JSON Encoding of Data Modeled with YANG

`draft-ietf-netmod-yang-json-01`

Ladislav Lhotka
⟨lhotka@nic.cz⟩

13 November 2014

Major Changes

- Metadata encoding moved to a separate draft: *draft-lhotka-netmod-yang-metadata-00*
- JSON encoding is now defined directly rather than via XML-JSON mapping.
- Rules for namespace encoding have changed.
- I-JSON compliance.

Namespace Encoding

Namespace encoding is as before: *module_name* : *node_name*.

Rules for its placement have changed: Namespace ID must be used

1. for all root data nodes,
2. whenever parent node’s namespace is different,
3. nowhere else.

Example:

```
"ietf-interfaces:interfaces": {
  "interface": {
    "name": "eth0",
    …
    "ietf-ip:ipv4": {
      "ip": "198.51.100.1",
      …
    }
  }
}
```

Instance Identifiers

Currently, all node names in an instance-identifier value have to be qualified with namespace ID (module name).

```
/ietf-interfaces:interfaces/ietf-interfaces:interface[
   ietf-interfaces:name='eth0']
```

Proposal: Use analogical rules as for instance encoding, i.e. namespace ID is used if (and only if) it differs from the parent's.

```
/ietf-interfaces:interfaces/interface[
   name='eth0']/ietf-ip:ipv4/ip
```

I-JSON

draft-ietf-json-i-json-02:
“I-JSON is a restricted profile of JSON designed to maximize interoperability and increase confidence that software can process it successfully with predictable results.”

I-JSON Compliance Issues:

- permitted characters,
- 64-bit numbers,
- values of *binary* type.

Character Set

I-JSON: “Object member names, and string values in arrays and object members, MUST NOT include code points which identify Surrogates or Noncharacters.”

Due to XML legacy, YANG *string* and *enumeration* types permit some Unicode noncharacters:

- block `U+FFD0..U+FDEF` in Basic Multilingual Plane
- last two codepoints in each of 16 supplementary planes, e.g. `U+1FFFE` and `U+1FFFE`.

Noncharacters are reserved for internal (private) use, and normally not interchanged.

Solution:

Noncharacters are likely to be banned in YANG 1.1, see issue Y56:

`https://svn.tools.ietf.org/svn/wg/netmod/yang-1.1/issues.html#sec-56`

64-bit Numbers

I-JSON: “I-JSON messages SHOULD NOT include numbers which express greater magnitude or precision than an IEEE 754 double precision number provides, for example $1E400$ or $3.141592653589793238462643383279$.

In particular, an I-JSON sender MUST NOT expect a receiver to treat an integer whose absolute value is greater than 9007199254740991 (i.e., that is outside the range $\langle -2^{53}+1, 2^{53}-1 \rangle$) as an exact value.”

Solution:

Values of *int64*, *uint64* and *decimal64* types are encoded as strings.

(Other numeric values are still encoded as JSON numbers.)

Encoding of *binary* Values

I-JSON: “When it is required that an I-JSON protocol element contain arbitrary binary data, it is RECOMMENDED that this data be encoded in a string value in base64url; see Section 5 of [RFC 4868].”

YANG *binary* type prescribes base64, which is perfectly fine – there is no need to have encoded binary values URL-safe.

Solution: Keep using *base64*.

Open Issue #1: *union* type

JSON carries partial type information in the encoding.

```
leaf foo {
  type union {
    type uint8;
    type string;
  }
}
```

application/yang.data+xml	*application/yang.data+json*
`<foo>42</foo>` $\Longrightarrow$ number,	`"foo"` : 42 $\Longrightarrow$ number,
`<foo>42.5</foo>` $\Longrightarrow$ string.	`"foo"` : `"42.5"` $\Longrightarrow$ string,
	`"foo"` : 42.5 $\Longrightarrow$ error.

Is it a problem?

Open Issue #2: *anyxml*

“An anyxml instance is encoded as a name/value pair. The value can be of any valid JSON type, i.e. an object, array, number, string or any of the literals ‘true’, ‘false’ and ‘null’.”

Example: For

```
anyxml foo;
```

this is a valid instance:

```
"foo": [true, null, true]
```

For JSON, *anyxml* means in fact ***anyjson***.

This should be solved in YANG 1.1, see issue Y34:

`https://svn.tools.ietf.org/svn/wg/netmod/yang-1.1/issues.html#sec-34`