YANG Open Issues

IETF 72
Martin Björklund
mbj@tail-f.com

Controlling Features 1(4)

- Problem:
  - The only way to define optional-to-implement data is to create a complete module for the data. With many small optional features, there will be many small modules. The capability list will be very long.

Controlling Features 2(4)

- Proposed solution:
  - Add two new statements, `feature` and `if-feature`, and add a new RPC `get-features`.

```
 // Example instance module

 feature “rollback-on-error” {
     description “...”;
}

type errorOptionType {
     enum “stop-on-error”;
     enum “continue-on-error”;
     enum “rollback-on-error” {
         if-feature “rollback-on-error”;
     }
}
```

Controlling Features 3(4)

```
// An example where an entire subtree is optional
// to implement

container server {
    ...
    container advanced {
        if-feature advanced-stuff;

        leaf foo { ... }
        ...
    }
}
```

Controlling Features 4(4)

```
// New RPC

rpc get-module-features {
    input {
        leaf namespace {
            type inet:uri;
        }
    }
    output {
        list features {
            leaf namespace {
                type inet:uri;
            }
            leaf-list feature {
                type string;
            }
        }
    }
}
```

Import by revision

- Problem
  - The current import mechanism leads to the conclusion that once a grouping or typedef is defined, it can never be changed.
- Proposed solution
  - Allow an optional import by revision:

```
module foo {
    ...
    import bar-types {
        prefix bar
        revision “2008-07-31”;
    }
    container x {
        uses bar:y;
    }
}
```

Revision 1(2)

- Problem:
  - Should the `revision` statement be mandatory?
- Proposed solution:
  - Make it mandatory.
    - A revision is important for schema discovery to function properly.
    - Necessary for import by revision.

Revision 2(2)

- Problem:
  - The `revision` statement's argument is currently a date string: YYYY-MM-DD. Some said that this is too restrictive; maybe a module has to be published more than one time per day.
- Proposed solution:
  - Keep as it is, this is not a problem
- Alternative 1:
  - Append an optional simple integer to the date:
    - 2008-07-26.1

Revision 3(3)

- Alternative 2:
  - Use RFC 3339 date-time:
    - 2008-07-26T14:48:10+02:00
- Alternative 3:
  - Ditto but UTC only (for simpler comparisons)

Clean up augment and uses 1(4)

- Problem:
  - The `augment` statement is used for two purposes; adding nodes to an external module's structure, and adding nodes to a local usage of grouping:

```
// external augment
augment “/if:interfaces/if:interface” {
    leaf my-interface-param { ... }
}


// local augment
uses Interface;
augment interface/unit {
    leaf my-vlan-param { ... }
}
```

Clean up augment and uses 2(4)

- Proposed solution:
  - move the `augment` statement inside the `uses`:

```
uses Interface {
    augment interface/unit {
        leaf my-vlan-param { ... }
    }
}
```

Clean up augment and uses 3(4)

- Problem:
  - The current way of doing refinements does not match how `augment` is used, and it makes the other statements' grammar context-dependent. E.g. a `leaf` within a `uses` cannot specify a `type`.

```
// current refinement
uses Interface {
   container interface {
       leaf mtu {
           default 1500; // add default
        }
   }
}
```

Clean up augment and uses 4(4)

- Proposed solution:
  - Add a new `refine` statement with similar syntax to `augment`.

```
uses Interface {
    refine interface/mtu {
        default 1500;
    }
    augment interface/unit {
        leaf my-vlan-param { ... }
    }
}
```

Server Variance

- Data model anticipated variance
  - features
    - optional-to-implement data
    - type variance
  - server-assigned leafs
  - server-supplied defaults
- Server specific legal variance
  - limits on max-elements
  - changing from config to non-config
- Server specific illegal variance
  - changing a list to a leaf; changing keys, ...

Server-supplied values

- Problem:
  - 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

Server-supplied defaults

- Problem:
  - There is no formal way to specify in the model where the server is free to choose its own default value, and there is no way for a client to learn server-specific default values.
- Proposed solution:
  - Add parameters to modules:

```
module foo {
    parameter mtu-default;
    ...
    leaf mtu {
        type uint32;
        default $mtu-default;
    }
}
```

Multiple patterns

- Problem:
  - Currently, there can be one `pattern` restriction to string types.
- Proposed solution:
  - Allow multiple `pattern` statements, which would be ANDed together. Each `pattern` can have it's own `error-message` which gives more precise errors. This is in alignment with XSD, which allows multiple patterns.

Conditional content

A proposal on the mailing list was to add the `when` statement to other statements, not only augment:

```
container ethernet {
    when “../ifType == 'ethernet'”;

    // ethernet specific stuff here
}
```

Why Constrain keyref?

- Problem:
  - A question on the ML was why a config keyref is constrained to refer to config data only.
  - A related question was why the keyref target must exist in a valid configuration. Sometimes it makes sense to say that something happens if the target exists, but it is perfectly ok if the target does not exist.
- Proposed solution:
  - Make it possible to mark the keyref to allow unsatisfied reference. Details TBD.

Other stuff

- Change `presence` to boolean? If so, is there a better word than presence? presence-meaningful.

- “Augment enumeration”. Is current solution with choice good enough? It means the designer must design for extensibility.

- Can the keys of a list be config false, while the rest of the list is config? Can one be config false and one config true? Should we describe this?

Overlays 1(2)

Q. should there be a std way to add vendor-specific annotations to existing modules? But the technique can be used for other things, see slide on implementation specific defaults.

Overlays 2(2)

Summary of mailing list discussion: overlay vs. annotate stmt.

annotate: does not work for things w/o identifiers (you cannot annotate 'uses', 'augment', 'import') we must put typedef, grouping in same naming scope. can only annotate schema tree, but maybe that's good enough?

overlay: introduces context-dependent grammar, e.g. a list stmt in an overlay must not have a key substmt.