NETMOD — Internet-Draft — Intended status: Standards Track — Expires: October 4, 2013

L. Lhotka
CZ.NIC
April 2, 2013

Modeling JSON Text with YANG
draft-lhotka-netmod-yang-json-01

Abstract

This document defines rules for mapping data models expressed in YANG to configuration and operational state data encoded as JSON text. It does so by specifying a procedure for translating the subset of YANG-compatible XML documents to JSON text, and vice versa.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The aim of this document is define rules for mapping data models expressed in the YANG data modeling language [RFC6020] to configuration and operational state data encoded as JavaScript Object Notation (JSON) text [RFC4627]. The result can be potentially applied in two different ways:

1. JSON may be used instead of the standard XML [XML] encoding in the context of the NETCONF protocol [RFC6241] and/or with existing data models expressed in YANG. An example application is the YANG-API Protocol [YANG-API].

2. Other documents that choose JSON to represent structured data can use YANG for defining the data model, i.e., both syntactic and semantic constraints that the data have to satisfy.

JSON mapping rules could be specified in a similar way as the XML mapping rules in [RFC6020]. This would however require solving several problems. To begin with, YANG uses XPath [XPath] quite extensively, but XPath is not defined for JSON and such a definition would be far from straightforward.

In order to avoid these technical difficulties, this document employs an alternative approach: it defines a relatively simple procedure which allows for translating the subset of XML that can be modeled using YANG to JSON, and vice versa. Consequently, validation of a JSON text against a data model can done by translating the JSON text to XML, which is then validated according to the rules stated in [RFC6020].

The translation procedure is adapted to YANG specifics and requirements, namely:

1. The translation is driven by a concrete YANG data model and uses information about data types to achieve better results than generic XML-JSON translation procedures.

2. Various document types are supported, namely configuration data, configuration + state data, RPC input and output parameters, and notifications.

3. XML namespaces specified in the data model are mapped to namespaces of JSON objects. However, explicit namespace identifiers are rarely needed in JSON text.

4. Translation of XML attributes, mixed content, comments and processing instructions is not supported.

2. Terminology and Notation

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

The following terms are defined in [RFC6020]:

o anyxml

o augment

o container

o data node

o data tree

o datatype

o feature

o identity

o instance identifier

o leaf

o leaf-list

o list

o module

o submodule

The following terms are defined in [XMLNS]:

o local name

o prefixed name

o qualified name

3. Specification of the Translation Procedure

The translation procedure defines a 1-1 correspondence between the subset of YANG-compatible XML documents and JSON text. This means that the translation can be applied in both directions and is always invertible.

Any YANG-compatible XML document can be translated, except documents with mixed content. This is only a minor limitation since mixed content is marginal in YANG - it is allowed only in "anyxml" nodes.

An implementation of the translation procedure MAY translate "anyxml" nodes and their contents from XML to JSON or vice versa, but the specific details of this translation are outside the scope of this document. Note that the contents of "anyxml" nodes are not relevant for validity in terms of a YANG data model.

The following subsections specify rules mainly for translating XML documents to JSON text. Rules for the inverse translation are stated only where necessary, otherwise they can be easily inferred.

REQUIRED parameters of the translation procedure are:

o YANG data model,

o type of the input XML document,

o optional features (defined via the "feature" statement) that are considered active.

The permissible types of XML documents are listed in Table 1 together with the corresponding part of the data model that is used for the translation.

Document Type	Data Model Section
configuration and state data	main data tree
configuration	main data tree ("config true")
RPC input parameters	"input" nodes under "rpc"
RPC output parameters	"output" nodes under "rpc"
notification	"notification" nodes

Table 1: YANG Document Types

A particular application may decide to use only a subset of document types from Table 1. For instance, YANG-API Protocol [YANG-API] does not use notifications.

XML documents can be translated to JSON text only if they are valid instances of the YANG data model and selected document type, also taking into account the active features, if there are any.

3.1. Names and Namespaces

The local part of a JSON name is always identical to the local name of the corresponding XML element.

Each JSON name lives in a namespace which is uniquely identified by the name of the YANG module where the corresponding data node is defined. If the data node is defined in a submodule, then the namespace identifier is the name of the main module to which the submodule belongs. The translation procedure MUST correctly map YANG namespace URIs to YANG module names and vice versa.

The namespace SHALL be expressed in JSON text by prefixing the local name in the following way:

```
        <module name>:<local name>
```

Figure 1: Encoding a namespace identifier with a local name.

The namespace identifier MUST be used for local names that are ambiguous, i.e., whenever the data model permits a sibling node with the same local name. Otherwise, the namespace identifier is OPTIONAL.

When mapping namespaces from JSON text to XML, the resulting XML document may use default namespace declarations (via the "xmlns" attribute), prefix-based namespace declarations (via attributes beginning with "xmlns:"), or any combination thereof, following the rules stated in [XMLNS]. If prefixed names are used, their prefix SHOULD be the one defined by the "prefix" statement in the YANG module where each data node is defined.

3.2. Mapping XML Elements to JSON Objects

XML elements are translated to JSON objects in a straightforward way:

o An XML element that is modeled as YANG leaf is translated to a name/value pair and the JSON datatype of the value is derived from the YANG datatype of the leaf (see Section 3.3 for the datatype mapping rules).

o An XML element that is modeled as YANG container is translated to a JSON object.

o A sequence of one or more sibling XML elements with the same qualified name that is modeled as YANG leaf-list is translated to a name/array pair, and the array elements are primitive values whose type depends on the datatype of the leaf-list (see Section 3.3).

o A sequence of one or more sibling XML elements with the same qualified name that is modeled as YANG list is translated to a name/array pair, and the array elements are JSON objects. Unlike the XML encoding, which requires the list keys to come first and in the order specified by the data model, the order of members within a list entry is arbitrary, because JSON objects are fundamentally unordered collections of members.

Note that the same XML element may be translated in different ways, depending on the definition of the corresponding data node in YANG. For example,

```
    <foo>42</foo>
```

is translated to

```
    "foo": 42
```

if the "foo" node is defined as a leaf with the "uint8" datatype, or to

```
    "foo": ["42"]
```

if the "foo" node is defined as a leaf-list with the "string" datatype.

3.3. Mapping YANG Datatypes to JSON Values

3.3.1. Numeric Types

A value of one of the YANG numeric types ("int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64" and "decimal64") is mapped to a JSON number using the same lexical representation.

3.3.2. The "string" Type

A "string" value is mapped to an identical JSON string, subject to JSON encoding rules.

3.3.3. The "boolean" Type

A "boolean" value is mapped to the corresponding JSON value 'true' or 'false'.

3.3.4. The "enumeration" Type

An "enumeration" value is mapped in the same way as a string except that the permitted values are defined by "enum" statements in YANG.

3.3.5. The "bits" Type

A "bits" value is mapped to a string identical to the lexical representation of this value in XML, i.e., space-separated names representing the individual bit values that are set.

3.3.6. The "binary" Type

A "binary" value is mapped to a JSON string identical to the lexical representation of this value in XML, i.e., base64-encoded binary data.

3.3.7. The "leafref" Type

A "leafref" value is mapped according to the same rules as the type of the leaf being referred to.

3.3.8. The "identityref" Type

An "identityref" value is mapped to a string representing the qualified name of the identity. Its namespace MAY be expressed as shown in Figure 1. If the namespace part is not present, the

namespace of the name of the JSON object containing the value is assumed.

3.3.9. The "empty" Type

An "empty" value is mapped to '[null]', i.e., an array with the 'null' value being its only element.

This representation was chosen instead of using simply 'null' in order to facilitate the use of empty leafs in common programming languages. When used in a boolean context, the '[null]' value, unlike 'null', evaluates to 'true'.

3.3.10. The "union" Type

YANG "union" type represents a choice among multiple alternative types. The actual type of the XML value MUST be determined using the procedure specified in Sec. 9.12 of [RFC6020] and the mapping rules for that type are used.

For example, consider the following YANG definition:

```
leaf-list bar {
    type union {
        type uint16;
        type string;
    }
}
```

The sequence of three XML elements

```
<bar>6378</bar>
<bar>14.5</bar>
<bar>infinity</bar>
```

will then be translated to this name/array pair:

```
"bar": [6378, "14.5", "infinity"]
```

3.3.11. The "instance-identifier" Type

An "instance-identifier" value is a string representing a simplified XPath specification. It is mapped to an analogical JSON string in which all occurrences of XML namespace prefixes are either removed or replaced with the corresponding module name according to the rules of Section 3.1.

When translating such a value from JSON to XML, all components of the

instance-identifier MUST be given appropriate XML namespace prefixes. It is RECOMMENDED that these prefixes be those defined via the "prefix" statement in the corresponding YANG modules.

3.4. Example

Consider a simple data model defined by the following YANG module:

```
   module ex-json {

     namespace "http://example.com/ex-json";

     prefix "ej";

     import ietf-inet-types {
       prefix "inet";
     }

     container top {
       list address {
         key "seqno";
         leaf seqno {
           type uint8;
         }
         leaf ip {
           type inet:ip-address;
           mandatory "true";
         }
       }
       container phases {
         typedef angle {
           type decimal64 {
             fraction-digits "2";
           }
           units "radians";
         }
         leaf max-phase {
           default "6.28";
           type angle;
         }
         leaf-list phase {
           type angle;
           must ". <= ../max-phase";
           min-elements "1";
         }
       }
     }
   }
```

Figure 2: Example YANG module.

By using the translation procedure defined in this document, we can conclude that the following JSON text is valid according to the data model:

```
{
  "top": {
    "address": [
      {
        "seqno": 1,
        "ip": "192.0.2.1"
      },
      {
        "seqno": 2,
        "ip": "2001:db8:0:1::1"
      }
    ],
    "phases": {
      "phase": [
        0.79,
        1.04,
        3.14
      ]
    }
  }
}
```

Figure 3: Example JSON text.

3.5. IANA Considerations

TBD.

3.6. Security Considerations

TBD.

3.7. Acknowledgments

The author wishes to thank Andy Bierman, Martin Bjorklund and Phil Shafer for their helpful comments and suggestions.

4. References

4.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, September 2010.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "NETCONF Configuration Protocol", RFC 6241, June 2011.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2006/REC-xml-20060816>.

[XMLNS] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

4.2. Informative References

[XPath] Clark, J., "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

[YANG-API] Bierman, A. and M. Bjorklund, "YANG-API Protocol", draft-bierman-netconf-yang-api-01 (work in progress), November 2012.

Author's Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz