YANG Status

IETF 73
Martin Björklund
mbj@tail-f.com

Status

- Draft -02 available
- Interim held in Washington D.C. in October
  - Lots of detailed discussion on open issues
  - Most issues addressed in the -02 draft
- This presentation covers the changes since -01

Canonical form of data types

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

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.

Refine, when, and augment 1(2)

Harmonize and simplify the syntax.

Old:

```
uses my-grouping {
    container connection {
        leaf port {
            default 80;
        }
    }
}
augment connection {
    leaf http-version { ... }
}
```

New:

```
uses my-grouping {
    refine “connection/port” {
        default 80;
    }
    augment “connection” {           // this should be 'extend'
        leaf http-version { ... }
    }
}
```

Refine, when, and augment 2(2)

Old:

```
container server {
    leaf type {
        ...
    }
    augment . {
        when “../type = http”;
        container http-settings {
            ...
        }
    }
}
```

New:

```
container server {
    leaf type {
        ...
    }
    container http-settings {
        when “../type = http”;
        ...
    }
}
```

Features 1(2)

A module can be partitioned into a set of optional parts, where each part is conditional based on *features* implemented by a device.

```
module my-syslog {
    namespace “http://example.com/syslog”;
    ...
    feature local-storage {
        description "This feature means the device supports local
            storage (memory, flash or disk) that can be used to
            store syslog messages.";
    }

    container syslog {
        leaf local-storage-limit {
            if-feature local-storage;
            config false;
            description "The amount of local storage that can be
                used to hold syslog messages.";
        }
    }
}
```

Features 2(2)

The supported features are advertised in the <hello> message:

```
<hello>
  <capabilities>
    ...
    <capability>
      http://example.com/syslog?features=local-storage
    </capability>
  </capabilities>
</hello>
```

Deviations 1(2)

In reality, all devices cannot for various reason fully implement all standard modules. The *deviation* statement is used to formally define how a device deviates from a module.

```
deviation /base:system/base:daytime {
    deviate not-supported;
}


// Limits the number of supported name-servers
// to 3.
deviation /base:system/base:name-server {
    deviate replace {
        max-elements 3;
    }
}
```

Deviations 2(2)

Deviations are typically written in a module which contains deviations only, i.e. they are not mixed with normal definitions.

The device reports the name of this module in the <hello> message:

```
<hello>
  <capabilities>
    ...
    <capability>
      http://example.com/base?deviations=my-base-deviatons
    </capability>
  </capabilities>
</hello>
```

Identity and identityref

Problem: Need distributed reusable enumerations. The enumeration type is reusable, but centralized. An augmentable choice is distributed, but not reusable.

Solution: Borrowed from SMIng (and SMIv2)

```
identity crypto-alg {
    description "Base identity from which all crypto algorithms
                 are derived.";
}
identity des3 {
    base "crypto:crypto-alg";
    description "Triple DES crypto algorithm";
}
leaf crypto {
    type identityref {
        base "crypto:crypto-alg";
    }
}
```

XML Encoding:

```
<crypto xmlns:des="http://example.com/des">des:des3</crypto>
```

Update rules

- Protect old clients

  We want a client that uses version x of a module to be able to function when talking to a server implementing version x+1.

  For example, cannot add a mandatory leaf to a list.

- Protect importers

  A new published module version must not break existing other modules that imports from the module.

Import by revision

- Not yet in the draft.

Needed for the update rule “Protect importers”. With import by revision, it is safe to update typedefs and groupings in new versions of a module.

```
import common-types {
    prefix common;
    revision “2008-04-01”;
}
```

Typedefs and groupings are taken from the specified revision of the module.

If module A imports B, revision 2008-04-01, and A augments B, a device that implements A and B, must implement module B of revision 2008-04-01 or later.