

# YANG Type Pattern Proposals for P802.1ASdn/D2.2

3 Author: Johannes Specht  
4 Affiliation: Self  
5 Date: July 16, 2024

## 6 About this Document

7 This document is an individual contribution in support of the comment resolution of P802.1ASdn/D2.2. It  
8 contains proposals for replacing the YANG patterns of YANG data type definitions scaled-ns, uscaled-ns  
9 and float64, and adjusting the associated YANG descriptions and references accordingly.

10 The proposals goe back to the rogue comment on the right column on page 5 of  
11 <https://iee802.org/1/files/private/asdn-drafts/d2/802-1ASdn-d2-0-dis-v01.pdf>, which was rejected due  
12 to its unspecific nature.

13 This document is very specific in the sense that it provides YANG code that can be used as copy&paste  
14 replacements. Additional notes and explanations are provided for each replacement, providing additional  
15 background information and remarks to the ballot resolution group.

## 16 Contents

17	About this Document.....	1
18	Specific Proposals.....	2
19	scaled-ns Type Definition.....	2
20	Current YANG Code.....	2
21	Proposed new YANG Code.....	2
22	Notes and Explanations .....	2
23	uscaled-ns Type Definition.....	3
24	Current YANG Code.....	3
25	Proposed new YANG Code.....	3
26	Notes and Explanations .....	3
27	float64 Type Definition .....	4
28	Current YANG Code.....	4
29	Proposed new YANG Code.....	4
30	Notes and Explanations .....	5

31

## 32 Specific Proposals

### 33 scaled-ns Type Definition

#### 34 Current YANG Code

```
35 typedef scaled-ns {
36     type string {
37         pattern "[0-9A-F]{2}(-[0-9A-F]{2}){11}";
38     }
39     description
40         "The IEEE Std 802.1AS ScaledNs type represents signed values of
41         time and time interval in units of 2^16 ns, as a signed 96-bit
42         integer. Each of the 12 octets is represented as a pair of
43         hexadecimal characters, using uppercase for a letter. Octets are
44         separated by a dash character. The most significant octet is first.";
45     reference
46         "6.4.3.1 of IEEE Std 802.1AS";
47 }
```

#### 48 Proposed new YANG Code

```
49 typedef scaled-ns {
50     type string {
51         pattern "0x[0-9A-F]{4}([0-9A-F]{4}){5}";
52     }
53     description
54         "The IEEE Std 802.1AS ScaledNs type represents signed values of
55         time and time interval in units of 2^16 ns, as a signed 96-bit
56         integer. The canonical and lexical representations are as
57         specified in 6.4.3.1 of IEEE Std 802.1AS (i.e., five upper case
58         hexadecimal words with 4 digits each and the words separated by
59         single whitespace characters.";
60     reference
61         "6.4.3.1 of IEEE Std 802.1AS";
62 }
```

## 63 Notes and Explanations

64 The original proposal from the rogue comment described two different patterns:

- 65 - A first pattern aligned with the notation illustrated in 6.4.3.1 of IEEE Std 802.1AS
- 66 - A second pattern aligned with the notation of Integers in YANG (9.2.1. of RFC 7950).

67 The proposed new YANG code limits on the former pattern as a result of discussion with 802.1AS experts.

## 68 uscaled-ns Type Definition

### 69 Current YANG Code

```
70 typedef uscaled-ns {
71     type string {
72         pattern "[0-9A-F]{2}(-[0-9A-F]{2}){11}";
73     }
74     description
75         "The IEEE Std 802.1AS UScaledNs type represents unsigned values of
76         time and time interval in units of 2^16 ns, as an unsigned 96-bit
77         integer. Each of the 12 octets is represented as a pair of
78         hexadecimal characters, using uppercase for a letter. Octets are
79         separated by a dash character. The most significant octet is first";
80     reference
81         "6.4.3.2 of IEEE Std 802.1AS";
82 }
```

### 83 Proposed new YANG Code

```
84 typedef uscaled-ns {
85     type string {
86         pattern " 0x[0-9A-F]{4}([ 0-9A-F]{4}){5}";
87     }
88     description
89         "The IEEE Std 802.1AS UScaledNs type represents unsigned values of
90         time and time interval in units of 2^16 ns, as an unsigned 96-bit
91         integer. The canonical and lexical representations are as
92         specified in 6.4.3.2 of IEEE Std 802.1AS (i.e., five upper case
93         hexadecimal words with 4 digits each and the words separated by
94         single whitespace characters.";
95     reference
96         "6.4.3.2 of IEEE Std 802.1AS";
97 }
```

### 98 Notes and Explanations

99 The notes and explanations for scaled-ns apply equally for uscaled-ns.

100

## 101 float64 Type Definition

### 102 Current YANG Code

```
103 typedef float64 {
104     type string {
105         pattern "[0-9A-F]{2}(-[0-9A-F]{2}){7}";
106     }
107     description
108         "The IEEE Std 802.1AS Float64 type represents IEEE Std 754 binary64. Each of the 8 octets is
109         represented as a pair of hexadecimal characters, using uppercase for a letter. Octets are
110         separated by a dash character. The most significant octet is first.";
111     reference
112         "6.4.2 of IEEE Std 802.1AS";
113 }
```

### 114 Proposed new YANG Code

```
115 typedef float64 {
116     type string {
117         pattern "([+-]?0[Xx]([0-9a-fA-F]*.[0-9a-fA-F]+|[0-9a-fA-F]+|[0-9a-fA-F]+)[Pp][+-]?[0-9]+)"
118             "|([+-]?([0-9]*.[0-9]+|[0-9]+|[0-9]+)[Ee][+-]?[0-9]+)";
119     }
120     description
121         "The IEEE Std 802.1AS Float64 type represents IEEE Std 754 binary64. The lexical
122         representation is either that of external hexadecimal-significand character sequences
123         representing finite numbers as specified in 5.12.3 of IEEE Std 754-2019, or that of
124         ISO/IEC 9899:1999 (C99) for decimal floating-point numbers with exponent, without
125         floating-suffix and limited to finite representable numbers (e.g., Inf. and NaN excluded).
126
127         Canonical form:
128         a) The canonical form of a positive number does not include the sign '+', and does not
129            include the sign '+' for positive exponents.
130         b) The hexadecimal/decimal point is required.
131         c) Lexically representable numbers that cannot be represented by binary64 shall be rounded
132            by a server to the closest finite number representable by binary64.
133         d) When a server sends XML-encoded data, only normalized values and are sent in the format
134            according to 5.12.3 of IEEE Std 754-2019 with at least one fractional digit and one
135            exponent digit (i.e., pattern '-?0[Xx]1.[0-9a-fA-F]+[pP]-?[0-9]+').
136         e) XPath expression evaluations are done using the canonical form specified in items a)
137            through d).";
138     reference
139         "6.4.2 of IEEE Std 802.1AS
```

140 5.12.3 of IEEE Std 754-2019  
141 6.4.4.2 of ISO/IEC 9899:1999";  
142 }

### 143 Notes and Explanations

144 The current YANG code is effectively a byte-wise memory dump of binary64 values.

145 This proposed new YANG code is intended to address several issues with this. This proposal is intended to  
146 address various usability concerns of memory-dumps that were raised in other contexts in IEEE 802.1, and  
147 likewise addresses potential issues with non-representable/non-finite numbers (e.g., NaN).

148 It is **to be discussed** whether a hexadecimal representation would be sufficient (i.e., omitting decimal  
149 entirely, which means all **red text** would disappear).

150 The proposed new YANG code is more aligned with a human readable form. A similar approach is found  
151 in "ietf-routing-types.yang" (<https://www.netconfcentral.org/modules/ietf-routing-types/2017-12-04>)  
152 for data type "bandwidth-ieee-float64". "bandwidth-ieee-float64" limits to hexadecimal, non-negative,  
153 normalized, non-fraction numbers. These limits are not implemented in "float64". Limiting to  
154 representable numbers is present in both, "float64" and "bandwidth-ieee-float64".

155 The proposed pattern does not account for potential values that (even) exceed the range of binary64.  
156 However, this should be covered by the definition of the canonical form. A simple (but naive) way for  
157 covering this in the pattern would be by limiting the number of digits. However, this would only narrow the  
158 range limits. Accurate range limitation by pattern would be possible (in theory) by excessive combinatorial  
159 expansion of digit-combinations. This was omitted in favor of readability, and because the range limits  
160 should be covered, as said.

161 Notes on the canonical form:

- 162 - The canonical form requirements a) and b) are derived from 9.3.2. of IETF RFC 7950. Note that  
163 RFC 7950 does not limit to either upper- or lower- case letters for hexadecimal integer values  
164 (9.2.1 and 9.2.2 of RFC 7950).
- 165 - Item c) covers various cases for config data.
- 166 - Items d) and e) are derived from 9.1. of RFC 7950, though the normalization is a logical  
167 consequence to ease XPath evaluation.