder using 6-LUT FPGAs", IEEE 10th International New Circuits and Systems Conference (NEWCAS 2012), pp. 13-16, DOI: 10.1109/NEWCAS.2012.6328944, June 2012.

[OID-INFO]
Orange SA, "OID Repository", 2016, <http://www.oid-info.com/>.

[PKILCAKE]
Kaminsky, D., Patterson, M., and L. Sassaman, "PKI Layer Cake: New Collision Attacks Against the Global X.509 Infrastructure", FC 2010, Lecture Notes in Computer Science 6052 289-303, DOI: 10.1007/978-3-642-14577-3_22, January 2010, <http://dl.acm.org/citation.cfm?id=2163593>.

[RFC2506] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, DOI 10.17487/RFC2506, March 1999, <http://www.rfc-editor.org/info/rfc2506>.

[RFC3030] Vaudreuil, G., "SMTP Service Extensions for Transmission of Large and Binary MIME Messages", RFC 3030, DOI 10.17487/RFC3030, December 2000, <http://www.rfc-editor.org/info/rfc3030>.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, DOI 10.17487/RFC4514, June 2006, <http://www.rfc-editor.org/info/rfc4514>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, DOI 10.17487/RFC6256, May 2011, <http://www.rfc-editor.org/info/rfc6256>.

[RFC7388] Schoenwaelder, J., Sehgal, A., Tsou, T., and C. Zhou, "Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7388, DOI 10.17487/RFC7388, October 2014, <http://www.rfc-editor.org/info/rfc7388>.

[X.672] International Telecommunications Union, "Information technology -- Open systems interconnection -- Object identifier resolution system", ITU-T Recommendation X.672, August 2010.

[X.681] International Telecommunications Union, "Information technology -- Abstract Syntax Notation One (ASN.1): Information object specification", ITU-T Recommendation X.681, August 2015.

Appendix A. Changes from -03 to -04

Changes occurred based on limited feedback, mainly centered around the abstract and introduction, rather than substantive technical changes. These changes include:

o Changed the title so that it is about tags and techniques.

- Rewrote the abstract to describe the content more accurately, and to point out that no changes to the wire protocol are being proposed.

- Removed "ASN.1" from "object identifiers", as OIDs are independent of ASN.1.

- Rewrote the introduction to be more about the present text.

- Proposed a concise OID arc.

- Provided binary regular expression forms for OID validation.

- Updated IANA registration tables.

Appendix B. Changes from -02 to -03

Many significant changes occurred in this version. These changes include:

- Expanded the draft scope to be a comprehensive CBOR update.

- Added OID-related sections: OID Enumerations, OID Maps and Arrays, and Applications and Examples of OIDs.

- Added Tag 36 update (binary MIME, better definitions).

- Added stub/experimental sections for X.690 Series Tags (tag <<X>>) and Regular Expressions (tag 35).

- Added technique for representing sets and multisets.

- Added references and fixed typos.

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Sean Leonard
Penango, Inc.
5900 Wilshire Boulevard
21st Floor
Los Angeles, CA 90036
USA

Email: dev+ietf@seantek.com
URI: http://www.penango.com/