

Internet Engineering Task Force (IETF)
Request for Comments: 8407
BCP: 216
Obsoletes: [6087](#)
Category: Best Current Practice
ISSN: 2070-1721

A. Bierman
YumaWorks
October 2018

Guidelines for Authors and Reviewers of Documents Containing YANG Data Models

Abstract

This memo provides guidelines for authors and reviewers of specifications containing YANG modules. Recommendations and procedures are defined, which are intended to increase interoperability and usability of Network Configuration Protocol (NETCONF) and RESTCONF protocol implementations that utilize YANG modules. This document obsoletes [RFC 6087](#).

Status of This Memo

This memo documents an Internet Best Current Practice.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on BCPs is available in [Section 2 of RFC 7841](#).

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <https://www.rfc-editor.org/info/rfc8407>.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
1.1.	Changes since RFC 6087	5
2.	Terminology	6
2.1.	NETCONF Terms	7
2.2.	YANG Terms	7
2.3.	NMDA Terms	7
2.4.	Requirements Notation	8
3.	General Documentation Guidelines	8
3.1.	Module Copyright	9
3.2.	Code Components	9
3.2.1.	Example Modules	9
3.3.	Terminology Section	10
3.4.	Tree Diagrams	10
3.5.	Narrative Sections	10
3.6.	Definitions Section	11
3.7.	Security Considerations Section	11
3.7.1.	Security Considerations Section Template	12
3.8.	IANA Considerations Section	13
3.8.1.	Documents That Create a New Namespace	14
3.8.2.	Documents That Extend an Existing Namespace	14
3.9.	References Sections	14
3.10.	Validation Tools	14
3.11.	Module Extraction Tools	15
3.12.	Module Usage Examples	15
4.	YANG Usage Guidelines	15
4.1.	Module Naming Conventions	16
4.2.	Prefixes	17
4.3.	Identifiers	18
4.3.1.	Identifier Naming Conventions	18
4.4.	Defaults	19
4.5.	Conditional Statements	19
4.6.	XPath Usage	20
4.6.1.	XPath Evaluation Contexts	20
4.6.2.	Function Library	21
4.6.3.	Axes	22
4.6.4.	Types	23
4.6.5.	Wildcards	24
4.6.6.	Boolean Expressions	24
4.7.	YANG Definition Lifecycle Management	25
4.8.	Module Header, Meta, and Revision Statements	26
4.9.	Namespace Assignments	28
4.10.	Top-Level Data Definitions	29
4.11.	Data Types	30
4.11.1.	Fixed-Value Extensibility	30
4.11.2.	Patterns and Ranges	31
4.11.3.	Enumerations and Bits	32

4.11.4. Union Types	33
4.11.5. Empty and Boolean	34
4.12. Reusable Type Definitions	35
4.13. Reusable Groupings	35
4.14. Data Definitions	36
4.14.1. Non-Presence Containers	38
4.14.2. Top-Level Data Nodes	38
4.15. Operation Definitions	39
4.16. Notification Definitions	39
4.17. Feature Definitions	40
4.18. YANG Data Node Constraints	41
4.18.1. Controlling Quantity	41
4.18.2. "must" versus "when"	41
4.19. "augment" Statements	41
4.19.1. Conditional Augment Statements	41
4.19.2. Conditionally Mandatory Data Definition Statements	42
4.20. Deviation Statements	43
4.21. Extension Statements	44
4.22. Data Correlation	45
4.22.1. Use of "leafref" for Key Correlation	46
4.23. Operational State	47
4.23.1. Combining Operational State and Configuration Data	47
4.23.2. Representing Operational Values of Configuration Data	47
4.23.3. NMDA Transition Guidelines	48
4.24. Performance Considerations	52
4.25. Open Systems Considerations	52
4.26. Guidelines for Constructs Specific to YANG 1.1	53
4.26.1. Importing Multiple Revisions	53
4.26.2. Using Feature Logic	53
4.26.3. "anyxml" versus "anydata"	53
4.26.4. "action" versus "rpc"	53
4.27. Updating YANG Modules (Published versus Unpublished)	54
5. IANA Considerations	55
6. Security Considerations	55
7. References	56
7.1. Normative References	56
7.2. Informative References	57
Appendix A. Module Review Checklist	59
Appendix B. YANG Module Template	61
Acknowledgments	62
Author's Address	63

1. Introduction

The standardization of network configuration interfaces for use with network configuration management protocols, such as the Network Configuration Protocol [RFC6241] and the RESTCONF protocol [RFC8040], requires a modular set of data models that can be reused and extended over time.

This document defines a set of usage guidelines for documents containing YANG 1.1 [RFC7950] and YANG 1.0 [RFC6020] data models. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF and/or RESTCONF server. A NETCONF or RESTCONF server that supports a particular YANG module will support client NETCONF and/or RESTCONF operation requests, as indicated by the specific content defined in the YANG module.

Many YANG constructs are defined as optional to use, such as the "description" statement. However, in order to make YANG modules more useful, it is desirable to define a set of usage guidelines that entails a higher level of compliance than the minimum level defined in the YANG specification [RFC7950].

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], and the RESTCONF methods and RESTCONF resources, as defined in [RFC8040].

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 implementing the guidance in this document.

1.1. Changes since RFC 6087

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

- o Updated NETCONF reference from RFC 4741 to RFC 6241
- o Updated NETCONF over the Secure Shell (SSH) citation from RFC 4742 to RFC 6242
- o Updated YANG Types reference from RFC 6021 to RFC 6991
- o Updated obsolete URLs for IETF resources
- o Changed top-level data node guideline
- o Clarified XML Path Language (XPath) usage for a literal value representing a YANG identity
- o Clarified XPath usage for a when-stmt
- o Clarified XPath usage for "preceding-sibling" and "following-sibling" axes
- o Added terminology guidelines
- o Added mention of RFC 8174, which updates RFC 2119 by clarifying the use of capitalized key words
- o Added YANG tree diagram guidelines
- o Updated XPath guidelines for type conversions and function library usage
- o Updated "Data Types" section
- o Updated "Notification Definitions" section
- o Clarified conditional key leaf nodes
- o Clarified usage of "uint64" and "int64" data types
- o Added text on YANG feature usage
- o Added "Identifier Naming Conventions" section
- o Clarified use of mandatory nodes with conditional augmentations

- o Clarified namespace and domain conventions for example modules
- o Clarified conventions for identifying code components
- o Added YANG 1.1 guidelines
- o Added "YANG Data Node Constraints" section
- o Added mention of the RESTCONF protocol
- o Added guidelines for datastores revised by the Network Management Datastore Architecture (NMDA)

2. Terminology

The following terms are used throughout this document:

- o published: A stable release of a module or submodule. For example, the "Request for Comments" described in [Section 2.1 of \[RFC2026\]](#) is considered a stable publication.
- o unpublished: An unstable release of a module or submodule. For example the "Internet-Draft" described in [Section 2.2 of \[RFC2026\]](#) is considered an unstable publication that is a work in progress, subject to change at any time.
- o YANG fragment: A set of YANG statements that are not intended to represent a complete YANG module or submodule. These statements are not intended for actual use, except to provide an example of YANG statement usage. The invalid syntax "..." is sometimes used to indicate that additional YANG statements would be present in a real YANG module.
- o YANG tree diagram: A diagram representing the contents of a YANG module, as defined in [\[RFC8340\]](#). It is also called a "tree diagram".

2.1. NETCONF Terms

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

- o capabilities
- o client
- o operation
- o server

2.2. YANG Terms

The following terms are defined in [RFC7950] and are not redefined here:

- o data node
- o module
- o namespace
- o submodule
- o version
- o YANG
- o 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.

2.3. NMDA Terms

The following terms are defined in [RFC8342] and are not redefined here:

- o configuration
- o conventional configuration datastore
- o datastore

- o operational state
- o operational state datastore

2.4. Requirements Notation

The key words "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 \[RFC2119\] \[RFC8174\]](#) when, and only when, they appear in all capitals, as shown here.

3. General Documentation Guidelines

YANG modules under review are likely to be contained in Internet-Drafts (I-Ds). All guidelines for I-D authors [[ID-Guidelines](#)] MUST be followed. The guidelines for RFCs should be followed and are defined in the following: [[RFC7322](#)] (and any future RFCs that obsolete it), [[RFC-STYLE](#)], and [[RFC7841](#)].

The following sections MUST be present in an I-D containing a module:

- o Narrative sections
- o Definition sections
- o Security Considerations section
- o IANA Considerations section
- o References section

There are three usage scenarios for YANG that can appear in an I-D or RFC:

- o normative module or submodule
- o example module or submodule
- o example YANG fragment not part of any module or submodule

The guidelines in this document refer mainly to a normative module or submodule but may be applicable to example modules and YANG fragments as well.

3.1. Module Copyright

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

[<https://trustee.ietf.org/license-info/>](https://trustee.ietf.org/license-info/)

3.2. Code Components

Each normative YANG module or submodule contained within an I-D 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 \[RFC7950\]](#). The name string form that includes the revision date SHOULD be used. The revision date MUST match the date used in the most recent revision of the module.

The following example is for the "2016-03-20" revision of the "ietf-foo" module:

```
<CODE BEGINS> file "ietf-foo@2016-03-20.yang"

module ietf-foo {
  namespace "urn:ietf:params:xml:ns:yang:ietf-foo";
  prefix "foo";
  organization "...";
  contact "...";
  description "...";
  revision 2016-03-20 {
    description "Latest revision";
    reference "RFC XXXX: Foo Protocol";
  }
  // ... more statements
}

<CODE ENDS>
```

3.2.1. Example Modules

Example modules are not code components. The <CODE BEGINS> convention MUST NOT be used for example modules.

An example module SHOULD be named using the term "example", followed by a hyphen, followed by a descriptive name, e.g., "example-toaster".

See [Section 4.9](#) regarding the namespace guidelines for example modules.

3.3. 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.

3.4. Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module and **SHOULD** be included to help readers understand YANG module structure. Guidelines on tree diagrams can be found in [Section 3 of \[RFC8340\]](#).

If YANG tree diagrams are used, then an informative reference to the YANG tree diagrams specification **MUST** be included in the document. Refer to [Section 2.2 of \[RFC8349\]](#) for an example of such a reference.

3.5. 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 or modules defined by the specification imports definitions from other modules (except for those defined in [\[RFC7950\]](#) or [\[RFC6991\]](#)) or are always implemented in conjunction with other modules, then those facts **MUST** be noted in the overview section; any special interpretations of definitions in other modules **MUST** be noted as well. Refer to [Section 2.3 of \[RFC8349\]](#) for an example of this overview section.

If the document contains a YANG module(s) that is compliant with NMDA [\[RFC8342\]](#), then the Introduction section should mention this fact.

Example:

The YANG data model in this document conforms to the Network Management Datastore Architecture defined in [RFC 8342](#).

Consistent indentation SHOULD be used for all examples, including YANG fragments and protocol message instance data. If line wrapping is done for formatting purposes, then this SHOULD be noted, as shown in the following example:

```
[note: '\' line wrapping for formatting only]
```

```
<myleaf xmlns="tag:example.com,2017:example-two">\
  this is a long value so the line needs to wrap to stay\
  within 72 characters\
</myleaf>
```

3.6. Definitions Section

This section contains the module(s) defined by the specification. These modules SHOULD be written using the YANG 1.1 [RFC7950] syntax. YANG 1.0 [RFC6020] syntax MAY be used if no YANG 1.1 constructs or semantics are needed in the module. If any of the imported YANG modules are written using YANG 1.1, then the module MUST be written using YANG 1.1.

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 Structure of Management Information Version 2 (SMIV2), which are not affected by these guidelines.

Note that if the module itself is considered normative and not an example module or example YANG fragment, then all YANG statements within a YANG module are considered normative. The use of keywords defined in [RFC2119] and [RFC8174] apply to YANG "description" statements in normative modules exactly as they would in any other normative section.

Example YANG modules and example YANG fragments MUST NOT contain any normative text, including any all-uppercase reserved words from [RFC2119] and [RFC8174].

Consistent indentation and formatting SHOULD be used in all YANG statements within a module.

See [Section 4](#) for guidelines on YANG usage.

3.7. 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 <<https://trac.ietf.org/trac/ops/wiki/yang-security-guidelines>>). Section 3.7.1 contains the security considerations template dated 2013-05-08 and last updated on 2018-07-02. Authors MUST check the web page at the URL listed above in case there is a more recent version available.

In particular:

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

3.7.1. Security Considerations Section Template

X. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

- 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 that 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>

3.8. IANA Considerations Section

In order to comply with IESG policy as set forth in <<https://www.ietf.org/id-info/checklist.html>>, every I-D 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 will state that "This document has no IANA actions". Refer to the guidelines in [RFC8126] for more details.

Each normative YANG module MUST be registered in both the "IETF XML Registry" [RFC3688] [IANA-XML] and the "YANG Module Names" registry [RFC6020] [IANA-MOD-NAMES]. This applies to new modules and updated modules. An example of an update registration for the "ietf-template" module can be found in [Section 5](#).

3.8.1. Documents That Create a New Namespace

If an I-D 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 entry in the "ns" subregistry within the "IETF XML Registry" **MUST** be requested from the IANA.

3.8.2. 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.

3.9. References Sections

For every import or include statement that appears in a module contained in the specification that 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 that 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 that identifies a separate document, a corresponding informative reference to that document **MAY** appear in the Informative References section.

3.10. Validation Tools

All modules need to be validated before submission in an I-D. The 'pyang' YANG compiler is freely available from GitHub:

<<https://github.com/mbj4668/pyang>>

If the 'pyang' compiler is used to validate a normative module, then the "--ietf" command-line option **MUST** be used to identify any IETF guideline issues.

If the 'pyang' compiler is used to validate an example module, then the "--ietf" command-line option MAY be used to identify any IETF guideline issues.

The "yanglint" program is also freely available from GitHub.

<<https://github.com/CESNET/libyang>>

This tool can be used to validate XPath statements within YANG modules.

3.11. Module Extraction Tools

A version of 'rfcstrip' that will extract YANG modules from an I-D or RFC is available. The 'rfcstrip' tool that supports YANG module extraction is freely available at:

<<https://github.com/mbj4668/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.

The "xym" tool is freely available on GitHub and can be used to extract YANG modules from a document.

<<https://github.com/xym-tool/xym>>

3.12. Module Usage Examples

Each specification that defines one or more modules SHOULD contain usage examples, either throughout the document or in an appendix. This includes example instance document snippets in an appropriate encoding (e.g., XML and/or JSON) to demonstrate the intended usage of the YANG module(s). Example modules MUST be validated. Refer to [Section 3.10](#) for tools that validate YANG modules. If IP addresses are used, then a mix of either IPv4 and IPv6 addresses or IPv6 addresses exclusively SHOULD be used in the examples.

4. YANG Usage Guidelines

Modules in IETF Standards Track specifications MUST comply with all syntactic and semantic requirements of YANG 1.1 [[RFC7950](#)]. See the exception for YANG 1.0 in [Section 3.6](#). The guidelines in this section are intended to supplement the YANG specification [[RFC7950](#)], 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.

4.1. Module Naming Conventions

Normative modules contained in Standards Track documents **MUST** be named according to the guidelines in the IANA Considerations section of [RFC7950].

A distinctive word or abbreviation (e.g., protocol name or working group abbreviation) **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 abbreviation 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.

Example modules are non-normative and **SHOULD** be named with the prefix "example-".

It is suggested that a stable prefix be selected that represents the entire organization. All normative YANG modules published by the IETF **MUST** begin with the prefix "ietf-". Another standards organization, such as the IEEE, might use the prefix "ieee-" for all YANG modules.

Once a module name is published, it **MUST NOT** be reused, even if the RFC containing the module is reclassified to "Historic" status. A module name cannot be changed in YANG, and this would be treated as a new module, not a name change.

4.2. 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 three modules:

```
module example-foo {
  namespace "tag:example.com,2017:example-foo";
  prefix f;

  container foo;
}

module example-bar {
  namespace "tag:example.com,2017:example-bar";
  prefix b;

  typedef foo { type uint32; }
}

module example-one {
  namespace "tag:example.com,2017:example-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:

- o Prefixes are never used for built-in data types and YANG keywords.
- o A prefix MUST be used for any external statement (i.e., a statement defined with the YANG "extension" statement).
- o The proper module prefix MUST be used for all identifiers imported from other modules.
- o 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:

- o The local module prefix SHOULD be used instead of no prefix in all path expressions.
- o The local module prefix MUST be used instead of no prefix in all "default" statements for an "identityref" or "instance-identifier" data type.
- o The local module prefix MAY be used for references to typedefs, groupings, extensions, features, and identities defined in the module.

Prefix values SHOULD be short but are also likely to be unique. Prefix values SHOULD NOT conflict with known modules that have been previously published.

4.3. 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 14 of \[RFC7950\]](#).

4.3.1. Identifier Naming Conventions

Identifiers SHOULD follow a consistent naming pattern throughout the module. Only lowercase letters, numbers, and dashes SHOULD be used in identifier names. Uppercase characters, the period character, and the underscore character MAY be used if the identifier represents a well-known value that uses these characters. YANG does not permit any other characters in YANG identifiers.

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 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 modeling properties. Only YANG statements and YANG extension statements are designed to convey machine-readable data modeling 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.

4.4. 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

4.5. 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 or RESTCONF protocol capability, then a YANG "feature" statement SHOULD be defined. The defined "feature" statement SHOULD then be used in the conditional "if-feature" statement referencing the optional data definition.

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 leaves 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.

4.6. XPath Usage

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

4.6.1. XPath Evaluation Contexts

YANG defines five 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 okay because when-stmt context is this notification  
    when "/if:interfaces/if:interface[name='eth0']";  
    type leafref {  
      // Okay 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.

4.6.2. 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 that defines the must or when expression containing the "local-name" function. Example of a "local-name" function that should not be used:

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

The "derived-from-or-self" function SHOULD be used instead of an equality expression for identityref values. This allows the identities to be conceptually augmented.

Example:

```
// do not use
when "md-name-format = 'name-format-null'";

// this is preferred
when "derived-from-or-self(md-name-format, 'name-format-null')";
```

4.6.3. Axes

The "attribute" and "namespace" axes are not supported in YANG and MAY be empty in a NETCONF or RESTCONF server implementation.

The "preceding" and "following" axes SHOULD NOT be used. These constructs rely on XML document order within a NETCONF or RESTCONF 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, and "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 be 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".

4.6.4. 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
}
```

4.6.5. Wildcards

It is possible to construct XPath expressions that will evaluate differently when combined with several modules within a server implementation rather than 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 to 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.";
}
```

4.6.6. 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:fool = 'one' or /f:fool = 'three'";  
  type string;  
}
```

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

```
leaf fool {  
  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.

4.7. YANG Definition Lifecycle Management

The YANG status statement MUST be present within a definition 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 a "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 import 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 an import statement is for a module from a stable source (e.g., an RFC for an IETF module), then a reference-stmt SHOULD be present within an import statement.

```
import ietf-yang-types {  
    prefix yang;  
    reference "RFC 6991: Common YANG Data Types";  
}
```

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

Definitions for future use SHOULD NOT be specified in a module. Do not specify placeholder objects like the "reserved" example below:

```
leaf reserved {  
    type string;  
    description  
        "This object has no purpose at this time, but a future  
        revision of this module might define a purpose  
        for this object.";  
}
```

4.8. 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 IETF Standards Track status, then the organization SHOULD be the IETF working group (WG) chartered to write the document. For other standards organizations, a similar approach is also suggested.

The "contact" statement MUST be present. If the module is contained in a document intended for Standards Track status, then the WG web and mailing information SHOULD 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. There is no need to include the contact information for WG Chairs.

The "description" statement MUST be present. For modules published within IETF documents, the appropriate IETF Trust Copyright text MUST be present, as described in [Section 3.1](#).

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.

The following example shows the revision statement for a published YANG module:

```
revision "2012-02-22" {  
  description  
    "Initial version";  
  reference  
    "RFC 8341: Network Configuration  
      Access Control Model";  
}
```

For an unpublished module, a complete history of each unpublished module revision is not required. That is, within a sequence of draft versions, only the most recent revision need be recorded in the module. Do not remove or reuse a revision statement for a published module. A new revision date is not required unless the module contents have changed. If the module contents have changed, then the revision date of that new module version **MUST** be updated to a date later than that of the previous version.

The following example shows the two revision statements for an unpublished update to a published YANG module:

```
revision "2017-12-11" {
  description
    "Added support for YANG 1.1 actions and notifications tied to
    data nodes. Clarify how NACM extensions can be used by other
    data models.";
  reference
    "RFC 8407: Network Configuration Protocol (NETCONF)
    Access Control Model";
}

revision "2012-02-22" {
  description
    "Initial version";
  reference
    "RFC 8341: Network Configuration
    Access Control Model";
}
```

4.9. Namespace Assignments

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

- o the module to compile correctly instead of generating disruptive fatal errors.
- o early implementors to use the modules without picking a random value for the XML namespace.
- o 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 Names" 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 modules that are not Standards Track. The string SHOULD be selected according to the guidelines in [RFC7950].

The following URIs exemplify what might be used by modules that are not Standards Track. Note that the domain "example.com" SHOULD be used by example modules in IETF I-Ds. These URIs are not intended to be dereferenced. They are used for module namespace identification only.

Example URIs using URLs per [RFC3986]:

```
https://example.com/ns/example-interfaces
```

```
https://example.com/ns/example-system
```

Example URIs using tags per [RFC4151]:

```
tag:example.com,2017:example-interfaces
```

```
tag:example.com,2017:example-system
```

4.10. 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 sometimes useful to define separate top-level containers for configuration and non-configuration data. For some existing top-level data nodes, configuration data was not in scope, so only one container representing operational state was created. Refer to NMDA [RFC8342] for details.

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, it 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.

4.11. Data Types

Selection of an appropriate data type (i.e., built-in type, existing derived type, or new derived type) is very subjective; 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.

4.11.1. 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.

4.11.2. 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. If the patterns used in a type definition have known limitations such as false negative or false positive matches, then these limitations SHOULD be documented within the typedef or data definition.

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";  
  }  
  ...  
}
```

4.11.3. 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...  ";  
}
```

4.11.4. Union Types

The YANG "union" type is evaluated by testing a value against each member type in the union. The first type definition that accepts a value as valid is the member type used. In general, member types SHOULD be ordered from most restrictive to least restrictive types.

In the following example, the "enumeration" type will never be matched because the preceding "string" type will match everything.

Incorrect:

```
type union {  
  type string;  
  type enumeration {  
    enum up;  
    enum down;  
  }  
}
```

Correct:

```
type union {  
  type enumeration {  
    enum up;  
    enum down;  
  }  
  type string;  
}
```

It is possible for different member types to match, depending on the input encoding format. In XML, all values are passed as string nodes; but in JSON, there are different value types for numbers, booleans, and strings.

In the following example, a JSON numeric value will always be matched by the "int32" type, but in XML the string value representing a number will be matched by the "string" type. The second version will match the "int32" member type no matter how the input is encoded.

Incorrect:

```
type union {  
    type string;  
    type int32;  
}
```

Correct:

```
type union {  
    type int32;  
    type string;  
}
```

4.11.5. Empty and Boolean

YANG provides an "empty" data type, which has one value (i.e., present). The default is "not present", which is not actually a value. When used within a list key, only one value can (and must) exist for this key leaf. The type "empty" SHOULD NOT be used for a key leaf since it is pointless.

There is really no difference between a leaf of type "empty" and a leaf-list of type "empty". Both are limited to one instance. The type "empty" SHOULD NOT be used for a leaf-list.

The advantage of using type "empty" instead of type "boolean" is that the default (not present) does not take up any bytes in a representation. The disadvantage is that the client may not be sure if an empty leaf is missing because it was filtered somehow or not implemented. The client may not have a complete and accurate schema for the data returned by the server and may not be aware of the missing leaf.

The YANG "boolean" data type provides two values ("true" and "false"). When used within a list key, two entries can exist for this key leaf. Default values are ignored for key leaves, but a default statement is often used for plain boolean leaves. The advantage of the "boolean" type is that the leaf or leaf-list has a clear representation for both values. The default value is usually not returned unless explicitly requested by the client, so no bytes are used in a typical representation.

In general, the "boolean" data type SHOULD be used instead of the "empty" data type, as shown in the example below:

Incorrect:

```
leaf flag1 {  
    type empty;  
}
```

Correct:

```
leaf flag2 {  
    type boolean;  
    default false;  
}
```

4.12. 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.

4.13. Reusable Groupings

A reusable grouping is a YANG grouping that can be imported by another module and is intended for use by other modules. This is not the same as a grouping that is used within the module in which it is defined, but it happens to be exportable to another module because it is defined at the top level of the YANG module.

The following guidelines apply to reusable groupings, in order to make them as robust as possible:

- o Clearly identify the purpose of the grouping in the "description" statement.
- o There are five different XPath contexts in YANG (rpc/input, rpc/output, notification, "config true" data nodes, and all data nodes). Clearly identify which XPath contexts are applicable or excluded for the grouping.
- o Do not reference data outside the grouping in any "path", "must", or "when" statements.
- o Do not include a "default" substatement on a leaf or choice unless the value applies on all possible contexts.
- o Do not include a "config" substatement on a data node unless the value applies on all possible contexts.
- o Clearly identify any external dependencies in the grouping "description" statement, such as nodes referenced by an absolute path from a "path", "must", or "when" statement.

4.14. Data Definitions

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

- o anyxml
- o augment
- o choice
- o container
- o extension
- o feature
- o grouping
- o identity
- o leaf
- o leaf-list

- o list
- o notification
- o rpc
- o 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 that configuration anyxml nodes will not be 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.

The "choice" statement is allowed to be directly present within a "case" statement in YANG 1.1. This needs to be considered carefully. Consider simply including the nested "choice" as additional "case" statements within the parent "choice" statement. Note that the "mandatory" and "default" statements within a nested "choice" statement only apply if the "case" containing the nested "choice" statement is first selected.

If a list defines any key leafs, then these leafs **SHOULD** be defined in order, as the first child nodes within the list. The key leafs **MAY** be in a different order in some cases, e.g., they are defined in a grouping, and not inline in the list statement.

4.14.1. Non-Presence Containers

A non-presence container is used to organize data into specific subtrees. It is not intended to have semantics within the data model beyond this purpose, although YANG allows it (e.g., a "must" statement within the non-presence container).

Example using container wrappers:

```
container top {  
  container foos {  
    list foo { ... }  
  }  
  container bars {  
    list bar { ... }  
  }  
}
```

Example without container wrappers:

```
container top {  
  list foo { ... }  
  list bar { ... }  
}
```

Use of non-presence containers to organize data is a subjective matter similar to use of subdirectories in a file system. Although these containers do not have any semantics, they can impact protocol operations for the descendant data nodes within a non-presence container, so use of these containers SHOULD be considered carefully.

The NETCONF and RESTCONF protocols do not currently support the ability to delete all list (or leaf-list) entries at once. This deficiency is sometimes avoided by use of a parent container (i.e., deleting the container also removes all child entries).

4.14.2. Top-Level Data Nodes

Use of top-level objects needs to be considered carefully:

- o top-level siblings are not ordered
- o top-level siblings are not static and depend on the modules that are loaded

- o for subtree filtering, retrieval of a top-level leaf-list will be treated as a content-match node for all top-level-siblings
- o a top-level list with many instances may impact performance

4.15. 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.

4.16. 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.

4.17. 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 description-stmt within a feature-stmt MUST specify any interactions with other features.

The set of YANG features defined in a module should be considered carefully. Very fine granular features increase interoperability complexity and should be avoided. A likely misuse of the feature mechanism is the tagging of individual leafs (e.g., counters) with separate 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 identify. Module names with a common prefix can help readers identify 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";
}
```


4.18. YANG Data Node Constraints

4.18.1. Controlling Quantity

The "min-elements" and "max-elements" statements can be used to control how many list or leaf-list instances are required for a particular data node. YANG constraint statements SHOULD be used to identify conditions that apply to all implementations of the data model. If platform-specific limitations (e.g., the "max-elements" supported for a particular list) are relevant to operations, then a data model definition statement (e.g., "max-ports" leaf) SHOULD be used to identify the limit.

4.18.2. "must" versus "when"

"must" and "when" YANG statements are used to provide cross-object referential tests. They have very different behavior. The "when" statement causes data node instances to be silently deleted as soon as the condition becomes false. A false "when" expression is not considered to be an error.

The "when" statement SHOULD be used together with "augment" or "uses" statements to achieve conditional model composition. The condition SHOULD be based on static properties of the augmented entry (e.g., list key leafs).

The "must" statement causes a datastore validation error if the condition is false. This statement SHOULD be used for enforcing parameter value restrictions that involve more than one data node (e.g., end-time parameter must be after the start-time parameter).

4.19. "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.

4.19.1. 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 {  
            ...  
        }  
    }  
}
```

4.19.2. 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 a 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 the target data node because the client does not have any control over this external data.

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

```
module example-module {  
  
    yang-version 1.1;  
    namespace "tag:example.com,2017:example-module";  
    prefix mymod;  
  
    import iana-if-type { prefix iana; }  
    import ietf-interfaces { prefix if; }  
  
    identity some-new-iftype {  
        base iana:iana-interface-type;  
    }  
  
    augment "/if:interfaces/if:interface" {  
        when "if:type = 'mymod:some-new-iftype'";  
  
        leaf mandatory-leaf {  
            type string;  
            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.

4.20. Deviation Statements

Per [RFC 7950, Section 7.20.3](#), the YANG "deviation" statement is not allowed to appear in IETF YANG modules, but it can be useful for documenting server capabilities. Deviation statements are not reusable and typically not shared across all platforms.

There are several reasons that deviations might be needed in an implementation, e.g., an object cannot be supported on all platforms, or feature delivery is done in multiple development phases. Deviation statements can also be used to add annotations to a module, which does not affect the conformance requirements for the module.

It is suggested that deviation statements be defined in separate modules from regular YANG definitions. This allows the deviations to be platform specific and/or temporary.

The order that deviation statements are evaluated can affect the result. Therefore, multiple deviation statements in the same module, for the same target object, SHOULD NOT be used.

The "max-elements" statement is intended to describe an architectural limit to the number of list entries. It is not intended to describe platform limitations. It is better to use a "deviation" statement for the platforms that have a hard resource limit.

Example documenting platform resource limits:

Wrong: (max-elements in the list itself)

```
container backups {
  list backup {
    ...
    max-elements 10;
    ...
  }
}
```

Correct: (max-elements in a deviation)

```
deviation /bk:backups/bk:backup {
  deviate add {
    max-elements 10;
  }
}
```

4.21. Extension Statements

The YANG "extension" statement is used to specify external definitions. This appears in the YANG syntax as an "unknown-statement". Usage of extension statements in a published module needs to be considered carefully.

The following guidelines apply to the usage of YANG extensions:

- o The semantics of the extension MUST NOT contradict any YANG statements. Extensions can add semantics not covered by the normal YANG statements.

- o The module containing the extension statement **MUST** clearly identify the conformance requirements for the extension. It should be clear whether all implementations of the YANG module containing the extension need to also implement the extension. If not, identify what conditions apply that would require implementation of the extension.
- o The extension **MUST** clearly identify where it can be used within other YANG statements.
- o The extension **MUST** clearly identify if YANG statements or other extensions are allowed or required within the extension as substatements.

4.22. 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:

- o **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.
- o **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.
- o **mirror**: create new objects in a new module or the original module, except use a new 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" substatement 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" substatement **MUST** be set to "false". Loose coupling is achieved by using key leaves with the same data type as the original data structure. This has the same semantics as setting the "require-instance" substatement to "false".

The relationship between configuration and operational state has been clarified in NMDA [[RFC8342](#)].

4.22.1. Use of "leafref" for Key Correlation

Sometimes it is not practical to augment a data structure. For example, the correlated data could have different keys or contain mandatory nodes.

The following example shows the use of the "leafref" data type for data correlation purposes:

Not preferred:

```
list foo {
  key name;
  leaf name {
    type string;
  }
  ...
}

list foo-addon {
  key name;
  config false;
  leaf name {
    type string;
  }
  ...
}
```

Preferred:

```
list foo {
  key name;
  leaf name {
    type string;
  }
  ...
}

list foo-addon {
  key name;
  config false;
  leaf name {
    type leafref {
      path "/foo/name";
      require-instance false;
    }
  }
  leaf addon {
```

```
        type string;
        mandatory true;
    }
}
```

4.23. Operational State

The modeling of operational state with YANG has been refined over time. At first, only data that has a "config" statement value of "false" was considered to be operational state. This data was not considered to be part of any datastore, which made the YANG XPath definition much more complicated.

Operational state is now modeled using YANG according to the new NMDA [RFC8342] and conceptually contained in the operational state datastore, which also includes the operational values of configuration data. There is no longer any need to duplicate data structures to provide separate configuration and operational state sections.

This section describes some data modeling issues related to operational state and guidelines for transitioning YANG data model design to be NMDA compatible.

4.23.1. Combining Operational State and Configuration Data

If possible, operational state SHOULD be combined with its associated configuration data. This prevents duplication of key leafs and ancestor nodes. It also prevents race conditions for retrieval of dynamic entries and allows configuration and operational state to be retrieved together with minimal message overhead.

```
container foo {
    ...
    // contains "config true" and "config false" nodes that have
    // no corresponding "config true" object (e.g., counters)
}
```

4.23.2. Representing Operational Values of Configuration Data

If possible, the same data type SHOULD be used to represent the configured value and the operational value, for a given leaf or leaf-list object.

Sometimes the configured value set is different than the operational value set for that object, for example, the "admin-status" and "oper-status" leafs in [RFC8343]. In this case, a separate object MAY be used to represent the configured and operational values.

Sometimes the list keys are not identical for configuration data and the corresponding operational state. In this case, separate lists MAY be used to represent the configured and operational values.

If it is not possible to combine configuration and operational state, then the keys used to represent list entries SHOULD be the same type. The "leafref" data type SHOULD be used in operational state for key leafs that have corresponding configuration instances. The "require-instance" statement MAY be set to "false" (in YANG 1.1 modules only) to indicate instances are allowed in the operational state that do not exist in the associated configuration data.

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 and justify any NMDA exceptions in detail, such as the use of separate subtrees and/or separate leafs. The "description" statements for both the configuration and the operational state SHOULD be used for this purpose.

4.23.3. NMDA Transition Guidelines

YANG modules SHOULD be designed with the assumption that they will be used on servers supporting the operational state datastore. With this in mind, YANG modules SHOULD define "config false" nodes wherever they make sense to the data model. "Config false" nodes SHOULD NOT be defined to provide the operational value for configuration nodes, except when the value space of a configured and operational value may differ, in which case a distinct "config false" node SHOULD be defined to hold the operational value for the configured node.

The following guidelines are meant to help modelers develop YANG modules that will maximize the utility of the model with both current and new implementations.

New modules and modules that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA compatible, as described in [Section 4.23.1](#). This transition MAY be deferred if the module does not contain any configuration datastore objects.

The remaining are options that MAY be followed during the time that NMDA mechanisms are being defined.

- (a) Modules that require immediate support for the NMDA features SHOULD be structured for NMDA. A temporary non-NMDA version of this type of module MAY exist, as either an existing model or a model created by hand or with suitable tools that mirror the current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative appendix. The use of the non-NMDA module will allow temporary bridging of the time period until NMDA implementations are available.
- (b) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] has been restructured as an NMDA-compatible model in [RFC8343]. The "/interfaces-state" hierarchy has been marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.
- (c) For models that augment models that have not been structured with the NMDA, the modeler will have to consider the structure of the base model and the guidelines listed above. Where possible, such models should move to new revisions of the base model that are NMDA compatible. When that is not possible, augmenting "state" containers SHOULD be avoided, with the expectation that the base model will be re-released with the state containers marked as deprecated. It is RECOMMENDED to augment only the "/foo" hierarchy of the base model. Where this recommendation cannot be followed, then any new "state" elements SHOULD be included in their own module.

4.23.3.1. Temporary Non-NMDA Modules

A temporary non-NMDA module allows a non-NMDA-aware client to access operational state from an NMDA-compliant server. It contains the top-level "config false" data nodes that would have been defined in a legacy YANG module (before NMDA).

A server that needs to support both NMDA and non-NMDA clients can advertise both the new NMDA module and the temporary non-NMDA module. A non-NMDA client can use separate "foo" and "foo-state" subtrees, except the "foo-state" subtree is located in a different (temporary)

module. The NMDA module can be used by a non-NMDA client to access the conventional configuration datastores and the deprecated <get> operation to access nested "config false" data nodes.

To create the temporary non-NMDA model from an NMDA model, the following steps can be taken:

- o Change the module name by appending "-state" to the original module name
- o Change the namespace by appending "-state" to the original namespace value
- o Change the prefix by appending "-s" to the original prefix value
- o Add an import to the original module (e.g., for typedef definitions)
- o Retain or create only the top-level nodes that have a "config" statement value "false". These subtrees represent "config false" data nodes that were combined into the configuration subtree; therefore, they are not available to non-NMDA aware clients. Set the "status" statement to "deprecated" for each new node.
- o The module description SHOULD clearly identify the module as a temporary non-NMDA module

4.23.3.2. Example: Create a New NMDA Module

Create an NMDA-compliant module, using combined configuration and state subtrees, whenever possible.

```
module example-foo {  
  namespace "urn:example.com:params:xml:ns:yang:example-foo";  
  prefix "foo";  
  
  container foo {  
    // configuration data child nodes  
    // operational value in operational state datastore only  
    // may contain "config false" nodes as needed  
  }  
}
```

4.23.3.3. Example: Convert an Old Non-NMDA Module

Do not remove non-compliant objects from existing modules. Instead, change the status to "deprecated". At some point, usually after 1 year, the status MAY be changed to "obsolete".

Old Module:

```
module example-foo {
  namespace "urn:example.com:params:xml:ns:yang:example-foo";
  prefix "foo";

  container foo {
    // configuration data child nodes
  }

  container foo-state {
    config false;
    // operational state child nodes
  }
}
```

Converted NMDA Module:

```
module example-foo {
  namespace "urn:example.com:params:xml:ns:yang:example-foo";
  prefix "foo";

  container foo {
    // configuration data child nodes
    // operational value in operational state datastore only
    // may contain "config false" nodes as needed
    // will contain any data nodes from old foo-state
  }

  // keep original foo-state but change status to deprecated
  container foo-state {
    config false;
    status deprecated;
    // operational state child nodes
  }
}
```

4.23.3.4. Example: Create a Temporary NMDA Module

Create a new module that contains the top-level operational state data nodes that would have been available before they were combined with configuration data nodes (to be NMDA compliant).

```
module example-foo-state {  
  namespace "urn:example.com:params:xml:ns:yang:example-foo-state";  
  prefix "foo-s";  
  
  // import new or converted module; not used in this example  
  import example-foo { prefix foo; }  
  
  container foo-state {  
    config false;  
    status deprecated;  
    // operational state child nodes  
  }  
}
```

4.24. Performance Considerations

It is generally likely that certain YANG statements require more runtime resources than other statements. Although there are no performance requirements for YANG validation, the following information MAY be considered when designing YANG data models:

- o Lists are generally more expensive than containers
- o "when" statement evaluation is generally more expensive than "if-feature" or "choice" statements
- o "must" statements are generally more expensive than "min-entries", "max-entries", "mandatory", or "unique" statements
- o "identityref" leafs are generally more expensive than "enumeration" leafs
- o "leafref" and "instance-identifier" types with "require-instance" set to true are generally more expensive than if "require-instance" is set to false

4.25. Open Systems Considerations

Only the modules imported by a particular module can be assumed to be present in an implementation. An open system MAY include any combination of YANG modules.

4.26. Guidelines for Constructs Specific to YANG 1.1

The set of guidelines for YANG 1.1 will grow as operational experience is gained with the new language features. This section contains an initial set of guidelines for new YANG 1.1 language features.

4.26.1. Importing Multiple Revisions

Standard modules SHOULD NOT import multiple revisions of the same module into a module. This MAY be done if independent definitions (e.g., enumeration typedefs) from specific revisions are needed in the importing module.

4.26.2. Using Feature Logic

The YANG 1.1 feature logic is much more expressive than YANG 1.0. A "description" statement SHOULD describe the "if-feature" logic in text, to help readers understand the module.

YANG features SHOULD be used instead of the "when" statement, if possible. Features are advertised by the server, and objects conditional by the "if-feature" statement are conceptually grouped together. There is no such commonality supported for "when" statements.

Features generally require less server implementation complexity and runtime resources than objects that use "when" statements. Features are generally static (i.e., set when a module is loaded and not changed at runtime). However, every client edit might cause a "when" statement result to change.

4.26.3. "anyxml" versus "anydata"

The "anyxml" statement MUST NOT be used to represent a conceptual subtree of YANG data nodes. The "anydata" statement MUST be used for this purpose.

4.26.4. "action" versus "rpc"

The use of "action" statements or "rpc" statements is a subjective design decision. RPC operations are not associated with any particular data node. Actions are associated with a specific data node definition. An "action" statement SHOULD be used if the protocol operation is specific to a subset of all data nodes instead of all possible data nodes.

The same action name MAY be used in different definitions within different data node. For example, a "reset" action defined with a data node definition for an interface might have different parameters than for a power supply or a VLAN. The same action name SHOULD be used to represent similar semantics.

The NETCONF Access Control Model (NACM) [RFC8341] does not support parameter-based access control for RPC operations. The user is given permission (or not) to invoke the RPC operation with any parameters. For example, if each client is only allowed to reset their own interface, then NACM cannot be used.

For example, NACM cannot enforce access control based on the value of the "interface" parameter, only the "reset" operation itself:

```
rpc reset {
  input {
    leaf interface {
      type if:interface-ref;
      mandatory true;
      description "The interface to reset.";
    }
  }
}
```

However, NACM can enforce access control for individual interface instances, using a "reset" action. If the user does not have read access to the specific "interface" instance, then it cannot invoke the "reset" action for that interface instance:

```
container interfaces {
  list interface {
    ...
    action reset { }
  }
}
```

4.27. Updating YANG Modules (Published versus Unpublished)

YANG modules can change over time. Typically, new data model definitions are needed to support new features. YANG update rules defined in [Section 11 of \[RFC7950\]](#) MUST be followed for published modules. They MAY be followed for unpublished modules.

The YANG update rules only apply to published module revisions. Each organization will have their own way to identify published work that is considered to be stable and unpublished work that is considered to

be unstable. For example, in the IETF, the RFC document is used for published work, and the I-D is used for unpublished work.

5. IANA Considerations

The following registration in the "ns" subregistry of the "IETF XML Registry" [RFC3688] was detailed in [RFC6087] and has been updated by IANA to reference this document.

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

Registrant Contact: The IESG.

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

The following assignment was detailed in [RFC6087] and has been updated by IANA in the "YANG Module Names" registry. This document has also been added as a reference for the "YANG Module Names" registry itself as it contains the template necessary for registration in [Appendix B](#).

Field	Value
Name	ietf-template
Namespace	urn:ietf:params:xml:ns:yang:ietf-template
Prefix	temp
Reference	RFC 8407

YANG Registry Assignment

6. Security Considerations

This document defines documentation guidelines for NETCONF or RESTCONF content defined with the YANG data modeling language; therefore, it does not introduce any new or increased security risks into the management system.

7. References

7.1. Normative References

- [ID-Guidelines] Housley, R., "Guidelines to Authors of Internet-Drafts", December 2010, <<https://www.ietf.org/standards/ids/guidelines/>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5378] Bradner, S., Ed. and J. Contreras, Ed., "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, DOI 10.17487/RFC5378, November 2008, <<https://www.rfc-editor.org/info/rfc5378>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.

- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", [RFC 8040](#), DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", [RFC 8342](#), DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [W3C.REC-xpath]
Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C Recommendation REC-xpath-19991116, November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.

7.2. Informative References

- [IANA-MOD-NAMES]
IANA, "YANG Module Names", <<https://www.iana.org/assignments/yang-parameters/>>.
- [IANA-XML] IANA, "IETF XML Registry", <<https://www.iana.org/assignments/xml-registry/>>.
- [RFC-STYLE]
RFC Editor, "Style Guide", <<http://www.rfc-editor.org/styleguide/>>.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", [BCP 9](#), [RFC 2026](#), DOI 10.17487/RFC2026, October 1996, <<https://www.rfc-editor.org/info/rfc2026>>.
- [RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", [RFC 4151](#), DOI 10.17487/RFC4151, October 2005, <<https://www.rfc-editor.org/info/rfc4151>>.
- [RFC4181] Heard, C., Ed., "Guidelines for Authors and Reviewers of MIB Documents", [BCP 111](#), [RFC 4181](#), DOI 10.17487/RFC4181, September 2005, <<https://www.rfc-editor.org/info/rfc4181>>.

- [RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", [RFC 6087](#), DOI 10.17487/RFC6087, January 2011, <<https://www.rfc-editor.org/info/rfc6087>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", [RFC 6991](#), DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", [RFC 7223](#), DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", [RFC 7322](#), DOI 10.17487/RFC7322, September 2014, <<https://www.rfc-editor.org/info/rfc7322>>.
- [RFC7841] Halpern, J., Ed., Daigle, L., Ed., and O. Kolkman, Ed., "RFC Streams, Headers, and Boilerplates", [RFC 7841](#), DOI 10.17487/RFC7841, May 2016, <<https://www.rfc-editor.org/info/rfc7841>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 8126](#), DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", [BCP 215](#), [RFC 8340](#), DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, [RFC 8341](#), DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", [RFC 8343](#), DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", [RFC 8349](#), DOI 10.17487/RFC8349, March 2018, <<https://www.rfc-editor.org/info/rfc8349>>.

Appendix A. Module Review Checklist

This section is adapted from RFC 4181.

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

- o I-D Boilerplate -- verify that the document contains the required I-D boilerplate (see <<https://www.ietf.org/id-info/guidelines.html>>), including the appropriate statement to permit publication as an RFC, and that the I-D boilerplate does not contain references or section numbers.
- o 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 <<https://www.ietf.org/id-info/guidelines.html>>.
- o Copyright Notice -- verify that the document has the appropriate text regarding the rights that document contributors 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:

<<https://trustee.ietf.org/license-info/>>
- o Security Considerations section -- verify that the document uses the latest approved template from the Operations and Management (OPS) area website (see <<https://trac.ietf.org/area/ops/trac/wiki/yang-security-guidelines>>) and that the guidelines therein have been followed.
- o 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].

- o References -- verify that the references are properly divided between normative and informative references, that RFCs 2119 and 8174 are included as normative references 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 okay 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). If a YANG module contains reference or "description" statements that refer to an I-D, then the I-D is included as an informative reference.
- o License -- verify that the document contains the Simplified BSD License in each YANG module or submodule. Some guidelines related to this requirement are described in [Section 3.1](#). Make sure that the correct year is used in all copyright dates. Use the approved text from the latest TLP document, which can be found at:

[<https://trustee.ietf.org/license-info/>](https://trustee.ietf.org/license-info/)

- o Other Issues -- check for any issues mentioned in [<https://www.ietf.org/id-info/checklist.html>](https://www.ietf.org/id-info/checklist.html) that are not covered elsewhere.
- o 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, including formatting advice, can be found at:

[<https://trac.ietf.org/trac/netconf/wiki>](https://trac.ietf.org/trac/netconf/wiki)
and
[<https://trac.ietf.org/trac/netmod/wiki>](https://trac.ietf.org/trac/netmod/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 B. YANG Module Template

```
<CODE BEGINS> file "ietf-template@2016-03-20.yang"

module iETF-template {
  yang-version 1.1;

  // 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://datatracker.ietf.org/wg/your-wg-name/>
     WG List:  <mailto:your-wg-name@ietf.org>

     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

// replace '2016-03-20' with the module publication date
// the format is (year-month-day)

revision 2016-03-20 {
  description
    "what changed in this revision";
  reference "RFC XXXX: <Replace With Document Title>";
}

// 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>
```

Acknowledgments

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

The working group thanks Martin Bjorklund, Juergen Schoenwaelder, Ladislav Lhotka, Jernej Tuljak, Lou Berger, Robert Wilton, Kent Watsen, and William Lupton for their extensive reviews and contributions to this document.

Author's Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com