YANG Open Issues

IETF 73

Martin Björklund
mbj@tail-f.com

Canonical form of data types 1(3)

Some data types accept multiple lexical representations of the same values. For example, the positive integer 42 can be represented as “42” and “+42”, and the ipv6 address "0:0:0:0:0:0:0:1” can be represented as “::1”.

Problem: If the value “42” is used as key in a list entry, can another entry with value “+42” be created? Clearly not.

Solution: Each data type has a *canonical form*, which is used in the conceptual data store.

Issue: How should the canonical form be specified in typedefs? For example, the *ipv6-address* type defined in the YANG module *inet-types*

Canonical form of data types 2(3)

Alternative 1: Specify all lexicographical representations in the pattern, and define the canonical form in text in the description clause.

Alternative 2: Specify all lexicographical representations in the pattern, and define the canonical form in text in a new clause “canonical”:

```
typedef ipv6-address {
    type string {
        pattern
         /* full */
         '((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})'
     +   '(%[\p{N}\p{L}]+)?)'
        /* mixed */
     +  ...
    }
    canonical “The canonical form is the 'full' form”;
}
```

Canonical form of data types 3(3)

Alternative 3: Specify all lexicographical representations in the pattern, and formally define the canonical form in a new clause “canonical”:

```
typedef ipv6-address {
    type string {
        pattern
          ...;
    }
    canonical {
        pattern
         /* full */
         '((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})';
    }
}
```

Alternative 4: Specify the canonical representation only, and let implementations be liberal and accept other forms if they want to.

Keyref vs. leafref

Currently we have keyref which can point to key leafs. A config keyref can point to config keys only. The idea is that a keyref is used to refer to other existing instances.

Several people have suggested a less restricted type, *leafref* which can point to any leaf, not just keys. Such a type would have a statement *require-instance*, which can be true or false. If require-instance is true, a config leafref can point to a config leaf only.

```
type keyref {
    path “/interface/name”;
}
// equivalent to:
type leafref {
    path “/interface/name”;
    require-instance true;
}
```

```
type leafref {
    path “/interface/ifAdminStatus;
}
```

Conformance statement 1(4)

Currently, the model designer has two mechanisms for conformance:

- The basic behavior of the module
- Features, which divides a module into optional subsets

On the ML, it was suggested that a mechanism is needed to define levels of conformance with regards to definitions in a module and/or features. This has been useful in SMI.

There are two use cases:

- change a node to be read only (config false)
  - compare with MIN-ACCESS
- limit the value space that is supported
  - compare with WRITE-SYNTAX

Conformance statement 2(4)

Possible solution: Add a statement to define named conformance levels. A device advertises which conformance level it implements.

```
conformance read-only {
    object-variance “/system” {
        config false;
    }
}

conformance partial {
    object-variance “/system/user” {
        config false;
    }
    object-variance “/system/name-server” {
        config false;
    }
}
```

```
<capability>
  http://example.com/sys?conformance=partial
</capability>
```

Conformance statement 3(4)

In IETF MIBs, WRITE-SYNTAX is mainly used to

- say that createAndWait is not necessary to support
  - does not apply to NETCONF
- specify which values can be written (e.g. notReady can be read but not written)
  - does not apply to NETCONF
- backwards compatibility
  - solved in YANG with revisions

Conformance statement 4(4)

MIN-ACCESS is often used to provide a read-only view of writable data.

- We're trying to standardize configuration data models, so is it really interesting to support such models in <get> only?
- There are occasional exceptions but these can be handled with the mechanisms we have.

Recommendation: Do not add a conformance statement.

Schema discovery

The YANG draft should specify how YANG modules and submodules are discovered through the schema discovery mechanism in draft-ietf-netconf-monitoring.

identifier – YANG module or submodule name

version – the revision string of the module or submodule

format – YANG

namespace – the namespace of the YANG module (that the submodule belongs to)

Also specify that library modules which are not advertised in the hello message may be present in this list.

Inline <rpc-error> in <data>

An inline RPC error is generated if an error occurs during processing of a <get> or <get-config> request for a particular object.

The YANG spec needs to specify how these RPC errors are generated.

```
container foo {
    leaf aa {
        ...
    }
    leaf bb {
        ...
    }
}
```

```
<foo>
  <aa>42</aa>
  <bb>
    <rpc-error>...</rpc-error>
  </bb>
</foo>
```

```
<foo>
  <aa>42</aa>
  <rpc-error>
    <error-info>
      <bad-element>bb</bad-element>
    </error-info>
  </rpc-error>
</foo>
```

assigned-by

- There is no formal way for a client to know if the server will assign a value for a missing optional leaf.
- Proposed solution:
  - Add a new statement
    - `assigned-by ( “user” / “system”)`
    - default is assigned-by user

create-only

There was some discussion about adding a statement to 'leaf' which would tell the system that the leaf can be set one time only. Once set, the value cannot be changed.

However, if the list or container containing the leaf is deleted, and then recreated, the leaf's value can be changed.

The main problem with this is that if you save a copy-config as a backup, you might not be able to copy-config it back.

Recommendation: Do not add this.

Remove some CLRs...

- Allow empty in unions
  - interim consensus, need confirmation on ML
- Allow keyref/leafref in unions
  - interim consensus, need confirmation on ML

Actions (a.k.a. rpc in list) 1(2)

- Provides encapsulation
  - If an operation affects a particular object, put the definition of the operation together with the object definition.
- Provides scoped names for operations
- Fits nicely into a subagent architecture
- Simplified access control

Actions (a.k.a. rpc in list) 2(2)

```
list interface {
    key name;
    leaf name {
        type string;
    }
    action restart {
        input {
            leaf immediate {
                type boolean;
            }
        }
    }
}
```

```
<rpc message-id="101">
  <yang:action>
    <interface xmlns=”http://example.com/if”>
      <name>eth0</name>
      <reset>
        <immediate>true</immediate>
      </reset
    </interfaces>
  </yang:action>
</rpc>
```