Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 22, 2014

A. Bierman
YumaWorks

September 18, 2013

YANG Conformance Specification
draft-bierman-netmod-yang-conformance-01

Abstract

This document describes conformance specification and advertisement mechanisms for NETCONF servers implementing YANG data model modules.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 22, 2014.

Copyright Notice

Table of Contents

1. Introduction 4
1.1. Terminology 4
1.1.1. NETCONF 4
1.1.2. YANG 4
1.1.3. Terms 5
2. Problems With YANG Conformance Mechanisms 7
2.1. YANG Conformance Specification Issues 7
2.1.1. Import by Revision is Unusable for Conformance 7
2.1.2. YANG Conformance Specification is Too Simplistic . . . 8
2.1.3. YANG Deviation Statements Do Not Help 9
2.1.4. Single Module Conformance in Not Expressive Enough . . 9
2.2. Module Capability Advertisement Issues 10
3. Solution Overview 11
3.1. Objectives 11
3.2. YANG Package 11
3.3. YANG Package File 12
3.4. Conformance Profile 12
3.5. Conformance Profile Capability 12
3.6. YANG Conformance Examples 13
4. YANG Conformance Statements 20
4.1. The package Statement 20
4.1.1. The package Substatements 20
4.2. The category Statement 20
4.2.1. The category Substatements 21
4.3. The subcategory Statement 21
4.4. The profile Statement 21
4.4.1. The profile Substatements 21
4.5. The include-profile Statement 21
4.6. The require-module Statement 22
4.6.1. The require-module Substatements 22
4.7. The min-revision Statement 22
4.8. The max-revision Statement 23
4.9. The require-conformance Statement 23
4.10. The require-feature Statement 24
4.10.1. The require-feature Substatements 24
4.10.2. Usage Example 24
4.11. The require-object Statement 25
4.11.1. The require-object Substatements 25
4.11.2. Usage Example 25
4.12. The require-package Statement 25
4.12.1. The require-package Substatements 26
4.12.2. Usage Example 26
4.13. The require-profile Statement 26
4.14. The require-capability Statement 26
4.14.1. The require-capability Substatements 26
4.14.2. Usage Example 27

5. Updating a YANG Package 28
6. YANG Package Conformance Advertisement 29
7. YANG Conformance ABNF 30
8. IANA Considerations 34
9. Security Considerations 35
10. Open Issues 36
11. Change Log 37
11.1. 00-01 37
12. Normative References 38
Author's Address 39

1. Introduction

There is a need for standard mechanisms to allow YANG [RFC6020] data model designers to express more precise and robust conformance levels for server implementations of a particular YANG module, or set of YANG modules.

There is also a need for standard mechanisms to allow NETCONF [RFC6241] servers to precisely advertise the conformance level of each YANG module it supports.

This document describes some problems with the current conformance specifications mechanisms in YANG and conformance advertisement mechanisms in NETCONF. Solution proposals are also presented to address these problems.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- capability
- client
- datastore
- protocol operation
- server

1.1.2. YANG

The following terms are defined in [RFC6020]:

- data node
- extension
- feature

- grouping
- identity
- module
- notification
- submodule
- typedef

1.1.3. Terms

The following terms are used within this document:

- conditional node: An object that has one or more "if-feature" sub-statements associated with it. Note that objects affected by "when" statements are not considered conditional for conformance purposes.
- conformance profile: A set of requirements that a server must support to comply with a given service level. These requirements can be specified in terms of required YANG modules (possibly with specific YANG features and objects supported), other conformance profiles, and/or NETCONF capability URIs (for server functionality that is not specified in YANG modules).
- import-by-revision: A YANG import statement that includes a revision-date statement. This specifies the exact revision of the YANG module to import, instead of the server picking the revision to import.
- module base: There is an implied "base" version of the module, which includes all statements which are not conditional. The module base may be empty, a subset of all statements, or the entire module.
- object: a conceptual data structure represented by a YANG data, rpc, or notification statement.
- schema tree: The conceptual tree of all objects derived from the set of all YANG modules supported by the server. This tree only includes conditional nodes if all corresponding if-feature statements are "true". Any deviation statements have also been conceptually applied to the schema tree as well.

- YANG feature set: The set of all objects from a particular module that contain an if-feature statement that corresponds to a particular YANG feature statement.

- YANG package: A set of conformance profiles that can be extended over time. Also called "package".

2. Problems With YANG Conformance Mechanisms

This section describes some perceived deficiencies with the current data model conformance specification and server conformance advertisement mechanisms used in NETCONF.

2.1. YANG Conformance Specification Issues

The YANG data modeling language provides many powerful data modeling constructs to allow the automation of network configuration protocol operations. However it does not provide enough control over the precise server conformance levels that a client can expect. This has a negative impact on interoperability.

A YANG module is conceptually divided into the module base and zero or more purely optional YANG feature sets.

This approach does not allow enough flexibility and can become difficult to use as the module size and number of YANG feature statements increases. A set of boolean flags that are logically combined as an "AND" expression is too simplistic a mechanism for expressing the criteria for specifying conditional conformance requirements.

2.1.1. Import by Revision is Unusable for Conformance

YANG provides a mechanism to import an exact revision of an external module in order to freeze conformance requirements for a module. If this is not used then the YANG compiler will most likely use the latest revision of the imported module that happens to be implemented by the server.

If new data nodes, notifications, or protocol operations are added to an imported module over time, then it can appear to a NETCONF client that the new objects are implemented if the imported module is updated but not all the modules that import it. Objects using imported typedefs will change syntax and semantics if the typedef (or any typedef it refines) is changed.

Unless import-by-revision is used everywhere an import is used within the dependency chain, the exact module definition cannot really be frozen for conformance purposes.

A server is not required to support multiple revisions of the same module at the same time. This may be very confusing to the client, and complex to implement as well. Instead, servers usually allow only one revision of each module to be implemented within the system.

If import-by-revision is used, then creating a new revision of the imported module requires that the import statements in all the importing modules be updated to use the new revision date. This requires a revision change, so any module that imports those modules also needs to be updated to specify the new revision date of those importing modules. This ripple effect can cause a lot of modules to be updated. It may not be possible to update a module import date in some cases, if that would incorrectly advertise to the client that new objects were implemented by the server.

2.1.2. YANG Conformance Specification is Too Simplistic

Conformance requirements can change over time. New use cases and new consensus about optionality can occur. A conformance statement for each use-case is needed, not just one or more (implied) conformance statements per module.

There are no mechanisms to clearly specify external module dependencies. There is no way to indicate the exact portions of an imported module which are required to comply with a particular conformance level for the importing module. There is no way to specify that multiple modules are required to provide a high-level service.

It is impossible to predict all valid use cases at design time. Partitioning a module into a base plus purely optional features can only account for the features and use cases known at the time. Future designers cannot alter if-feature statements or add new if-feature statements to an augmented module.

YANG features are purely optional to implement. There is no way to specify that a set of objects are conditionally mandatory, based on some data-model specific criteria. Conditions could be expressed with XPath must or when expressions, but this has to be repeated everywhere it is used and the set of objects with the same conditionally mandatory properties is un-named and hard for the reader to identify.

YANG features are too simplistic. They are good for a small number of use-cases within one module. Once there are lots of features, interactions between features, and refined use-cases, they turn the module into a bowl of boolean spaghetti.

YANG features are not really purely optional in practice. Sometimes they are used to express separate roles or service subsets within the module. It is difficult for the reader to identify the valid combinations of purely optional YANG features that represent high-level roles. The YANG if-feature statements are logically combined

as a boolean "AND" expression and not very flexible. YANG description statements are not really machine parsable, so these high-level roles or service groupings are not easily identifiable.

2.1.3. YANG Deviation Statements Do Not Help

YANG deviations could possibly be used as a low-level conformance solution, but they are undesirable and not used by server vendors. YANG deviation statements provide a fairly comprehensive "patch" mechanism to conceptually alter YANG data definition statements. This alteration, or declaration of non-implementation, describes how a server deviates from the standard data definitions.

These statements are not allowed to appear in standard YANG modules, and it turns out that vendors would rather not specify exactly how their server is non-compliant to a standard YANG module. A vendor would rarely need a deviation statement for their own YANG data modules.

YANG deviation statements are too low-level anyway, even if vendors were willing to use them. They do not fully address the future use-case problem because they can only be used to make specific patches to data statements.

2.1.4. Single Module Conformance in Not Expressive Enough

YANG conformance applies only to one module. There are no mechanisms to precisely identify the conformance relationship between modules. Since YANG is designed to be modular and reusable, it is quite likely that a high-level feature or service will be specified with more than one YANG module.

All the top-level definitions are imported from a module whether the importing module uses all the definitions or not. This is too general from a conformance perspective. Sometimes modules are imported just for typedefs or identities, which are always part of the base.

If a module augments a node in another module, it does not imply that it supports all other objects from that module. YANG conformance does not actually address any relationship between modules. There are no mechanisms to express multi-module conformance requirements.

It is difficult for a client application developer to identify the high level server capabilities from a large set of module capabilities. There are no formal mechanisms to identity the definition of a high-level service across multiple modules.

2.2. Module Capability Advertisement Issues

NETCONF servers advertise the YANG modules they support as <capability> URI strings in the <hello> message. The complete list of modules used by the server needs to be advertised in order for the client application to correctly parse the YANG modules and reproduce the schema tree used by the server. However the client does not really know which modules are advertised for full conformance, and which are advertised for partial conformance (such as importing typedef and identity statements from the module).

3. Solution Overview

3.1. Objectives

The solution in this document attempts to achieve several objectives:

- Provide simple documentation mechanisms that are readable and easy to understand.
- Provide simple mechanisms that can scale in usage from one module to thousands of modules.
- Provide per use-case conformance profiles, which allow multiple conformance levels to be specified for a single module.
- Provide per use-case conformance profiles, which allow multiple modules to be specified for a single conformance profile.
- Clarify the usage relationship between an augmented module and the augmenting module.
- Clarify the usage relationship between modules that represent parts of the same conceptual high-level service.
- Provide the ability to specify a stable conformance definition that cannot implicitly change if YANG modules are updated.
- Provide the ability to specify the YANG features that must be supported by a server to meet conformance requirements.

3.2. YANG Package

A YANG package is a conformance definition for zero or more YANG modules and/or NETCONF protocol capabilities. Each package has zero or more conformance profiles that describe the server implementation requirements to conform to a specific profile within a package. A YANG package without any conformance profile statements can be used as a placeholder to reserve the package name, but it cannot be advertised as a YANG package capability.

YANG packages are static representations of YANG conformance, meaning there are no server-dependent variables (e.g, set of purely optional YANG features selected by the server). Instead a conformance profile specifies which YANG features a server needs to support to conform to the profile.

YANG packages are defined using a text file similar to YANG modules. However they are separate from YANG modules, since a package can

require more than one YANG module for conformance.

Unlike YANG modules, YANG package definitions do not represent content that would appear in a protocol message. They represent server conformance requirements and are therefore separate from YANG module definitions.

A YANG package is advertised with a <capability> URI string, similar to a YANG module. A NETCONF server will advertise all its supported package capability statements in the <hello> message it sends to each client.

3.3. YANG Package File

A YANG package file consists of UTF-8 characters. The basic syntax is exactly the same as for YANG modules. Several YANG statements are "imported" from the YANG ABNF, and some new statements are defined. Specifically, YANG package syntax is the same as RFC 6020, sections 6, 6.1, 6.2, and 6.3.

[FIXME: not all namespaces in sec. 6.2.1 are supported and a namespace for YANG package names is not defined there.]

At least one revision statement MUST be present in a YANG package file. A new revision MUST be added each time the YANG package file is published. This requirement is more strict than RFC 6020 to ensure that conformance requirements can be properly identified for each server implementation.

3.4. Conformance Profile

A conformance profile represents a conceptual set of server implementation requirements to meet one use-case or variant of the conceptual service represented by the YANG package.

A conformance profile can overlap or even include other conformance profiles. It is a data-model specific matter what requirements make operational sense.

A server can only conform to one conformance profile within a YANG package. Although profile contents can overlap, only one profile per package can be active on a server at a time.

3.5. Conformance Profile Capability

A new NETCONF capability URI is defined to advertise YANG package conformance. A server will announce conformance for a specific conformance profile for each YANG package it supports. Refer to

Section 6 for details on YANG package conformance advertisement.

3.6. YANG Conformance Examples

In this example, 1 conformance profile called "base" is defined for the YANG package named "ietf-types-pkg".

```
  package ietf-types-pkg {

    namespace "urn:ietf:params:xml:ns:yang:ietf-types-pkg";
    prefix "typespkg";
    organization
      "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
    contact " ** WG Chairs ** ";
    description
       "This package defines a conformance profile for the standard
        typedef statements.";

    revision 2013-09-16 {
       description "First revision";
       reference "TBD";
    }

    category general {
      subcategory types;
    }

    profile base {
      description "Basic requirements for YANG types.";

      require-module ietf-yang-types {
        min-revision "2013-07-15";
        require-conformance import;
        description
          "Support for YANG types  is required.";
        reference "RFC 6991, section 3.";
      }

      require-module ietf-inet-types {
        min-revision "2013-07-15";
        require-conformance import;
        description
          "Support for INET types  is required.";
        reference "RFC 6991, section 4.";
      }
    }
  }
```

In this example, 4 different conformance profiles are defined for the YANG package named "ietf-routing-pkg":

- base: a server that supports the base routing profile. This profile is probably not useful without adding routing protocols, but it is needed for extensibility.
- ipv4: a server that supports IPv4 routing configuration
- ipv6: a server that supports IPv6 routing configuration
- ip: a server that supports IPv4 and IPv6 routing configuration

```
  package ietf-routing-pkg {

    namespace "urn:ietf:params:xml:ns:yang:ietf-routing-pkg";
    prefix "rtpkg";
    organization
      "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
    contact " ** WG Chairs ** ";
    description
       "This package defines conformance profiles for IPv4 and IPv6
        routers.";
    reference "draft-ietf-netmod-routing-cfg-10.txt";

    revision 2013-09-16 {
       description "First revision";
       reference "TBD";
    }

    category protocols {
      subcategory routing;
      subcategory ip;
    }

    profile base {
      description "Base module requirements for routing";
      reference "draft-ietf-netmod-routing-cfg-10.txt";

      require-package ietf-types-pkg;

      require-module ietf-routing {
         min-revision 2013-07-13;
         require-conformance full;
         description "The base routing module is required";
      }
```

```
        require-module ietf-interfaces {
           min-revision 2013-07-04;
           require-conformance augment;
           description
             "The interface and interface-state tables are augmented.";
        }

      }

      profile ipv4 {
        description "Base module requirements for routing";
        reference "draft-ietf-netmod-routing-cfg-10.txt";

        include-profile base;

        require-module ietf-ipv4-unicast-routing {
           min-revision 2013-07-13;
           require-conformance full;
        }
      }

      profile ipv6 {
        description "Base module requirements for routing";
        reference "draft-ietf-netmod-routing-cfg-10.txt";

        include-profile base;

        require-module ietf-ipv6-unicast-routing {
           min-revision 2013-07-13;
           require-conformance full;
        }
      }

      profile ip {
        description "Base module requirements for routing";
        reference "draft-ietf-netmod-routing-cfg-10.txt";

        include-profile ipv4;

        include-profile ipv6;
      }

    }
```

In this example, 5 different conformance profiles are defined for the YANG package named "ietf-netconf-pkg":

- core: core NETCONF functionality.

- running: a server that supports writing directly to the running datastore.

- startup: a server that supports writing directly to the running datastore and also has a separate startup datastore.

- candidate:- running: a server that supports writing to the candidate datastore and committing all edits at once to the running datastore.

- confirmed:- running: a server that supports the candidate conformance profile and also supports the confirmed commit operations.

```
  package ietf-netconf-pkg {

    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-pkg";
    prefix "ncpkg";
    organization
      "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
    contact " ** WG Chairs ** ";
    description
       "This package defines some conformance profiles for the
        NETCONF protocol.";

    revision 2013-09-16 {
       description "First revision";
       reference "TBD";
    }

    category protocols {
      subcategory netconf;
    }

    profile core {
      description
        "Basic requirements for complete NETCONF servers.";

      require-package ietf-types-pkg;

      require-capability "urn:ietf:params:netconf:base:1.1" {
        description "NETCONF base protocol is required.";
        reference "RFC 6241, section 8.1";
      }
```

```
        require-capability
          "urn:ietf:params:netconf:capability:xpath:1.0" {
          description "XPath filtering is required.";
          reference "RFC 6241, section 8.9";
        }

        require-capability
          "urn:ietf:params:netconf:capability:validate:1.1" {
          description "NETCONF :validate capability is required.";
          reference "RFC 6241, section 8.6";
        }

        require-capability
          "urn:ietf:params:xml:ns:netconf:partial-lock:1.0" {
          description
            "Partial lock capability is required. The YANG module
             ietf-netconf-partial-lock.yang is non-normative
             so a require-module statement is not used instead.";
          reference "RFC 5717, section 4";
        }

        require-capability
          "urn:ietf:params:netconf:capability:with-defaults:1.0" {
          description
            "With defaults capability advertisement is required.";
          reference "RFC 6243, section 4.3";
        }

        require-capability
          "urn:ietf:params:netconf:capability:notification:1.0" {
          description
            "Notification delivery support is required.";
          reference "RFC 5277, section 3.1";
        }

        require-capability
          "urn:ietf:params:netconf:capability:interleave:1.0" {
          description
            "Interleave of commands is required while notification
             delivery is active .";
          reference "RFC 5277, section 6";
        }

        require-module ietf-netconf-with-defaults {
          description
            "Support for <with-defaults> RPC parameter is required.";
          reference "RFC 6243, section 5";
        }
```

```
        require-module ietf-netconf-acm {
          description
            "Base module implementation of NACM is required.";
          reference "RFC 6536, section 3.5.2";
        }

        require-module ietf-netconf-monitoring {
          description
            "Implementation of NETCONF monitoring is required.";
          reference "RFC 6022, section 5";
        }

        require-module ietf-netconf-notifications {
          description
            "Implementation of NETCONF base notifications is
             required.";
          reference "RFC 6470, section 2.2";
        }
      }

      profile running {
        description
          "Basic requirements for a complete NETCONF server
           that supports writing directly to the the running
           datastore.";

        include-profile core;

        require-capability
          "urn:ietf:params:netconf:capability:writable-running:1.0" {
          description
            "NETCONF :writable-running capability is required.";
          reference "RFC 6241, section 8.2";
        }

        require-capability
          "urn:ietf:params:netconf:capability:rollback-on-error:1.0" {
          description
            "NETCONF :rollback-on-error capability is required.";
          reference "RFC 6241, section 8.5";
        }
      }

      profile startup {
        description
          "Basic requirements for a complete NETCONF server
           that supports writing directly to the the running
           datastore and also have a distinct startup datastore.";
```

```
        include-profile running;

        require-capability
          "urn:ietf:params:netconf:capability:startup:1.0" {
          description
            "NETCONF distinct startup capability is required.";
          reference "RFC 6241, section 8.7";
        }
      }

      profile candidate {
        description
          "Basic requirements for a complete NETCONF server
           that supports the candidate datastore.";

        include-profile core;

        require-capability
          "urn:ietf:params:netconf:capability:candidate:1.0" {
          description "NETCONF :candidate capability is required.";
          reference "RFC 6241, section 8.3";
        }
      }

      profile confirmed {
        description
          "Basic requirements for a complete NETCONF server
           that supports the candidate datastore, and confirmed
           commit functionality.";

        include-profile candidate;

        require-capability
          "urn:ietf:params:netconf:capability:confirmed-commit:1.1" {
          description
            "NETCONF :confirmed-commit capability is required.";
          reference "RFC 6241, section 8.4";
        }
      }
    }
```

4. YANG Conformance Statements

4.1. The package Statement

The "package" statement defines the YANG package's name, and contains all YANG package header information and conformance profile statements. The "package" statement's argument is the name of the YANG package, followed by a block of substatements that hold detailed package information. The package name follows the rules for identifiers in RFC 6020, section 6.2.

A YANG package name is defined in the same conceptual namespace as YANG module names. The same rules for selecting non-conflicting names apply as defined in RFC 6020, section 7.1.

An IANA registry for YANG package names will be needed, similar to mechanism described in RFC 6020, section 14.

4.1.1. The package Substatements

Substatement	Reference	Cardinality
category	4.2	0..1
contact	RFC 6020, 7.1.8	0..1
description	RFC 6020, 7.19.3	0..1
namespace	RFC 6020, 7.1.3	1
organization	RFC 6020, 7.1.7	0..1
prefix	RFC 6020, 7.1.4	1
profile	4.4	1..n
reference	RFC 6020, 7.19.4	0..1
revision	RFC 6020, 7.1.9	1..n

4.2. The category Statement

The "category" statement, which is optional, takes as an argument the category identifier that best describes the type of functionality provided by the YANG package.

There are no constraints or guidelines on selection of a classification system at this time. It is expected that the IETF will create a classification system for standard YANG modules.

4.2.1. The category Substatements

Substatement	Reference	Cardinality
subcategory	4.3	0..n

4.3. The subcategory Statement

The "subcategory" statement, which is optional, takes as an argument the sub-category identifier that best describes the type of functionality provided by the YANG package. Zero or more subcategory statements are allowed within each category statement;

4.4. The profile Statement

The "profile" statement is used to define one conformance profile within a YANG package. It takes as an argument the profile name, which is followed by a block of substatements that hold detailed conformance information. The package name follows the rules for identifiers in RFC 6020, section 6.2.

4.4.1. The profile Substatements

Substatement	Reference	Cardinality
description	RFC 6020, 7.19.3	0..1
include-profile	4.5	0..n
reference	RFC 6020, 7.19.4	0..1
require-capability	4.14	0..n
require-module	4.6	0..n
require-package	4.12	0..n
status	RFC 6020, 7.19.2	0..1

4.5. The include-profile Statement

The "include-profile" statement is used to combine multiple conformance profiles from the same YANG package. It takes as an argument the name of the conformance profile to include. There are no substatements defined. All of the requirements in the included profile are also required in the profile that contains the include-profile statement.

If any require-module, require-package, and/or require-capability statements overlap due to multiple included profiles, then they are

logically combined such that all requirements from all profiles are included.

A profile MUST NOT include itself or any conformance profile that would cause itself to be included via a dependency loop.

4.6. The require-module Statement

The "require-module" statement is used to require support for some or all of the definitions in a specific module. It takes as an argument the name of the module to require, followed by a block of substatements that hold detailed module server support requirements.

A require-module statement MUST NOT specify the same module name as another require-module statement in the same profile statement.

Submodules are invisible for conformance purposes, because they are used as an implementation mechanism, and are not directly accessible from an external module.

4.6.1. The require-module Substatements

Substatement	Reference	Cardinality
description	RFC 6020, 7.19.3	0..1
max-revision	4.8	0..1
min-revision	4.7	0..1
reference	RFC 6020, 7.19.4	0..1
require-conformance	4.9	0..1
require-feature	4.10	0..n
require-object	4.11	0..n

4.7. The min-revision Statement

The "min-revision" statement is used to specify the minimum acceptable revision date for the required module. It takes as argument a date string in the form "YYYY-MM-DD" where "YYYY" is the year, "MM" is the month, and "DD" is the day.

It is used in combination with the "max-revision" statement to identify a range of release dates that are acceptable for profile conformance.

If the min-revision statement is not present, then there is no minimum revision date in the acceptable range.

If the min-revision statement is present, then only revisions of the required module released on or after that date are acceptable for profile conformance.

If the min-revision statement is present, and the max-revision statement is also present, then the min-revision statement MUST represent a date which is the same as or before the max-revision date.

This statement is needed if the conformance profile relies on definitions that were added to the required module in a particular revision.

4.8. The max-revision Statement

The "max-revision" statement is used to specify the maximum acceptable revision date for the required module. It takes as argument a date string in the form "YYYY-MM-DD" where "YYYY" is the year, "MM" is the month, and "DD" is the day.

It is used in combination with the "min-revision" statement to identify a range of release dates that are acceptable for profile conformance.

If the max-revision statement is not present, then there is no maximum revision date in the acceptable range.

If the max-revision statement is present, then only revisions of the required module released on or before that date are acceptable for profile conformance.

If the max-revision statement is present, and the min-revision statement is also present, then the max-revision statement MUST represent a date which is the same as or after the min-revision date.

This statement is needed if the conformance profile relies on definitions that have been changed to obsolete status in the required module, or have been extended or altered in a manner that is not required in the conformance profile.

4.9. The require-conformance Statement

The "require-conformance" statement is used to describe the type of module conformance that is needed to meet the conformance profile requirements. Its argument is an enumerated string value indicating the type of module conformance required.

There are 4 types of module conformance supported:

- full: Full implementation of the module base is required. This is the default value if the require-conformance statement is not present.

- augment: full implementation of the objects that are augmented in this module (from the augmenting module) is required.

- import: the module is required for meta-data definitions, which includes extension, typedef, grouping, identity, and feature statements. No implementation of any objects in the module is required.

- ad-hoc: Partial implementation of the module base and/or some conditional nodes is required. The require-module statement SHOULD contain "require-object" statements to identify the ad-hoc requirements.

4.10. The require-feature Statement

The "require-feature" statement is used to indicate that the specified YANG feature set is required for profile conformance. It takes as argument the name of the YANG feature that is required, and is followed by a block of substatements that describe the YANG feature usage within the conformance profile.

4.10.1. The require-feature Substatements

Substatement	Reference	Cardinality
description	RFC 6020, 7.19.3	0..1
reference	RFC 6020, 7.19.4	0..1

4.10.2. Usage Example

```
require-module ietf-ip {
  min-revision 2013-02-11;
  require-conformance full;
  require-feature ipv4-non-contiguous-netmasks {
    description
      "Configuration of non-contiguous subnet masks
       is required.";
    reference
       "RFC XXXX; Section XXXX";
  }
}
```

4.11. The require-object Statement

The "require-object" statement is used to indicate that the specified YANG object is required for profile conformance. It takes as argument the path string identifying the object. This is similar to a YANG "absolute-schema-nodeid" except that prefixes are not allowed. Only objects defined in the required module can be specified with this statement.

[FIXME: there is no way to specify that the objects that 1 module adds to another with augment-stmt are required in ad-hoc mode.]

4.11.1. The require-object Substatements

Substatement	Reference	Cardinality
description	RFC 6020, 7.19.3	0..1
reference	RFC 6020, 7.19.4	0..1

4.11.2. Usage Example

```
require-module ietf-system {
  require-conformance ad-hoc;
  require-object /system-state/clock/current-datetime {
    description
      "The current system time must be provided.";
    reference
       "RFC XXXX; Section XXXX";
  }
}
```

4.12. The require-package Statement

The "require-package" statement is used to require support for an external YANG package. It takes as an argument the name of the YANG package to require, followed by a block of substatements that hold detailed server support requirements.

A require-package statement MUST NOT specify the same YANG package name as another require-package statement in the same profile statement.

4.12.1. The require-package Substatements

Substatement	Reference	Cardinality
description	RFC 6020, 7.19.3	0..1
max-revision	4.8	0..1
min-revision	4.7	0..1
reference	RFC 6020, 7.19.4	0..1
require-profile	4.13	0..1

4.12.2. Usage Example

```
require-package ietf-routing-pkg {
  min-revision 2013-09-16;
  require-profile ipv4;
  description
    "Support for IPv4 routing configuration is required.";
}
```

4.13. The require-profile Statement

The "require-profile" statement is used to require support for a specific conformance profile within an external YANG package. It takes as an argument the name of the conformance profile to require.

4.14. The require-capability Statement

The "require-capability" statement is used to indicate that the specified NETCONF capability URI is required for profile conformance. It takes as argument a URI string identifying the NETCONF capability that is required, and is followed by a block of substatements that describe the NETCONF capability usage within the conformance profile.

4.14.1. The require-capability Substatements

Substatement	Reference	Cardinality
description	RFC 6020, 7.19.3	0..1
reference	RFC 6020, 7.19.4	0..1

4.14.2. Usage Example

```
profile full-notifications {
  description
    "A profile for requiring full standard NETCONF
     notification functionality.";
  require-capability
    "urn:ietf:params:netconf:capability:notification:1.0" {
    description
      "Support for NETCONF notifications is required.";
    reference "RFC 5277, section 3.1.1";
  }
  require-capability
    "urn:ietf:params:netconf:capability:interleave:1.0" {
    description
      "Support for the ability to accept <rpc> requests when
       NETCONF notification delivery is active is required.";
    reference "RFC 5277, section 6.3";
  }
}
```

5. Updating a YANG Package

A YANG conformance profile definition needs to be altered very carefully after it has been published, in order not to break old clients that expect certain server behavior.

When a new revision of a YANG package is published, the following restrictions apply:

- An existing conformance profile definition MAY be altered to correct errors in the definition.
- New statements (e.g. new requirements) MAY be added to an existing conformance profile.
- Existing requirements MUST NOT be removed from a conformance profile.
- Existing requirements MUST NOT be altered such that the existing functionality would be taken away from clients.
- Existing requirements MAY be altered such that the existing functionality appears unaffected to existing clients that are using a previous revision of the conformance profile.
- An existing conformance profile can be split into multiple new conformance profiles, if the existing conformance profile adds "include-profile" statements such that the required functionality for any existing conformance profile does not change.

6. YANG Package Conformance Advertisement

The YANG Package Conformance capability is used to allow the client to quickly identify which packages and conformance profiles are supported by a particular NETCONF server. The server will advertise each supported YANG package, similar to the YANG module conformance advertisement in RFC 6020, section 5.6.4.

The YANG package namespace URI MUST be advertised as a capability in the NETCONF <hello> message to indicate support for a specific conformance profile within the YANG package. The capability URI MUST be of the form:

```
conf-capability-string   = namespace-uri [ parameter-list ]
parameter-list      = "?" parameter *( "&" parameter )
parameter           = package-parameter /
                      revision-parameter /
                      profile-parameter
package-parameter   = "package=" package-name
revision-parameter  = "revision=" revision-date
profile-parameter   = "profile=" profile-name
```

Where:

- "package-name" is the name of the YANG package
- "revision-date" is the revision date of the YANG package
- "profile-name" is the name of a conformance profile within the YANG package

All 3 parameters MUST be present in the capability string. Refer to Section 4.1 for details on the acceptable values for these parameters.

Example: (capability string wrapped for display purposes only)

```
<capability>urn:ietf:params:xml:ns:yang:ietf-routing-pkg?
    package=ietf-routing-pkg&revision=2013-09-16&profile=ipv4
</capability>
```

7. YANG Conformance ABNF

<CODE BEGINS> file "yang-conformance.abnf"

```
   package-stmt     = optsep package-keyword sep identifier-arg-str
                      optsep
                      "{" stmtsep
                          module-header-stmts
                          meta-stmts
                          revision-stmts
                          package-classify-stmts
                          package-body-stmts
                      "}" optsep

    package-classify-stmts = [category-stmt]

    category-stmt = category-keyword sep identifier-arg-str optsep
                    (";" /
                    "{" stmtsep
                        *(subcategory-stmt stmtsep)
                    "}")

    subcategory-stmt = subcategory-keyword sep string stmtend

    package-body-stmts  = *(profile-stmt stmtsep)

    profile-stmt =    profile-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                           *(include-profile-stmt stmtsep)
                           *(require-package-stmt stmtsep)
                           *(require-capability-stmt stmtsep)
                           *(require-module-stmt stmtsep)
                        "}")

    include-profile-stmt = include-profile-keyword sep
                           identifier-arg-str stmtend

    require-module-stmt = require-module-keyword sep
                      identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [min-revision-stmt stmtsep]
```

```
                            [max-revision-stmt stmtsep]
                            [require-conformance-stmt stmtsep]
                            [description-stmt stmtsep]
                            [reference-stmt stmtsep]
                            *(require-feature-stmt stmtsep)
                            *(require-object-stmt stmtsep)
                         "}")

       require-object-stmt = require-object-keyword sep
                           require-object-arg-str optsep
                           (";" /
                            "{" stmtsep
                                ;; these stmts can appear in any order
                                [description-stmt stmtsep]
                                [reference-stmt stmtsep]
                            "}")

       require-object-arg-str = < a string that matches the rule
                                  require-object-arg >

       require-object-arg   = pkg-absolute-schema-nodeid

       require-package-stmt = require-package-keyword sep
                           identifier-arg-str optsep
                           (";" /
                            "{" stmtsep
                                ;; these stmts can appear in any order
                                [min-revision-stmt stmtsep]
                                [max-revision-stmt stmtsep]
                                [require-profile-stmt stmtsep]
                                [description-stmt stmtsep]
                                [reference-stmt stmtsep]
                            "}")

       require-profile-stmt = require-profile-keyword sep
                           identifier-arg-str stmtend

       require-capability-stmt = require-capability-keyword sep
                           uri-str optsep
                           (";" /
                            "{" stmtsep
                                ;; these stmts can appear in any order
                                [description-stmt stmtsep]
                                [reference-stmt stmtsep]
                            "}")

       min-revision-stmt = min-revision-keyword sep date-arg-str stmtend
```

```
      max-revision-stmt = max-revision-keyword sep date-arg-str stmtend

      require-feature-stmt = require-feature-keyword sep
                         identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; these stmts can appear in any order
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                          "}")

      require-conformance-stmt = require-conformance-keyword sep
                          require-conformance-arg-str stmtend

      require-conformance-arg-str =
                           < a string that matches the rule
                             require-conformance-arg >

      require-conformance-arg = full-keyword /
                                augment-keyword /
                                import-keyword /
                                ad-hoc-keyword

      pkg-schema-nodeid   = pkg-absolute-schema-nodeid /
                            pkg-descendant-schema-nodeid

      pkg-absolute-schema-nodeid = 1*("/" pkg-node-identifier)

      pkg-descendant-schema-nodeid =
                            pkg-node-identifier
                            pkg-absolute-schema-nodeid

      pkg-node-identifier = identifier

      ;; new keywords
      ad-hoc-keyword              = 'ad-hoc'
      augment-keyword             = 'augment'
      category-keyword            = 'category'
      conformance-keyword         = 'conformance'
      full-keyword                = 'full'
      include-profile-keyword     = 'include-profile'
      max-revision-keyword        = 'max-revision'
      min-revision-keyword        = 'min-revision'
      package-keyword             = 'package'
      profile-keyword             = 'profile'
      require-capability-keyword  = 'require-capability'
      require-module-keyword      = 'require-module'
      require-feature-keyword     = 'require-feature'
```

```
      require-package-keyword     = 'require-package'
      require-profile-keyword     = 'require-profile'
      subcategory-keyword         = 'subcategory'

      ;; all other symbols are defined in RFC 6020, section 12
```

<CODE ENDS>

8. IANA Considerations

TBD

9. Security Considerations

TBD

10. Open Issues

- How can logical OR expressions be supported for modules, features, and capabilities? E.g. a "writable-server" profile will need a require-capability for the :writable-running or for the :candidate capabilities. A server will usually advertise one of these 2 capabilities, but not both of them.

- Is some sort of "choice-stmt" needed within a profile-stmt, to indicate that 1 of N cases of requirements must be supported? Does the server need to advertise the selected cases in every choice it supports?

- Can a server support multiple conformance profiles at once? If so, then multiple profiles per package would need to be advertised in the YANG package capability exchange.

- What if a server is configurable to support different profiles from a given package? Does the server advertise only the configured profile? What if no profile has been configured yet? Is a default conformance profile per YANG package needed?

- Should Category be removed? If not, how much structure does it need? How many category statements should be allowed per package? Do conformance profiles need category statements?

- What package and profile naming conventions are needed? Do these identifiers need to share the same namespace as YANG modules?

- Package file name layout conventions are probably needed like those in RFC 6020, section 5.2. A file extension for a YANG package is needed.

- Is a media type for a YANG package needed similar to the definition in RFC 6020, section 14.1?

- Is a statement needed to specify acceptable parameter values for require-capability statements? E.g., a profile that required that the "report-all-tagged" enumeration be supported in the "also-supported" parameter for the ":with-defaults" capability.

11. Change Log

-- RFC Ed.: remove this section before publication.

11.1. 00-01

- fixed typos in text and examples
- updated ietf-routing-pkg example
- added ’require parameters for capabilities’ as an open issue

12. Normative References

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

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

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

Author's Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com