Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 14, 2015

A. Bierman
YumaWorks

June 12, 2015

Guidelines for Authors and Reviewers of YANG Data Model Documents
draft-ietf-netmod-rfc6087bis-03

Abstract

This memo provides guidelines for authors and reviewers of Standards Track specifications containing YANG data model modules. Applicable portions may be used as a basis for reviews of other YANG data model documents. Recommendations and procedures are defined, which are intended to increase interoperability and usability of Network Configuration Protocol (NETCONF) implementations that utilize 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 December 14, 2015.

Table of Contents

1. Introduction

The standardization of network configuration interfaces for use with the Network Configuration Protocol [RFC6241] requires a modular set of data models, which can be reused and extended over time.

This document defines a set of usage guidelines for Standards Track documents containing [RFC6020] data models. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF server. A server that supports a particular YANG module will support client NETCONF operation requests, as indicated by the specific content defined in the YANG module.

This document is similar to the Structure of Management Information version 2 (SMIv2) usage guidelines specification [RFC4181] in intent and structure. However, since that document was written a decade after SMIv2 modules had been in use, it was published as a 'Best Current Practice' (BCP). This document is not a BCP, but rather an informational reference, intended to promote consistency in documents containing YANG modules.

Many YANG constructs are defined as optional to use, such as the description statement. However, in order to maximize interoperability of NETCONF implementations utilizing YANG data models, it is desirable to define a set of usage guidelines that may require a higher level of compliance than the minimum level defined in the YANG specification.

In addition, YANG allows constructs such as infinite length identifiers and string values, or top-level mandatory nodes, that a compliant server is not required to support. Only constructs that all servers are required to support can be used in IETF YANG modules.

This document defines usage guidelines related to the NETCONF operations layer and NETCONF content layer, as defined in [RFC6241]. These guidelines are intended to be used by authors and reviewers to improve the readability and interoperability of published YANG data models.

Note that this document is not a YANG tutorial and the reader is expected to know the YANG data modeling language before using this document.

2. Terminology

2.1. Requirements Notation

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

RFC 2119 language is used here to express the views of the NETMOD working group regarding content for YANG modules. YANG modules complying with this document will treat the RFC 2119 terminology as if it were describing best current practices.

2.2. NETCONF Terms

The following terms are defined in [RFC6241] and are not redefined here:

- capabilities
- client
- operation
- server

2.3. YANG Terms

The following terms are defined in [RFC6020] and are not redefined here:

- data node
- module
- namespace
- submodule
- version
- YANG
- YIN

Note that the term 'module' may be used as a generic term for a YANG module or submodule. When describing properties that are specific to submodules, the term 'submodule' is used instead.

2.4. Terms

The following terms are used throughout this document:

- published: A stable release of a module or submodule, usually contained in an RFC.

- unpublished: An unstable release of a module or submodule, usually contained in an Internet-Draft.

3. YANG Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module to help readers understand the module structure.

The meaning of the symbols in YANG tree diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

4. General Documentation Guidelines

YANG data model modules under review are likely to be contained in Internet-Drafts. All guidelines for Internet-Draft authors MUST be followed. The RFC Editor provides guidelines for authors of RFCs, which are first published as Internet-Drafts. These guidelines should be followed and are defined in [RFC2223] and updated in [RFC5741] and "RFC Document Style" [RFC-STYLE].

The following sections MUST be present in an Internet-Draft containing a module:

- Narrative sections
- Definitions section
- Security Considerations section
- IANA Considerations section
- References section

4.1. Module Copyright

The module description statement MUST contain a reference to the latest approved IETF Trust Copyright statement, which is available online at:

http://trustee.ietf.org/license-info/

Each YANG module or submodule contained within an Internet-Draft or RFC is considered to be a code component. The strings "<CODE BEGINS>" and "<CODE ENDS>" MUST be used to identify each code component.

The "<CODE BEGINS>" tag SHOULD be followed by a string identifying the file name specified in Section 5.2 of [RFC6020]. The following example is for the '2010-01-18' revision of the 'ietf-foo' module:

```
<CODE BEGINS> file "ietf-foo@2010-01-18.yang"
  module ietf-foo {
    // ...
    revision 2010-01-18 {
      description "Latest revision";
      reference "RFC XXXX";
    }
    // ...
  }
```

<CODE ENDS>

Note that this convention MUST NOT be used for example modules or module fragments.

4.2. Terminology Section

A terminology section MUST be present if any terms are defined in the document or if any terms are imported from other documents.

If YANG tree diagrams are used, then a sub-section explaining the YANG tree diagram syntax MUST be present, containing the following text:

> A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is defined in [RFCXXXX].
>
> -- RFC Editor: Replace XXXX with RFC number and remove note

4.3. Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module, and SHOULD be included to help readers understand YANG module structure. Tree diagrams MAY be split into sections to correspond to document structure.

The following example shows a simple YANG tree diagram:

```
  +--rw top-level-config-container
  |  +--rw config-list* [key-name]
  |     +--rw key-name                    string
  |     +--rw optional-parm?              string
  |     +--rw mandatory-parm              identityref
  |     +--ro read-only-leaf              string
  +--ro top-level-nonconfig-container
     +--ro nonconfig-list* [name]
        +--ro name               string
        +--ro type               string
```

4.4. Narrative Sections

The narrative part MUST include an overview section that describes the scope and field of application of the module(s) defined by the specification and that specifies the relationship (if any) of these modules to other standards, particularly to standards containing other YANG modules. The narrative part SHOULD include one or more sections to briefly describe the structure of the modules defined in

the specification.

If the module(s) defined by the specification imports definitions from other modules (except for those defined in the [RFC6020] or [RFC6991] documents), or are always implemented in conjunction with other modules, then those facts MUST be noted in the overview section, as MUST be noted any special interpretations of definitions in other modules.

4.5. Definitions Section

This section contains the module(s) defined by the specification. These modules MUST be written using the YANG syntax defined in [RFC6020]. A YIN syntax version of the module MAY also be present in the document. There MAY also be other types of modules present in the document, such as SMIv2, which are not affected by these guidelines.

See Section 5 for guidelines on YANG usage.

4.6. Security Considerations Section

Each specification that defines one or more modules MUST contain a section that discusses security considerations relevant to those modules.

This section MUST be patterned after the latest approved template (available at http://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines). Section 7.1 contains the security considerations template dated 2013-05-08. Authors MUST check the webpage at the URL listed above in case there is a more recent version available.

In particular:

- Writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be explained.

- Readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

- Operations (i.e., YANG 'rpc' statements) that are potentially harmful to system behavior or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

4.7. IANA Considerations Section

In order to comply with IESG policy as set forth in http://www.ietf.org/id-info/checklist.html, every Internet-Draft that is submitted to the IESG for publication MUST contain an IANA Considerations section. The requirements for this section vary depending on what actions are required of the IANA. If there are no IANA considerations applicable to the document, then the IANA Considerations section stating that there are no actions is removed by the RFC Editor before publication. Refer to the guidelines in [RFC5226] for more details.

4.7.1. Documents that Create a New Namespace

If an Internet-Draft defines a new namespace that is to be administered by the IANA, then the document MUST include an IANA Considerations section that specifies how the namespace is to be administered.

Specifically, if any YANG module namespace statement value contained in the document is not already registered with IANA, then a new YANG Namespace registry entry MUST be requested from the IANA. The [RFC6020] specification includes the procedure for this purpose in its IANA Considerations section.

4.7.2. Documents that Extend an Existing Namespace

It is possible to extend an existing namespace using a YANG submodule that belongs to an existing module already administered by IANA. In this case, the document containing the main module MUST be updated to use the latest revision of the submodule.

4.8. Reference Sections

For every import or include statement that appears in a module contained in the specification, which identifies a module in a separate document, a corresponding normative reference to that document MUST appear in the Normative References section. The reference MUST correspond to the specific module version actually used within the specification.

For every normative reference statement that appears in a module contained in the specification, which identifies a separate document, a corresponding normative reference to that document SHOULD appear in the Normative References section. The reference SHOULD correspond to the specific document version actually used within the specification. If the reference statement identifies an informative reference, which identifies a separate document, a corresponding informative reference

to that document MAY appear in the Informative References section.

4.9. Validation Tools

All modules need to be validated before submission in an Internet Draft. The 'pyang' YANG compiler is freely available from github:

https://github.com/mbj4668/pyang

If the 'pyang' compiler is used, then the "--ietf" command line option SHOULD be used to identify any IETF guideline issues.

4.10. Module Extraction Tools

A version of 'rfcstrip' is available which will extract YANG modules from an Internet Draft or RFC. The 'rfcstrip' tool which supports YANG module extraction is freely available:

http://www.yang-central.org/twiki/pub/Main/YangTools/rfcstrip

This tool can be used to verify that the "<CODE BEGINS>" and "<CODE ENDS>" tags are used correctly and that the normative YANG modules can be extracted correctly.

5. YANG Usage Guidelines

In general, modules in IETF Standards Track specifications MUST comply with all syntactic and semantic requirements of YANG [RFC6020]. The guidelines in this section are intended to supplement the YANG specification, which is intended to define a minimum set of conformance requirements.

In order to promote interoperability and establish a set of practices based on previous experience, the following sections establish usage guidelines for specific YANG constructs.

Only guidelines that clarify or restrict the minimum conformance requirements are included here.

5.1. Module Naming Conventions

Modules contained in Standards Track documents SHOULD be named according to the guidelines in the IANA Considerations section of [RFC6020].

A distinctive word or acronym (e.g., protocol name or working group acronym) SHOULD be used in the module name. If new definitions are being defined to extend one or more existing modules, then the same word or acronym should be reused, instead of creating a new one.

All published module names MUST be unique. For a YANG module published in an RFC, this uniqueness is guaranteed by IANA. For unpublished modules, the authors need to check that no other work in progress is using the same module name.

Once a module name is published, it MUST NOT be reused, even if the RFC containing the module is reclassified to 'Historic' status.

5.2. Prefixes

All YANG definitions are scoped by the module containing the definition being referenced. This allows definitions from multiple modules to be used, even if the names are not unique. In the example below, the identifier "foo" is used in all 3 modules:

```
       module example-foo {
         namespace "http://example.com/ns/foo";
         prefix f;

         container foo;
       }

       module example-bar {
         namespace "http://example.com/ns/bar";
         prefix b;

         typedef foo { type uint32; }
       }

       module example-one {
         namespace "http://example.com/ns/one";
         prefix one;
         import example-foo { prefix f; }
         import example-bar { prefix b; }

         augment "/f:foo" {
            leaf foo { type b:foo; }
         }
       }
```

YANG defines the following rules for prefix usage:

- Prefixes are never allowed for built in data types and YANG keywords.

- A prefix MUST be used for any external statement (i.e., a statement defined with the YANG "extension" statement)

- The proper module prefix MUST be used for all identifiers imported from other modules

- The proper module prefix MUST be used for all identifiers included from a submodule.

The following guidelines apply to prefix usage of the current (local) module:

- The local module prefix SHOULD be used instead of no prefix in all path expressions.

- The local module prefix MUST be used instead of no prefix in all "default" statements for an "identityref" or "instance-identifier" data type

- The lcaol module prefix MAY be used for references to typedefs, groupings, extensions, features, and identities defined in the module.

5.3. Identifiers

Identifiers for all YANG identifiers in published modules MUST be between 1 and 64 characters in length. These include any construct specified as an 'identifier-arg-str' token in the ABNF in Section 12 of [RFC6020].

5.3.1. Identifier Naming Conventions

Identifiers SHOULD follow a consistent naming pattern throughout the module. Only lower-case letters, numbers, and dashes SHOULD be used in identifier names. Upper-case characters and the underscore character MAY be used if the identifier represents a well-known value that uses these characters.

Identifiers SHOULD include complete words and/or well-known acronyms or abbreviations. Child nodes within a container or list SHOULD NOT replicate the parent identifier. YANG identifiers are hierarchical and are only meant to be unique within the the set of sibling nodes defined in the same module namespace.

It is permissible to use common identifiers such as "name" or "id" in data definition statements, especially if these data nodes share a common data type.

Identifiers SHOULD NOT carry any special semantics that identify data modelling properties. Only YANG statements and YANG extension statements are designed to convey machine readable data modelling properties. For example, naming an object "config" or "state" does not change whether it is configuration data or state data. Only defined YANG statements or YANG extension statements can be used to assign semantics in a machine readable format in YANG.

5.4. Defaults

In general, it is suggested that substatements containing very common default values SHOULD NOT be present. The following substatements are commonly used with the default value, which would make the module difficult to read if used everywhere they are allowed.

Statement	Default Value
config	true
mandatory	false
max-elements	unbounded
min-elements	0
ordered-by	system
status	current
yin-element	false

Statement Defaults

5.5. Conditional Statements

A module may be conceptually partitioned in several ways, using the 'if-feature' and/or 'when' statements.

Data model designers need to carefully consider all modularity aspects, including the use of YANG conditional statements.

If a data definition is optional, depending on server support for a NETCONF protocol capability, then a YANG 'feature' statement SHOULD be defined to indicate that the NETCONF capability is supported within the data model.

If any notification data, or any data definition, for a non-configuration data node is not mandatory, then the server may or may not be required to return an instance of this data node. If any conditional requirements exist for returning the data node in a notification payload or retrieval request, they MUST be documented somewhere. For example, a 'when' or 'if-feature' statement could apply to the data node, or the conditional requirements could be explained in a 'description' statement within the data node or one of its ancestors (if any).

If any 'if-feature' statements apply to a list node, then the same 'if-feature' statements MUST apply to any key leaf nodes for the list. There MUST NOT be any 'if-feature' statements applied to any key leaf that do not also apply to the parent list node.

There SHOULD NOT be any 'when' statements applied to a key leaf node. It is possible that a 'when' statement for an ancestor node of a key leaf will have the exact node-set result as the key leaf. In such a case, the 'when' statement for the key leaf is redundant and SHOULD be avoided.

5.6. XPath Usage

This section describes guidelines for using the XML Path Language [W3C.REC-xpath-19991116] (XPath) within YANG modules.

5.6.1. XPath Evaluation Contexts

YANG defines 5 separate contexts for evaluation of XPath statements:

1) The "running" datastore: collection of all YANG configuration data nodes. The document root is the conceptual container, (e.g., "config" in the "edit-config" operation), which is the parent of all top-level data definition statements with a "config" statement value of "true".

2) State data + the "running" datastore: collection of all YANG data nodes. The document root is the conceptual container, parent of all top-level data definition statements.

3) Notification: an event notification document. The document root is the notification element.

4) RPC Input: The document root is the conceptual "input" node, which is the parent of all RPC input parameter definitions.

5) RPC Output: The document root is the conceptual "output" node, which is the parent of all RPC output parameter definitions.

Note that these XPath contexts cannot be mixed. For example, a "when" statement in a notification context cannot reference configuration data.

```
    notification foo {
      leaf mtu {
        // NOT OK because when-stmt context is this notification
        when "/if:interfaces/if:interface[name='eth0']";
        type leafref {
          // OK because path-stmt has a different context
          path "/if:interfaces/if:interface/if:mtu";
        }
      }
    }
```

It is especially important to consider the XPath evaluation context for XPath expressions defined in groupings. An XPath expression defined in a grouping may not be portable, meaning it cannot be used in multiple contexts and produce proper results.

If the XPath expressions defined in a grouping are intended for a particular context, then this context SHOULD be identified in the "description" statement for the grouping.

5.6.2. Function Library

The 'position' and 'last' functions SHOULD NOT be used. This applies to implicit use of the 'position' function as well (e.g., '//chapter[42]'). A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'position' and 'last' functions MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

The 'id' function SHOULD NOT be used. The 'ID' attribute is not present in YANG documents so this function has no meaning. The YANG compiler SHOULD return an empty string for this function.

The 'namespace-uri' and 'name' functions SHOULD NOT be used. Expanded names in XPath are different than YANG. A specific canonical representation of a YANG expanded name does not exist.

The 'lang' function SHOULD NOT be used. This function does not apply to YANG because there is no 'lang' attribute set with the document. The YANG compiler SHOULD return 'false' for this function.

The 'local-name', 'namespace-uri', 'name', 'string', and 'number' functions SHOULD NOT be used if the argument is a node-set. If so, the function result will be determined by the document order of the node-set. Since this order can be different on each server, the function results can also be different. Any function call that implicitly converts a node-set to a string will also have this issue.

The 'local-name' function SHOULD NOT be used to reference local names outside of the YANG module defining the must or when expression containing the 'local-name' function. Example of a local-name function that should not be used:

```
/*[local-name()='foo']
```

5.6.3. Axes

The 'attribute' and 'namespace' axes are not supported in YANG, and MAY be empty in a NETCONF server implementation.

The 'preceding', and 'following' axes SHOULD NOT be used. These constructs rely on XML document order within a NETCONF server configuration database, which may not be supported consistently or

produce reliable results across implementations. Predicate expressions based on static node properties (e.g., element name or value, 'ancestor' or 'descendant' axes) SHOULD be used instead. The 'preceding' and 'following' axes MAY be used if document order is not relevant to the outcome of the expression (e.g., check for global uniqueness of a parameter value).

The 'preceding-sibling' and 'following-sibling' axes SHOULD NOT used, however they MAY be used if document order is not relevant to the outcome of the expression.

A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'preceding-sibling' and 'following-sibling' axes MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

5.6.4. Types

Data nodes that use the 'int64' and 'uint64' built-in type SHOULD NOT be used within numeric or boolean expressions. There are boundary conditions in which the translation from the YANG 64-bit type to an XPath number can cause incorrect results. Specifically, an XPath 'double' precision floating point number cannot represent very large positive or negative 64-bit numbers because it only provides a total precision of 53 bits. The 'int64' and 'uint64' data types MAY be used in numeric expressions if the value can be represented with no more than 53 bits of precision.

Data modelers need to be careful not to confuse the YANG value space and the XPath value space. The data types are not the same in both, and conversion between YANG and XPath data types SHOULD be considered carefully.

Explicit XPath data type conversions MAY be used (e.g., 'string', 'boolean', or 'number' functions), instead of implicit XPath data type conversions.

XPath expressions that contain a literal value representing a YANG identity SHOULD always include the declared prefix of the module where the identity is defined.

XPath expressions for 'when' statements SHOULD NOT reference the context node or any descendant nodes of the context node. They MAY reference descendant nodes if the 'when' statement is contained within an 'augment' statement, and the referenced nodes are not defined within the 'augment' statement.

Example:

```
augment "/rt:active-route/rt:input/rt:destination-address" {
   when "rt:address-family='v4ur:ipv4-unicast'" {
     description
       "This augment is valid only for IPv4 unicast.";
   }
   // nodes defined here within the augment-stmt
   // cannot be referenced in the when-stmt
}
```

5.6.5. Wildcards

It is possible to construct XPath expressions that will evaluate differently when combined with several modules within a server implementation, then when evaluated within the single module. This is due to augmenting nodes from other modules.

Wildcard expansion is done within a server against all the nodes from all namespaces, so it is possible for a 'must' or 'when' expression that uses the '*' operator will always evaluate to false if processed within a single YANG module. In such cases, the 'description' statement SHOULD clarify that augmenting objects are expected to match the wildcard expansion.

```
when /foo/services/*/active {
  description
    "No services directly defined in this module.
     Matches objects that have augmented the services container.";
}
```

5.6.6. Boolean Expressions

The YANG "must" and "when" statements use an XPath boolean expression to define the test condition for the statement. It is important to specify these expressions in a way that will not cause inadvertent changes in the result if the objects referenced in the expression are updated in future revisions of the module.

For example, the leaf "foo2" must exist if the leaf "foo1" is equal to "one" or "three":

```
leaf foo1 {
  type enumeration {
     enum one;
     enum two;
     enum three;
  }
}

leaf foo2 {
  // INCORRECT
  must "/f:foo1 != 'two'";
  type string;
}

leaf foo2 {
  // CORRECT
  must "/f:foo1 = 'one' or /f:foo1 = 'three'";
  type string;
}
```

In the next revision of the module, leaf "foo1" is extended with a nem enum named "four":

```
leaf foo1 {
  type enumeration {
     enum one;
     enum two;
     enum three;
     enum four;
  }
}
```

Now the first XPath expression will allow the enum "four" to be accepted in addition to the "one" and "three" enum values.

5.7. Lifecycle Management

The status statement MUST be present if its value is 'deprecated' or 'obsolete'. The status SHOULD NOT be changed from 'current' directly to 'obsolete'. An object SHOULD be available for at least one year with 'deprecated' status before it is changed to 'obsolete'.

The module or submodule name MUST NOT be changed, once the document containing the module or submodule is published.

The module namespace URI value MUST NOT be changed, once the document containing the module is published.

The revision-date substatement within the imports statement SHOULD be present if any groupings are used from the external module.

The revision-date substatement within the include statement SHOULD be present if any groupings are used from the external submodule.

If submodules are used, then the document containing the main module MUST be updated so that the main module revision date is equal or more recent than the revision date of any submodule that is (directly or indirectly) included by the main module.

5.8. Module Header, Meta, and Revision Statements

For published modules, the namespace MUST be a globally unique URI, as defined in [RFC3986]. This value is usually assigned by the IANA.

The organization statement MUST be present. If the module is contained in a document intended for Standards Track status, then the organization SHOULD be the IETF working group chartered to write the document.

The contact statement MUST be present. If the module is contained in a document intended for Standards Track status, then the working group web and mailing information MUST be present, and the main document author or editor contact information SHOULD be present. If additional authors or editors exist, their contact information MAY be present. In addition, the Area Director and other contact information MAY be present.

The description statement MUST be present. The appropriate IETF Trust Copyright text MUST be present, as described in Section 4.1.

If the module relies on information contained in other documents, which are not the same documents implied by the import statements present in the module, then these documents MUST be identified in the reference statement.

A revision statement MUST be present for each published version of the module. The revision statement MUST have a reference substatement. It MUST identify the published document that contains the module. Modules are often extracted from their original documents, and it is useful for developers and operators to know how to find the original source document in a consistent manner. The revision statement MAY have a description substatement.

Each new revision MUST include a revision date that is higher than any other revision date in the module. The revision date does not need to be updated if the module contents do not change in the new

document revision.

It is acceptable to reuse the same revision statement within unpublished versions (i.e., Internet-Drafts), but the revision date MUST be updated to a higher value each time the Internet-Draft is re-posted.

5.9. Namespace Assignments

It is RECOMMENDED that only valid YANG modules be included in documents, whether or not they are published yet. This allows:

- the module to compile correctly instead of generating disruptive fatal errors.
- early implementors to use the modules without picking a random value for the XML namespace.
- early interoperability testing since independent implementations will use the same XML namespace value.

Until a URI is assigned by the IANA, a proposed namespace URI MUST be provided for the namespace statement in a YANG module. A value SHOULD be selected that is not likely to collide with other YANG namespaces. Standard module names, prefixes, and URI strings already listed in the YANG Module Registry MUST NOT be used.

A standard namespace statement value SHOULD have the following form:

```
<URN prefix string>:<module-name>
```

The following URN prefix string SHOULD be used for published and unpublished YANG modules:

```
urn:ietf:params:xml:ns:yang:
```

The following example URNs would be valid namespace statement values for Standards Track modules:

```
urn:ietf:params:xml:ns:yang:ietf-netconf-partial-lock

urn:ietf:params:xml:ns:yang:ietf-netconf-state

urn:ietf:params:xml:ns:yang:ietf-netconf
```

Note that a different URN prefix string SHOULD be used for non-Standards-Track modules. The string SHOULD be selected according to the guidelines in [RFC6020].

The following examples are for non-Standards-Track modules. The domain "example.com" SHOULD be used in all namespace URIs for example modules.

http://example.com/ns/example-interfaces

http://example.com/ns/example-system

5.10. Top-Level Data Definitions

The top-level data organization SHOULD be considered carefully, in advance. Data model designers need to consider how the functionality for a given protocol or protocol family will grow over time.

The separation of configuration data and operational state SHOULD be considered carefully. It is often useful to define separate top-level containers for configuration and non-configuration data.

The number of top-level data nodes within a module SHOULD be minimized. It is often useful to retrieve related information within a single subtree. If data is too distributed, is becomes difficult to retrieve all at once.

The names and data organization SHOULD reflect persistent information, such as the name of a protocol. The name of the working group SHOULD NOT be used because this may change over time.

A mandatory database data definition is defined as a node that a client must provide for the database to be valid. The server is not required to provide a value.

Top-level database data definitions MUST NOT be mandatory. If a mandatory node appears at the top level, it will immediately cause the database to be invalid. This can occur when the server boots or when a module is loaded dynamically at runtime.

5.11. Data Types

Selection of an appropriate data type (i.e., built-in type, existing derived type, or new derived type) is very subjective, and therefore few requirements can be specified on that subject.

Data model designers SHOULD use the most appropriate built-in data type for the particular application.

The signed numeric data types (i.e., 'int8', 'int16', 'int32', and 'int64') SHOULD NOT be used unless negative values are allowed for the desired semantics.

5.11.1. Fixed Value Extensibility

If the set of values is fixed and the data type contents are controlled by a single naming authority, then an enumeration data type SHOULD be used.

```
leaf foo {
  type enumeration {
    enum one;
    enum two;
  }
}
```

If extensibility of enumerated values is required, then the 'identityref' data type SHOULD be used instead of an enumeration or other built-in type.

```
identity foo-type {
  description "Base for the extensible type";
}

identity one {
  base f:foo-type;
}
identity two {
  base f:foo-type;
}

leaf foo {
  type identityref {
    base f:foo-type;
  }
}
```

Note that any module can declare an identity with base "foo-type" that is valid for the "foo" leaf. Identityref values are considered to be qualified names.

5.11.2. Patterns and Ranges

For string data types, if a machine-readable pattern can be defined for the desired semantics, then one or more pattern statements SHOULD be present. A single quoted string SHOULD be used to specify the pattern, since a double-quoted string can modify the content.

The following typedef from [RFC6991] demonstrates the proper use of the "pattern" statement:

```
typedef ipv4-address-no-zone {
  type inet:ipv4-address {
    pattern '[0-9\.]*';
  }
  ...
}
```

For string data types, if the length of the string is required to be bounded in all implementations, then a length statement MUST be present.

The following typedef from [RFC6991] demonstrates the proper use of the "length" statement:

```
typedef yang-identifier {
  type string {
    length "1..max";
    pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
    pattern '.|..|[^xX].*|.[^mM].*|..[^lL].*';
  }
  ...
}
```

For numeric data types, if the values allowed by the intended semantics are different than those allowed by the unbounded intrinsic data type (e.g., 'int32'), then a range statement SHOULD be present.

The following typedef from [RFC6991] demonstrates the proper use of the "range" statement:

```
typedef dscp {
  type uint8 {
     range "0..63";
  }
  ...
}
```

5.11.3. Enumerations and Bits

For 'enumeration' or 'bits' data types, the semantics for each 'enum' or 'bit' SHOULD be documented. A separate description statement (within each 'enum' or 'bit' statement) SHOULD be present.

```
leaf foo {
  // INCORRECT
  type enumeration {
    enum one;
    enum two;
  }
  description
    "The foo enum...
     one: The first enum
     two: The second enum";
}

leaf foo {
  // CORRECT
  type enumeration {
    enum one {
      description "The first enum";
    }
    enum two {
      description "The second enum";
    }
  }
  description
    "The foo enum...  ";
}
```

5.12. Reusable Type Definitions

If an appropriate derived type exists in any standard module, such as [RFC6991], then it SHOULD be used instead of defining a new derived type.

If an appropriate units identifier can be associated with the desired semantics, then a units statement SHOULD be present.

If an appropriate default value can be associated with the desired semantics, then a default statement SHOULD be present.

If a significant number of derived types are defined, and it is anticipated that these data types will be reused by multiple modules, then these derived types SHOULD be contained in a separate module or submodule, to allow easier reuse without unnecessary coupling.

The description statement MUST be present.

If the type definition semantics are defined in an external document (other than another YANG module indicated by an import statement), then the reference statement MUST be present.

5.13. Data Definitions

The description statement MUST be present in the following YANG statements:

- anyxml
- augment
- choice
- container
- extension
- feature
- grouping
- identity
- leaf
- leaf-list
- list
- notification
- rpc
- typedef

If the data definition semantics are defined in an external document, (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

The 'anyxml' construct may be useful to represent an HTML banner containing markup elements, such as '' and '', and MAY be used in such cases. However, this construct SHOULD NOT be used if other YANG data node types can be used instead to represent the desired syntax and semantics.

It has been found that the 'anyxml' statement is not implemented consistently across all servers. It is possible that mixed mode XML will not be supported, or configuration anyxml nodes will not supported.

If there are referential integrity constraints associated with the desired semantics that can be represented with XPath, then one or more 'must' statements SHOULD be present.

For list and leaf-list data definitions, if the number of possible instances is required to be bounded for all implementations, then the max-elements statements SHOULD be present.

If any 'must' or 'when' statements are used within the data definition, then the data definition description statement SHOULD describe the purpose of each one.

5.14. Operation Definitions

If the operation semantics are defined in an external document (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

If the operation impacts system behavior in some way, it SHOULD be mentioned in the description statement.

If the operation is potentially harmful to system behavior in some way, it MUST be mentioned in the Security Considerations section of the document.

5.15. Notification Definitions

The description statement MUST be present.

If the notification semantics are defined in an external document (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

If the notification refers to a specific resource instance, then this instance SHOULD be identified in the notification data. This is usually done by including 'leafref' leaf nodes with the key leaf values for the resource instance. For example:

```
  notification interface-up {
    description "Sent when an interface is activated.";
    leaf name {
      type leafref {
        path "/if:interfaces/if:interface/if:name";
      }
    }
  }
```

Note that there are no formal YANG statements to identify any data

node resources associated with a notification. The description statement for the notification SHOULD specify if and how the notification identifies any data node resources associated with the specific event.

5.16. Feature Definitions

The YANG "feature" statement is used to define a label for a set of optional functionality within a module. The "if-feature" statement is used in the YANG statements associated with a feature.

The set of YANG features available in a module should be considered carefully. The description-stmt within a feature-stmt MUST specify any interactions with other features.

If there is a large set of objects associated with a YANG feature, then consider moving those objects to a separate module, instead of using a YANG feature. Note that the set of features within a module is easily discovered by the reader, but the set of related modules within the entire YANG library is not as easy to identity. Module names with a common prefix can help readers identity the set of related modules, but this assumes the reader will have discovered and installed all the relevant modules.

Another consideration for deciding whether to create a new module or add a YANG feature is the stability of the module in question. It may be desirable to have a stable base module that is not changed frequently. If new functionality is placed in a separate module, then the base module does not need to be republished. If it is designed as a YANG feature then the module will need to be republished.

If one feature requires implementation of another feature, then an "if-feature" statement SHOULD be used in the dependent "feature" statement.

For example, feature2 requires implementation of feature1:

```
feature feature1 {
  description "Some protocol feature";
}

feature feature2 {
  if-feature "feature1";
  description "Another protocol feature";
}
```

5.17. Augment Statements

The YANG "augment" statement is used to define a set of data definition statements that will be added as child nodes of a target data node. The module namespace for these data nodes will be the augmenting module, not the augmented module.

A top-level "augment" statement SHOULD NOT be used if the target data node is in the same module or submodule as the evaluated "augment" statement. The data definition statements SHOULD be added inline instead.

5.17.1. Conditional Augment Statements

The "augment" statement is often used together with the "when" statement and/or "if-feature" statement to make the augmentation conditional on some portion of the data model.

The following example from [RFC7223] shows how a conditional container called "ethernet" is added to the "interface" list only for entries of the type "ethernetCsmacd".

```
augment "/if:interfaces/if:interface" {
    when "if:type = 'ianaift:ethernetCsmacd'";

    container ethernet {
        leaf duplex {
            ...
        }
    }
}
```

5.17.2. Conditionally Mandatory Data Definition Statements

YANG has very specific rules about how configuration data can be updated in new releases of a module. These rules allow an "old client" to continue interoperating with a "new server".

If data nodes are added to an existing entry, the old client MUST NOT be required to provide any mandatory parameters that were not in the original module definition.

It is possible to add conditional augment statements such that the old client would not know about the new condition, and would not specify the new condition. The conditional augment statement can contain mandatory objects only if the condition is false unless explicitly requested by the client.

Only a conditional augment statement that uses the "when" statement form of condition can be used in this manner. The YANG features enabled on the server cannot be controlled by the client in any way, so it is not safe to add mandatory augmenting data nodes based on the "if-feature" statement.

The XPath "when" statement condition MUST NOT reference data outside of target data node because the client does not have any control over this external data.

In the following dummy example, it is OK to augment the "interface" entry with "mandatory-leaf" because the augmentation depends on support for "some-new-iftype". The old client does not know about this type so it would never select this type, and therefore not be adding a mandatory data node.

```
  module my-module {
    ...

    identity some-new-iftype {
       base iana:iana-interface-type;
    }

    augment "/if:interfaces/if:interface" {
       when "if:type = 'mymod:some-new-iftype'";

       leaf mandatory-leaf {
          mandatory true;
          ...
       }
    }
  }
```

Note that this practice is safe only for creating data resources. It is not safe for replacing or modifying resources if the client does not know about the new condition. The YANG data model MUST be packaged in a way that requires the client to be aware of the mandatory data nodes if it is aware of the condition for this data. In the example above, the "some-new-iftype" identity is defined in the same module as the "mandatory-leaf" data definition statement.

This practice is not safe for identities defined in a common module such as "iana-if-type" because the client is not required to know about "my-module" just because it knows about the "iana-if-type" module.

5.18. Data Correlation

Data can be correlated in various ways, using common data types, common data naming, and common data organization. There are several ways to extend the functionality of a module, based on the degree of coupling between the old and new functionality:

- inline: update the module with new protocol-accessible objects. The naming and data organization of the original objects is used. The new objects are in the original module namespace.

- augment: create a new module with new protocol-accessible objects that augment the original data structure. The naming and data organization of the original objects is used. The new objects are in the new module namespace.

- mirror: create new objects in a new module or the original module, except use new a naming scheme and data location. The naming can be coupled in different ways. Tight coupling is achieved with a "leafref" data type, with the "require-instance" sub-statement set to "true". This method SHOULD be used.

If the new data instances are not limited to the values in use in the original data structure, then the "require-instance" sub-statement MUST be set to "false". Loose coupling is achieved by using key leafs with the same data type as the original data structure. This has the same semantics as setting the "require-instance" sub-statement to "false".

It is sometimes useful to separate configuration and operational state, so that they do not not even share the exact same naming characteristics. The correlation between configuration the operational state data that is affected by changes in configuration is a complex problem. There may not be a simple 1:1 relationship between a configuration data node and an operational data node. Further work is needed in YANG to clarify this relationship. Protocol work may also be needed to allow a client to retrieve this type of information from a server. At this time the best practice is to clearly document any relationship to other data structures in the "description" statement.

5.19. Operational State

In YANG, any data that has a "config" statement value of "false" could be considered operational state. The relationship between configuration (i.e., "config" statement has a value of "true") and operational state can be complex.

One challenge for client developers is determining if the configured value is being used, which requires the developer to know which operational state parameters are associated with the particular configuration object (or group of objects).

The simplest interaction between configuration and operational state is "none". For example, the arbitrary administrative name or sequence number assigned to an access control rule. The configured value is always the value that is being used by the system.

However, some configuration parameters interact with routing and other signalling protocols, such that the operational value in use by the system may not be the same as the configured value. Other parameters specify the desired state, but environmental and other factors can cause the actual state to be different.

For example a "temperature" configuration setting only represents the desired temperature. An operational state parameter is needed that reports the actual temperature in order to determine if the cooling system is operating correctly. YANG has no mechanism other than the "description" statement to associate the desired temperature and the actual temperature.

Careful consideration needs to be given to the location of operational state data. It can either be located within the configuration subtree for which it applies, or it can be located outside the particular configuration subtree. Placing operation state within the configuration subtree is appropriate if the operational values can only exist if the configuration exists.

The "interfaces" and "interfaces-state" subtrees defined in [RFC7223] are an example of a complex relationship between configuration and operational state. The operational values can include interface entries that have been discovered or initialized by the system. An interface may be in use that has not been configured at all. Therefore, the operational state for an interface cannot be located within the configuration for that same interface.

Sometimes the configured value represents some sort of procedure to be followed, in which the system will select an actual value, based on protocol negotiation.

```
    leaf duplex-admin-mode {
      type enumeration {
        enum auto;
        enum half;
        enum full;
      }
    }

    leaf duplex-oper-mode {
      config false;
      type enumeration {
        enum half;
        enum full;
      }
    }
```

For example a "duplex" mode configuration may be "auto" to auto-negotiate the actual value to be used. The operational parameter will never contain the value "auto". It will always contain the result of the auto-negotiation, such as "half" or "full". This is just one way in which the configuration data model is not exactly the same as the operational data model. Another is if the detailed properties of the data are different for configured vs. learned entries.

If all the data model properties are aligned between configuration and operational data, then it can be useful to define the configuration parameters within a grouping, and then replicate that grouping within the operational state portion of the data model.

```
    grouping parms {
       // do not use config-stmt in any of the nodes
       // placed in this grouping
    }

    container foo {
      uses parms;  // these are all config=true by default
      state {
        config false;  // only exists if foo config exists
        uses parms;
      }
    }
```

Note that this mechanism can also be used if the configuration and operational state data are in separate sub-trees:

```
container bar { // bar config can exist without bar-state
  config true;
  uses parms;
}

container bar-state {  // bar-state can exist without bar
  config false;
  uses parms;
}
```

The need to replicate objects or define different operational state objects depends on the data model. It is not possible to define one approach that will be optimal for all data models. Designers SHOULD describe the relationship in detail between configuration objects and any associated operational state objects. The "description" statements for both the configuration and the operational state SHOULD be used for this purpose.

6. IANA Considerations

This document registers one URI in the IETF XML registry [RFC3688].

The following registration has been made:

URI: urn:ietf:params:xml:ns:yang:ietf-template

Registrant Contact: The NETMOD WG of the IETF.

XML: N/A, the requested URI is an XML namespace.

Per this document, the following assignment has been made in the YANG Module Names Registry for the YANG module template in Appendix C.

Field	Value
Name	ietf-template
Namespace	urn:ietf:params:xml:ns:yang:ietf-template
Prefix	temp
Reference	RFC XXXX

YANG Registry Assignment

7. Security Considerations

This document defines documentation guidelines for NETCONF content defined with the YANG data modeling language. The guidelines for how to write a Security Considerations section for a YANG module are defined in the online document

http://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines

This document does not introduce any new or increased security risks into the management system.

The following section contains the security considerations template dated 2010-06-16. Be sure to check the webpage at the URL listed above in case there is a more recent version available.

Each specification that defines one or more YANG modules MUST contain a section that discusses security considerations relevant to those modules. This section MUST be patterned after the latest approved template (available at

http://www.ops.ietf.org/netconf/yang-security-considerations.txt).

In particular, writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be spelled out.

Similarly, readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

Further, if new RPC operations have been defined, then the security considerations of each new RPC operation MUST be explained.

7.1. Security Considerations Section Template

X. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242].

```
-- if you have any writable data nodes (those are all the
-- "config true" nodes, and remember, that is the default)
-- describe their specific sensitivity or vulnerability.
```

There are a number of data nodes defined in this YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

<list subtrees and data nodes and state why they are sensitive>

```
-- for all YANG modules you must evaluate whether any readable data
-- nodes (those are all the "config false" nodes, but also all other
-- nodes, because they can also be read via operations like get or
-- get-config) are sensitive or vulnerable (for instance, if they
-- might reveal customer information or violate personal privacy
-- laws such as those of the European Union if exposed to
-- unauthorized parties)
```

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

<list subtrees and data nodes and state why they are sensitive>

```
-- if your YANG module has defined any rpc operations
-- describe their specific sensitivity or vulnerability.
```

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

<list RPC operations and state why they are sensitive>

8. Acknowledgments

The structure and contents of this document are adapted from [RFC4181], guidelines for MIB Documents, by C. M. Heard.

The working group thanks Martin Bjorklund, Juergen Schoenwaelder, Ladislav Lhotka, and Jernej Tuljak for their extensive reviews and contributions to this document.

9. Changes Since RFC 6087

The following changes have been made to the guidelines published in [RFC6087]:

- Updated NETCONF reference from RFC 4741 to RFC 6241
- Updated NETCONF over SSH citation from RFC 4742 to RFC 6242
- Updated YANG Types reference from RFC 6021 to RFC 6991
- Updated obsolete URLs for IETF resources
- Changed top-level data node guideline
- Clarified XPath usage for a literal value representing a YANG identity
- Clarified XPath usage for a when-stmt
- Clarified XPath usage for 'proceeding-sibling' and 'following-sibling' axes
- Added terminology guidelines
- Added YANG tree diagram definition and guideline
- Updated XPath guidelines for type conversions and function library usage.
- Updated data types section
- Updated notifications section
- Clarified conditional key leaf nodes
- Clarify usage of 'uint64' and 'int64' data types
- Added text on YANG feature usage
- Added Identifier Naming Conventions
- Clarified use of mandatory nodes with conditional augmentations
- Clarified namespace and domain conventions for example modules

10. References

10.1. Normative References

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

[RFC2223] Postel, J. and J. Reynolds, "Instructions to RFC Authors", RFC 2223, October 1997.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, November 2008.

[RFC5741] Daigle, L., Kolkman, O., and IAB, "RFC Streams, Headers, and Boilerplates", RFC 5741, December 2009.

[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.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[W3C.REC-xpath-19991116] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

10.2. Informative References

[RFC-STYLE] Braden, R., Ginoza, S., and A. Hagens, "RFC Document Style", September 2009, <http://www.rfc-editor.org/rfc-style-guide/rfc-style>.

[RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB

Documents", BCP 111, RFC 4181, September 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, January 2011.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

Appendix A. Change Log

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

A.1. 02 to 03

- Updated draft based on github data tracker issues added by Benoit Clause (Issues 12 - 18)

A.2. 01 to 02

- Updated draft based on mailing list comments.

A.3. 00 to 01

All issues from the issue tracker have been addressed.

https://github.com/netmod-wg/rfc6087bis/issues

- Issue 1: Tree Diagrams: Added Section 3 so RFCs with YANG modules can use an Informative reference to this RFC for tree diagrams. Updated guidelines to reference this RFC when tree diagrams are used

- Issue 2: XPath function restrictions: Added paragraphs in XPath usage section for 'id', 'namespace-uri', 'name', and 'lang' functions

- Issue 3: XPath function document order issues: Added paragraph in XPath usage section about node-set ordering for 'local-name', 'namespace-uri', 'name', 'string' and 'number' functions. Also any function that implicitly converts a node-set to a string.

- Issue 4: XPath preceding-sibling and following-sibling: Checked and text in XPath usage section already has proposed text from Lada.

- Issue 5: XPath 'when-stmt' reference to descendant nodes: Added exception and example in XPath Usage section for augmented nodes.

- Issue 6: XPath numeric conversions: Changed 'numeric expressions' to 'numeric and boolean expressions'

- Issue 7: XPath module containment: Added sub-section on XPath wildcards

- Issue 8: status-stmt usage: Added text to Lifecycle Management section about transitioning from active to deprecated and then to

obsolete.

- Issue 9: resource identification in notifications: Add text to Notifications section about identifying resources and using the leafref data type.

- Issue 10: single quoted strings: Added text to Data Types section about using a single-quoted string for patterns.

Appendix B. Module Review Checklist

This section is adapted from RFC 4181.

The purpose of a YANG module review is to review the YANG module both for technical correctness and for adherence to IETF documentation requirements. The following checklist may be helpful when reviewing an Internet-Draft:

- I-D Boilerplate -- verify that the draft contains the required Internet-Draft boilerplate (see http://www.ietf.org/id-info/guidelines.html), including the appropriate statement to permit publication as an RFC, and that I-D boilerplate does not contain references or section numbers.

- Abstract -- verify that the abstract does not contain references, that it does not have a section number, and that its content follows the guidelines in http://www.ietf.org/id-info/guidelines.html.

- Copyright Notice -- verify that the draft has the appropriate text regarding the rights that document contributers provide to the IETF Trust [RFC5378]. Verify that it contains the full IETF Trust copyright notice at the beginning of the document. The IETF Trust Legal Provisions (TLP) can be found at:

  http://trustee.ietf.org/license-info/

- Security Considerations section -- verify that the draft uses the latest approved template from the OPS area website (http://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines) and that the guidelines therein have been followed.

- IANA Considerations section -- this section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:

  XML Namespace Registry: Register the YANG module namespace.

  YANG Module Registry: Register the YANG module name, prefix, namespace, and RFC number, according to the rules specified in [RFC6020].

- References -- verify that the references are properly divided between normative and informative references, that RFC 2119 is included as a normative reference if the terminology defined therein is used in the document, that all references required by the boilerplate are present, that all YANG modules containing imported items are cited as normative references, and that all citations point to the most current RFCs unless there is a valid reason to do otherwise (for example, it is OK to include an informative reference to a previous version of a specification to help explain a feature included for backward compatibility). Be sure citations for all imported modules are present somewhere in the document text (outside the YANG module).

- License -- verify that the draft contains the Simplified BSD License in each YANG module or submodule. Some guidelines related to this requirement are described in Section 4.1. Make sure that the correct year is used in all copyright dates. Use the approved text from the latest Trust Legal Provisions (TLP) document, which can be found at:

  http://trustee.ietf.org/license-info/

- Other Issues -- check for any issues mentioned in http://www.ietf.org/id-info/checklist.html that are not covered elsewhere.

- Technical Content -- review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information can be found at:

  http://trac.tools.ietf.org/wg/netconf/trac/wiki

  Checking for correct syntax, however, is only part of the job. It is just as important to actually read the YANG module document from the point of view of a potential implementor. It is particularly important to check that description statements are sufficiently clear and unambiguous to allow interoperable implementations to be created.

Appendix C. YANG Module Template

```
<CODE BEGINS> file "ietf-template@2010-05-18.yang"


  module ietf-template {

    // replace this string with a unique namespace URN value
    namespace
      "urn:ietf:params:xml:ns:yang:ietf-template";

    // replace this string, and try to pick a unique prefix
    prefix "temp";


    // import statements here: e.g.,
    // import ietf-yang-types { prefix yang; }
    // import ietf-inet-types { prefix inet; }

    // identify the IETF working group if applicable
    organization
       "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

    // update this contact statement with your info
    contact
       "WG Web:   <http://tools.ietf.org/wg/your-wg-name/>
        WG List:  <mailto:your-wg-name@ietf.org>

        WG Chair: your-WG-chair
                  <mailto:your-WG-chair@example.com>

        Editor:   your-name
                  <mailto:your-email@example.com>";


    // replace the first sentence in this description statement.
    // replace the copyright notice with the most recent
    // version, if it has been updated since the publication
    // of this document
    description
     "This module defines a template for other YANG modules.

      Copyright (c) <insert year> IETF Trust and the persons
      identified as authors of the code.  All rights reserved.

      Redistribution and use in source and binary forms, with or
      without modification, is permitted pursuant to, and subject
      to the license terms contained in, the Simplified BSD License
```

```
          set forth in Section 4.c of the IETF Trust's Legal Provisions
          Relating to IETF Documents
          (http://trustee.ietf.org/license-info).

          This version of this YANG module is part of RFC XXXX; see
          the RFC itself for full legal notices.";

        // RFC Ed.: replace XXXX with actual RFC number and remove
        // this note

        reference "RFC XXXX";

        // RFC Ed.: remove this note
        // Note: extracted from RFC XXXX


        // replace '2010-05-18' with the module publication date
        // The format is (year-month-day)
        revision "2010-05-18" {
          description
            "Initial version";
        }

        // extension statements

        // feature statements

        // identity statements

        // typedef statements

        // grouping statements

        // data definition statements

        // augment statements

        // rpc statements

        // notification statements

        // DO NOT put deviation statements in a published module

      }

    <CODE ENDS>
```

Author's Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com