draft-fedyk-netmod-yang-normal-form-01

Datatracker Link

IETF 127

Don Fedyk

dfedyk@labn.net

Scott Mansfield

scott.mansfield@ericsson.com

Introduction

- For string-derived types, YANG currently provides no mechanism to define a normalized form distinct from the lexical representation.
- As a result, semantically equivalent values that have multiple valid lexical representations may not compare equal and may not be detected as duplicates.

- This presentation provides the changes to the draft since -00

Changes since IETF 126

- Clarifications regarding the normalization procedure.
- Identifies the areas in NETCONF and specifically notes
  - A leafref whose target type has a normalized form MUST use that normalized form when determining whether its value identifies an existing target instance.
- Specific details for an extension to define "mac-48" normalized form.
- Extensive updates to the semantics section.
- A design rationale is provided.
- History of the MAC format issue is provided in an Appendix.

Request

- Authors would like to request working group adoption
- This is a low impact, high value mechanism that provides a generic solution for a long-standing issue with strings in YANG.

Backup Slides (IETF 126 Information)

Introduction

- YANG has an issue with strings when strings are used as keys or simply when comparing strings.
- If a string has a pattern, it is possible to create a situation when equivalent strings are not considered equivalent.
- The example in this presentation is mac-address.
- Recommendation
  - While we are waiting for a fix that may never come it is possible to express our intent in current YANG with a small extension.

MAC Address Format (recap)

- IETF and IEEE 802.1 have different patterns for mac-address
  - IETF Format: pattern '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}';
    - uses ':' as separator
  - IEEE 802.1 Format: pattern "[0-9a-fA-F]{2}(-[0-9a-fA-F]{2}){5}";
    - uses '-' as separator
    - Also ':' has a different interpretation in IEEE 802 specs than '-' does. The ':' indicates bit-reversal of each hex digit.
      - The bit-reversal usage is historic and common usage is that '-' and ':' are treated the same.
- However, today (in 2026)
  - AA-BB-CC-DD-EE-FF should be equivlant to aa:bb:cc:dd:ee:ff

Not just a ‘-’ or ‘:’ problem (recap)

- IEEE definition
- ieee802-types.yang
- “-”
- Pattern allows uppercase and lowercase but description says uppercase is used.

```
typedef mac-address {
  type string {
    pattern "[0-9a-fA-F]{2}(-[0-9a-fA-F]{2}){5}";
  }
  description
    "The mac-address type represents a MAC address in the canonical format
    and hexadecimal format specified by IEEE Std 802. The hexadecimal
    representation uses uppercase characters.";
  reference
    "3.1, 8.1 of IEEE Std 802";
}
```

- IETF definition
- ietf-yang-types.yang
- “:”
- Pattern allows uppercase and lowercase but notes that lowercase is canonical.

```
typedef mac-address {
  type string {
    pattern '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}';
  }
  description
    "The mac-address type represents a 48-bit IEEE 802 Media
     Access Control (MAC) address.  The canonical representation
     uses lowercase characters.  Note that there are IEEE 802 MAC
     addresses with a different length that this type cannot
     represent.  The phys-address type may be used to represent
     physical addresses of varying length.

     In the value set and its semantics, this type is equivalent
     to the MacAddress textual convention of the SMIv2.";
  reference
    "IEEE 802: IEEE Standard for Local and Metropolitan Area
               Networks: Overview and Architecture
     RFC 2579: Textual Conventions for SMIv2";
}
```

Rules for a solution

- MUST preserve the lexical representation of values.
- MUST provide deterministic comparison semantics.
- MUST be minimally invasive to existing YANG modules.
- MUST be opt-in and backward compatible.
- MUST be general across string-derived types.
- SHOULD be simple to implement.
- SHOULD avoid complex transformation languages.

A small extension with identities

- YANG allows extensions these extension extend the language.
- It documents the behavior, but it does not implement a solution.
- The argument attribute is a string.
- We can use an identity to define the string.
- Implementors can read the associated string
- Additional identities can be added.

```
extension canonical-form {
   argument form;
   description
     "Specifies a deterministic canonicalization form
      used to derive the canonical form of a value.";
 }

 identity canonical-form {
   description
     "Base identity for canonical forms.";
 }
 identity mac-48 {
   base canonical-form;
   description
     "Canonical form for 48-bit MAC addresses.";
 }
```

Usage One simple line

- This extension is usable with existing YANG tooling. Tools such as pyang can parse the extension and make its argument available as a string.
- Implementation Focused
  - Validation that the string identifies a supported canonical form, and enforcement of the associated comparison semantics, are performed by implementations that support this extension. Pyang does not enforce “mac-48” but implementation code can.
- Allows for other canonical formats mac-64, ip-32, ip-128 etc.
- Works with all MAC address definitions.
- Does need YANG support for seamless processing, of keys, lists and comparison.
- We could ask for YANG Next to enable checking identities in extensions.

```
leaf address {
    type ieee:mac-address;
    yang:canonical-form "yang:mac-48";
    mandatory true;
    description
      "A sample IEEE MAC address.";
  }
```

Implementation for IETF – not part of this

- The canonicalization procedure which would have to be implemented in YANG is:
  1. Validate the input against the lexical pattern.
  2. Remove all separators.
  3. Convert hexadecimal digits to uppercase.
  4. Use the resulting 12 hexadecimal digits as the canonical form of the MAC address.
- Equivalent inputs include:
  - aa:bb:cc:dd:ee:ff
  - AA:BB:CC:DD:EE:FF
  - aa-bb-cc-dd-ee-ff
  - AA-BB-CC-DD-EE-FF

  These values yield the same canonical form:

  - 0xAABBCCDDEEFF

A step to a complete solution

- From a YANG syntax perspective, the solution is complete.
- However, YANG processing would need to be modified to read the extension behavior for :
  - List keys uniqueness
  - Leaf-list uniqueness
  - MAC address Comparison
  - X-path comparisons

  Questions? Comments?