Defining and Using Metadata with YANG

`draft-lhotka-netmod-yang-metadata-00`

Ladislav Lhotka
⟨lhotka@nic.cz⟩

13 November 2014

Introduction

Use cases for metadata annotations:

- deactivating a subtree in a datastore while keeping the data in place;
- supplementing data model info with instance-specific data;
- RPC operations (example: `<edit-config>` in NETCONF).

XML attributes are primarily intended for metadata but JSON doesn't provide any canonical means for encoding metadata.

YANG doesn't model XML attributes (by design) but the client needs to be able to learn what metadata types the server supports.

The set of metadata annotations should be extensible in a distributed way $\Rightarrow$ namespaces.

Goals

1. Define the annotation (semantics, datatype).
2. Allow servers to advertise support for a particular annotation.
3. XML encoding: assign a namespace URI (+ recommended prefix).
4. Define a suitable JSON encoding – it must not change the encoding rules for YANG node instances.
5. JSON encoding: assign a YANG module name as the namespace ID.

Solution

Module *ietf-yang-metadata*: YANG extension for defining annotations.

```
extension annotation {
  argument name;
  description
    "...";
}
```

Substatements:

- description,
- reference,
- status,
- type – default: *string*,
- units.

Example:

```
module ex-inactive {
  namespace
    "http://example.org/ex-inactive";
  prefix "ein";
  import ietf-yang-metadata {
    prefix "md";
  }
  md:annotation inactive {
    type boolean;
    description
      "If this annotation is attached
       to a configuration data node,
       and its value is 'true', then
       the server MUST behave as if
       the data subtree rooted at this
       node was not present.";
  }
}
```

XML Encoding

Standard XML attribute with a namespace.

Example:

```
<foo xmlns:ein="http://example.org/ex-inactive"
     ein:inactive="true">
  ...
</foo>
```

JSON Encoding

Character @ in a member name signals a metadata object that contains all annotations attached to a data node instance or list entry.

For object values, the metadata object becomes a member of that object.

Otherwise, the metadata object is a sibling member of the annotated instance.

Namespace encoding is the same as for data node instances:

module_name : *annotation_name*.

Annotation names must always be prefixed with namespace ID.

Examples

`container cask`

```
"cask": {
  "@": {
    "ex-inactive:inactive": true
  },
  ...
}
```

`leaf flag`

```
"flag": true,
"@flag": {
  "ex-inactive:inactive": true
}
```

Entry of `list seq`

```
"seq": [
  {
    "@": {
      "ex-inactive:inactive": true
    },
    "name": "one",
    ...
  },
  {
    "name": "two",
    ...
  }
]
```

`leaf-list folio`

```
"folio": [6, 3, 7, 8],
"@folio": [
  null,
  {"ex-inactive:inactive": true},
  {"ex-inactive:inactive": true}
]
```

Validation of Annotations

The I-D contains an update to RFC 6110 so as to support the `md:annotation` statement in the mapping to RELAX NG schemas.

Metadata attributes in XML instance documents can then be completely validated.

It is now integrated into DSDL plugin of *pyang*.

Open Issues

1. Would it make sense to include `annotation` as a built-in statement in YANG 1.1 instead of using an extension?
2. Do we need to annotate entire leaf-lists or lists? (There is no straight-forward representation in XML.)
3. Namespace ID could be avoided if the annotation is defined in the same module as the node whose instance is being annotated. Does it make sense?