Stream:	Internet Engineering Task Force (IETF)
RFC:	9907
BCP:	216
Obsoletes:	8407
Updates:	8126
Category:	Best Current Practice
Published:	March 2026
ISSN:	2070-1721
Authors:	A. Bierman	M. Boucadair, Ed.	Q. Wu
---	---	---	---
	YumaWorks	*Orange*	*Huawei*

RFC 9907
Guidelines for Authors and Reviewers of Documents Containing YANG Data Models

Abstract

This document provides guidelines for authors and reviewers of specifications containing YANG data models, including IANA-maintained 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 8407; it also updates RFC 8126 by providing additional guidelines for writing the IANA considerations for RFCs that specify IANA-maintained YANG modules.

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/rfc9907.

Copyright Notice

Copyright (c) 2026 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 5
   1.1. Changes Since RFC 8407 6
2. Terminology and Notation Conventions 8
   2.1. NETCONF Terms 8
   2.2. YANG Terms 9
   2.3. Network Management Datastore Architecture (NMDA) Terms 9
   2.4. Requirements Notation 9
   2.5. YANG Data Model versus YANG Module 9
3. General Documentation Guidelines 10
   3.1. Module Copyright 11
   3.2. Code Components 11
      3.2.1. Example Modules 11
   3.3. Terminology Section 11
   3.4. Tree Diagrams 12
   3.5. Narrative Sections 12
      3.5.1. YANG Module Classification 13
   3.6. Definitions Section 13
   3.7. Security Considerations Section 14
      3.7.1. Security Considerations Section Template 15
   3.8. IANA Considerations Section 18
      3.8.1. Documents That Create a New Namespace 18
      3.8.2. Documents That Extend an Existing Namespace 18
      3.8.3. Registration Templates 18

3.9. References Sections 19
3.10. Validation Tools 20
3.11. Module Extraction Tools 20
3.12. Module Usage Examples 20
4. YANG Usage Guidelines 21
4.1. Module Naming Conventions 21
4.2. Prefixes 22
4.3. Identifiers 23
4.3.1. Identifier Naming Conventions 23
4.4. Defaults 24
4.5. Conditional Statements 24
4.6. XPath Usage 27
4.6.1. XPath Evaluation Contexts 28
4.6.2. Function Library 28
4.6.3. Axes 29
4.6.4. Types 30
4.6.5. Wildcards 31
4.6.6. Boolean Expressions 31
4.7. YANG Definition Lifecycle Management 32
4.8. Module Header, Meta, and Revision Statements 33
4.9. Namespace Assignments 35
4.10. Top-Level Data Definitions 37
4.11. Data Types 37
4.11.1. Fixed-Value Extensibility 37
4.11.2. Patterns and Ranges 38
4.11.3. Enumerations and Bits 39
4.11.4. Union Types 40
4.11.5. Empty and Boolean 41
4.12. Reusable Type Definitions 42
4.13. Reusable Groupings 43

4.14. Data Definitions 43
4.14.1. Non-Presence Containers 44
4.14.2. Top-Level Data Nodes 45
4.15. Operation Definitions 45
4.16. Notification Definitions 45
4.17. Feature Definitions 46
4.18. YANG Data Node Constraints 47
4.18.1. Controlling Quantity 47
4.18.2. "must" versus "when" 47
4.19. "augment" Statements 47
4.19.1. Conditional Augment Statements 47
4.19.2. Conditionally Mandatory Data Definition Statements 48
4.20. Deviation Statements 49
4.21. Extension Statements 50
4.22. Data Correlation 50
4.22.1. Use of "leafref" for Key Correlation 51
4.23. Operational State 52
4.23.1. Combining Operational State and Configuration Data 52
4.23.2. Representing Operational Values of Configuration Data 53
4.23.3. NMDA Transition Guidelines 53
4.24. Performance Considerations 56
4.25. Open Systems Considerations 57
4.26. Guidelines for Constructs Specific to YANG 1.1 57
4.26.1. Importing Multiple Revisions 57
4.26.2. Using Feature Logic 57
4.26.3. "anyxml" versus "anydata" 57
4.26.4. "action" versus "rpc" 57
4.27. Updating YANG Modules (Published versus Unpublished) 58
4.28. Defining Standard Tags 59
4.29. Modeling Abstract Data Structures 59

4.30. IANA-Maintained YANG Modules 59
4.30.1. Context 59
4.30.2. Guidelines for IANA-Maintained YANG Modules 60
4.30.3. Guidance for Writing the IANA Considerations for RFCs Defining IANA-Maintained YANG Modules 62
5. IANA Considerations 71
5.1. YANG Modules 71
5.2. Update in YANG Parameters Registry Group 72
5.3. IANA-Maintained YANG Modules 72
5.3.1. Requirements for All Modules 72
5.3.2. Requirements Subject to Customization 73
6. Operational Considerations 73
7. Security Considerations 73
8. References 73
8.1. Normative References 73
8.2. Informative References 75
Appendix A. Module Review Checklist 78
Appendix B. Template for IETF Modules 80
Appendix C. Template for IANA-Maintained YANG Modules 81
Acknowledgments 83
Authors' Addresses 84

1. Introduction

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

This document defines a set of guidelines for documents containing YANG 1.1 [RFC7950] and YANG 1.0 [RFC6020] data models, including IANA-maintained YANG modules. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF and/or RESTCONF server. YANG is also used to define abstract data structures [RFC8791]. 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 readable and interoperable, it is desirable to define a set of descriptive 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. These guidelines can be used independent of the IETF Stream of publication or even by other organizations.

YANG 1.0 modules have to conform to [RFC6020] while YANG 1.1 modules have to conform to [RFC7950]; this document adds usage guidelines in addition to these RFCs.

Section 4.30.3 updates [RFC8126] by providing guidance for writing the IANA Considerations sections for RFCs that specify IANA-maintained YANG modules.

Note that this document is not a YANG tutorial; the reader is expected to know the YANG data modeling language before implementing the guidance in this document.

This RFC contains text intended for use as a template as designated below by the markers "`<BEGIN TEMPLATE TEXT>`" and "`<END TEMPLATE TEXT>`" or other clear designation. Such Template Text is subject to the provisions of Section 9(b) of the Trust Legal Provisions.

1.1. Changes Since RFC 8407

The following changes have been made to the guidelines published in [RFC8407]:

- Implemented the following errata reports: [Err5693], [Err5800], [Err6899], and [Err7416].
- Updated the terminology.
- Added a note about notation conventions.
- Updated the reference information of the IETF author guidelines.
- Updated the guidance so that the "file name" after the "`<CODE BEGINS>`" tag is mandatory.
- Added template markers for the security template.
- Updated the YANG security considerations template to better insist on the key secure transport features.

- Added statements that the security template is not required for modules that follow [RFC8791] or define YANG extensions such as [RFC7952].
- Added a statement about how to cite the RFCs that are listed in the security template.
- Added a template for IANA registrations.
- Added a note that folding of the examples should be done as per the conventions described in [RFC8792].
- Added a recommendation about long trees.
- Fixed a reference bug in Section 4.1.
- Added a recommendation for the use of meaningful prefix values.
- Added a note that folding of YANG modules as described in RFC 8792 can be used if and only if built-in YANG features (e.g., break line, "+") are not sufficient.
- Added tool validation checks to ensure that YANG modules fit into the line limits of an I-D.
- Added tool validation checks of JSON-encoded examples.
- Added a recommendation to ease extracting and validating examples.
- Updated many examples to be aligned with the consistent indentation recommendation (internal consistency).
- Updated the guidance for writing IANA Considerations sections to encourage registration requests to indicate whether or not a module is maintained by IANA.
- Added guidelines for IANA-maintained YANG modules.
- Added guidelines about the use of the terms "YANG module" and "YANG data model".
- Elaborated the guidance for the use of values reserved for documentation in examples.
- Recommended the use of "example:" for URI examples.
- Added a new section "Defining Standard Tags" (Section 4.28) to echo the guidance in [RFC8819].
- Recommended against the use of "case + when" construct in some cases.
- Added a discussion about the prefix pattern to use for example modules.
- Updated the NMDA guidance in the narrative text to highlight modules that are not compliant with NMDA.
- Added a new section about the classification of YANG modules.
- Fixed an inconsistency in Section 4.6.2 where the example mentions identities but uses them without their prefix as per Section 4.6.4.
- Fixed an inconsistency in Section 4.6.4 that failed to use "derived-from-or-self()" mentioned back in Section 4.6.2.
- Added a new section for modeling abstract data structures.
- Added a discussion about "must + error-message" constructs for state data.
- Added text about summary of changes in "revision" statements.
- Added a template for IANA-maintained YANG modules.
- Updated the wiki URLs to use the new structure.
- Added "anydata" to the list of statements with mandatory description(s) (Section 4.14).
- Fixed an error (invalid statements) in Section 4.24.

- Softened generic I-D authorship guidance.

2. Terminology and Notation Conventions

Some of the templates defined in the document use "--" to easily identify specific instructions to the authors. Text prefixed with "--" must not be copied as such when using a template. Note that for YANG templates, "//" is used to convey such instructions.

RFC IIII is used to refer to an RFC that defines an initial version of an IANA-maintained YANG module.

The following terms are used throughout this document:

IANA-maintained YANG module: A YANG module that is maintained by IANA and has an IANA registry associated with it (e.g., "iana-tunnel-type" [RFC8675] or "iana-pseudowire-types" [RFC9291]).

Once an IANA-maintained YANG module is initialized, new values are not directly added to the module. These values are instead added to the companion registry.

IETF module: A YANG module that is published by the IETF and that is not maintained by IANA.

published: A stable release of a module or submodule. For example, the Request for Comments Series described in Section 2.1 of [RFC2026] is considered a stable publication.

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 work in progress, subject to change at any time.

YANG fragment: A set of YANG statements that is 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.

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:

- capability
- client
- protocol operation (or simply "operation")
- server

2.2. YANG Terms

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

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

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

2.3. Network Management Datastore Architecture (NMDA) Terms

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

- configuration
- conventional configuration datastore
- datastore
- operational state
- 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.

2.5. YANG Data Model versus YANG Module

Both [RFC6020] and [RFC7950] make a distinction between the following concepts:

data model: Describes how data is represented and accessed.

YANG structures data models into modules for ease of use [RFC8309].

module: Defines hierarchies of schema nodes to make a self-contained and compilable block of YANG definitions and inclusions.

A YANG module is typically a single ".yang" file, starting with a "module" statement.

A YANG module may include any number of submodules that are stored in separate ".yang" files starting with a "submodule" statement. Regardless of the presence of submodules, the module and its submodules are externally viewed as a single YANG module.

A YANG data model can consist of:

1. a single YANG module (e.g., [RFC9129]) or
2. multiple YANG modules (e.g., [RFC7407]).

Note that the term "YANG model" is sometimes used as an abbreviation of "YANG data model". However, that term should be avoided in favor of "YANG data model". Likewise, "YANG data module" has no meaning and must be avoided.

Even if a YANG data model is structured as a single YANG module, the term "YANG data model" should be used in the title, abstract, and in the body of the document where the overall design is described. "YANG module" should be used when a specific "*.yang" file is referenced. Likewise, "YANG module" should be used when using terms related to YANG module specifications (e.g., augmentation or deviation). However, when extending the concepts embodied in a YANG module, authors should refer to those as an extension to the "YANG data model".

3. General Documentation Guidelines

YANG modules being considered for publication in an RFC are contained in Internet-Drafts (I-Ds). Guidelines for authoring an I-D can be found at [ID-Guidelines]. These guidelines are not repeated here.

The following sections **MUST** be present in an I-D or RFC containing a YANG module:

- Narrative sections (Section 3.5)
- A Definitions section(s) (Section 3.6)

Additional YANG-specific considerations **MUST** be included for the following sections:

- Security Considerations (Section 3.7)
- IANA Considerations (Section 3.8)
- References (Section 3.9)

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

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

The guidelines in this document refer mainly to a normative module or submodule, but they 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 at: <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 **MUST** 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 FFFF: 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. However, example modules **MUST** be validated (Section 3.10).

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]. Tree diagrams longer than one page **SHOULD** be included in an appendix, i.e., not in the main body of the document.

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 sections **MUST** include an overview section that describes the scope and field of application of the data model(s) defined by the specification and that specifies the relationship (if any) of these data models to other standards, particularly to standards containing other YANG data models. The narrative part **SHOULD** include one or more sections to briefly describe the structure of the data models 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 [RFC9911]) or is 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 major Network Management Datastore Architecture (NMDA) exceptions or includes a temporary non-NMDA module [RFC8342], then the Introduction section **SHOULD** mention this fact with the reasoning that motivated that design. Refer to Section 4.23 for more NMDA-related guidance. Specifically, Section 4.23.2 includes a recommendation for designers to describe and justify any NMDA exceptions in detail as part of the module itself.

Consistent indentation **SHOULD** be used for all examples, including YANG fragments and protocol message instance data. If line wrapping is used for formatting purposes, then this **SHOULD** be indicated per the guidance in [RFC8792], as shown in the following example:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<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>
```

Built-in YANG features (e.g., breaking line, "+") **SHOULD** be used to fit a module into the line limits. Exceptionally, YANG modules **MAY** be folded as described in RFC 8792 if and only if built-in YANG features are not sufficient. A similar approach (e.g., using "--tree-line-length 69" or splitting a tree into subtrees) **SHOULD** be followed for tree diagrams.

3.5.1. YANG Module Classification

The narrative section **SHOULD** include a mention of the classification of a given model. Such a mention is meant to ease positioning the module in the overall operational ecosystem. Specifically, the following types from [RFC8309] and [RFC8969] can be used:

Service Model: Describes a service and the parameters of the service in a portable way that can be used uniformly and independent of the equipment and operating environment.

Examples of service models are the L3VPN Service Model (L3SM) [RFC8299] and the L2VPN Service Model (L2SM) [RFC8466].

Network Model: Describes a network-level abstraction (or a subset of aspects of a network infrastructure), including devices and their subsystems, and relevant protocols operating at the link and network layers across multiple devices. This model corresponds to the network configuration model discussed in [RFC8309].

This model can be used by a network operator to allocate resources (e.g., a tunnel resource or a topology resource) for the service or to schedule resources to meet the service requirements defined in a service model.

Examples of network models are the L3VPN Network Model (L3NM) [RFC9182] or the L2VPN Network Model (L2NM) [RFC9291].

Device Model: Refers to the Network Element YANG data model described in [RFC8199] or the device configuration model discussed in [RFC8309].

Device models are also used to model a function embedded in a device (e.g., Access Control Lists (ACLs) [RFC8519]).

A non-comprehensive list of device models is provided in Appendix A.4.4 of [RFC8969].

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 YANG Independent Notation (YIN) syntax version (Section 13 of [RFC7950]) 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 key words from [RFC2119] and [RFC8174].

Consistent indentation and formatting (e.g., folding) **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.

Unless the modules comply with [RFC8791] or define YANG extensions (e.g., [RFC7952]), the security section **MUST** be modeled after the latest approved template (available at <https://wiki.ietf.org/group/ops/yang-security-guidelines>). Section 3.7.1 contains the security considerations template. Authors **MUST** check the web page at the URL listed above in case there is a more recent version available.

In particular:

- Writable data nodes that could be especially disruptive if abused **MUST** be explicitly listed by name, and the associated security risks **MUST** be explained.
- Readable data nodes that contain especially sensitive information or that raise significant privacy concerns **MUST** be explicitly listed by name, and the reasons for the sensitivity/privacy concerns **MUST** be explained.
- Operations (i.e., YANG "rpc" statements) that are potentially harmful to system behavior or that raise significant privacy concerns **MUST** be explicitly listed by name, and the reasons for the sensitivity/privacy concerns **MUST** be explained.

Documents that exclusively define modules that follow the extension in [RFC8791] are not required to include the security template in Section 3.7.1. Likewise, following the template is not required for modules that define YANG extensions such as [RFC7952].

3.7.1. Security Considerations Section Template

```
<BEGIN TEMPLATE TEXT>

X.  Security Considerations

This section is modeled after the template described in Section 3.7.1
of [RFC9907].

The "<module-name>" YANG module defines a data model that is
designed to be accessed via YANG-based management protocols,
such as the Network Configuration Protocol (NETCONF) [RFC6241]
and RESTCONF [RFC8040].  These YANG-based management protocols
(1) have to use a secure transport layer (e.g., Secure Shell (SSH)
[RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and (2) have to use
mutual authentication.

The Network Configuration Access Control Model (NACM) [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.

-- Note: RFC 8341 (or a future RFC that replaces it) MUST be listed
-- as a normative reference.

-- By default, RFC 4252, RFC 6241, RFC 8040, RFC 8446, RFC 9000, and
-- RFC 9907 (or future RFCs that replace any of them) are listed as
-- informative references unless normatively cited in other sections
-- of the document that specifies the YANG module.

-- Writable nodes section:
--
-- If the data model contains any writable data nodes (those are all
-- the "config true" nodes), then include the following text:

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).  All writable data nodes are likely to be sensitive or
vulnerable in some network environments.  Write
operations (e.g., edit-config) and delete operations to these data
nodes without proper protection or authentication can have a negative
effect on network operations.  The following subtrees and data nodes
have particular sensitivities/vulnerabilities:

-- If the data model contains any particularly sensitive data nodes,
-- e.g., ones that are protected by a "nacm:default-deny-write"
-- or a "nacm:default-deny-all" extensions statement, then those
-- subtrees and data nodes must be listed, with an explanation of the
-- associated security risks with a focus on how they can be
-- disruptive if abused.  Otherwise, state:
--
-- "There are no particularly sensitive writable data nodes."

-- Readable nodes section:
--
-- If the data model contains any readable data nodes (i.e., those
-- that are "config false" nodes, but also all other nodes because
-- they can also be read via operations like get or get-config), then
-- include the following text:
```

```
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.  Specifically, the following
subtrees and data nodes have particular sensitivities/
vulnerabilities:

-- You must evaluate whether the data model contains 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) that are particularly sensitive or vulnerable
-- (e.g., if they might reveal customer information or violate
-- personal privacy laws).  Typically, particularly sensitive
-- readable data nodes are ones that are protected by a
-- "nacm:default-deny-read" or a "nacm:default-deny-all" extensions
-- statement.
--
-- Otherwise, state:
-- "There are no particularly sensitive readable data nodes."

-- RPC/action operations section:
--
-- If the data model contains any RPC or action operations, then
-- include the following text:

Some of the RPC or action 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.
Specifically, the following operations have particular
sensitivities/ vulnerabilities:

-- If the data model contains any particularly sensitive RPC
-- or action operations, then those operations must be listed
-- here, along with an explanation of the associated specific
-- sensitivity or vulnerability concerns.
--
-- Otherwise, state:
-- "There are no particularly sensitive RPC or action operations."

-- Reusable groupings from other modules section:
--
-- If the data model reuses groupings from other modules and
-- the document that specifies these groupings also
-- includes those as data nodes, then add this text as a
-- reminder of the specific sensitivity or vulnerability of
-- reused nodes.

This YANG module uses groupings from other YANG modules that
define nodes that may be considered sensitive or vulnerable
in network environments.  Refer to the Security Considerations
of <RFC-insert-numbers> for information as to which nodes may
be considered sensitive or vulnerable in network environments.

-- No data nodes section:
--
-- If the data model does not define any data nodes (i.e., none
-- of the above sections or readable/writable data nodes or RPCs
```

```
-- have been included), then add the following text:

The YANG module defines a set of identities, types, and
groupings.  These nodes are intended to be reused by other YANG
modules.  The module by itself does not expose any data nodes that
are writable, data nodes that contain read-only state, or RPCs.
As such, there are no additional security issues related to
the YANG module that need to be considered.

Modules that use the groupings that are defined in this document
should identify the corresponding security considerations.  For
example, reusing some of these groupings will expose privacy-related
information (e.g., 'node-example').

<END TEMPLATE TEXT>
```

3.8. IANA Considerations Section

Each normative YANG module **MUST** be registered in both the "IETF XML Registry" group [RFC3688] [IANA-XML] and the "YANG Module Names" registry [RFC6020] [IANA-MOD-NAMES]. The registration request in the "YANG Module Names" registry should indicate whether or not the module is IANA-maintained. 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.

Additional IANA considerations applicable to IANA-maintained YANG modules (including instructions to maintain them) are provided in Section 4.30.3.

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" registry within the "IETF XML Registry" registry group **MUST** be requested from the IANA.

A registration template for new YANG modules is provided in Section 3.8.3.1.

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.

3.8.3. Registration Templates

3.8.3.1. IANA Template for Documents Defining New YANG Modules

A registration template for a new module is provided below:

```
IANA is requested to register the following URI in the "ns"
registry within the "IETF XML Registry" group [RFC3688]:

   URI:
   Registrant Contact:  The IESG
   XML: N/A; the requested URI is an XML namespace.

IANA is requested to register the following YANG module in the "YANG
Module Names" registry [RFC6020] within the "YANG Parameters"
registry group.

   Name:
   Maintained by IANA?  Y/N
   Namespace:
   Prefix:
   Reference:
```

3.8.3.2. IANA Template for Revising YANG Modules

A registration template for a revised module is provided below:

```
IANA is requested to update the following registration in the "ns"
registry within the "IETF XML Registry" group [RFC3688] to
reference this document:

   URI:
   Registrant Contact:  The IESG
   XML: N/A; the requested URI is an XML namespace.

IANA is requested to register the following YANG module in the "YANG
Module Names" registry [RFC6020] [RFC9890] within the "YANG Parameters"
registry group.

   Name:
   Maintained by IANA?  Y/N
   Namespace:
   Prefix:
   Reference:
```

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.

To ensure that a module fits into the line limits of an I-D, the command "pyang -f yang --keep-comments --yang-line-length 69" should be used.

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.

To check that JSON-encoded examples [RFC7951] comply with the target data models, programs such as 'yangson' or 'yanglint' should be used. Both programs are freely available from GitHub: <https://github.com/CZ-NIC/yangson> and <https://github.com/CESNET/libyang>.

3.11. Module Extraction Tools

A version of 'rfcstrip' that will extract YANG modules from an I-D or RFC 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). Examples that are meant to illustrate a valid data instance **MUST** be validated (Section 3.10). Refer to Section 3.10 for tools that validate YANG modules and examples. If IP addresses/prefixes are used, then a mix of either IPv4 and IPv6 addresses/prefixes or IPv6 addresses/prefixes exclusively **SHOULD** be used in the examples.

For some types (IP addresses, domain names, etc.), the IETF has reserved values for documentation use. Authors **SHOULD** use these reserved values in the usage examples if these types are used. Examples of reserved values are listed below:

- IPv4 and IPv6 addresses/prefixes reserved for documentation are defined in [RFC5737], [RFC3849], and [RFC9637].
- The Enterprise Number 32473 reserved for documentation use is defined in [RFC5612].
- Autonomous System Numbers (ASNs) reserved for documentation use are defined in [RFC5398].
- Reserved domain names for documentation are defined in [RFC2606].

URI examples **SHOULD** be prefixed with "example:".

In order to ease extraction and validation of examples, it is **RECOMMENDED** to use code markers.

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.

A template for IETF modules is provided in Appendix B.

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 [RFC6020].

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 (Section 14 of [RFC6020]). 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 module name prefix be selected that represents the entire organization. All normative YANG modules published by the IETF **MUST** begin with the prefix "ietf-". All IANA-maintained YANG modules **MUST** begin with the prefix "iana-". 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 the same name to be used in multiple modules, 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:

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

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

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

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

Consistent with Section 7.1.4 of [RFC7950], the prefix defined by a module **SHOULD** be used when the module is imported, unless there is a conflict.

Prefix values **SHOULD** be short but meaningful to the intended user. Prefix values **SHOULD NOT** conflict with known modules that have been previously published.

For convenience, prefix values of example modules **SHOULD** be prefixed with "ex" or similar patterns. In doing so, readers of example modules or tree diagrams that mix both example and standard modules can easily identify example parts.

4.3. Identifiers

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.

List identifiers **SHOULD** be singular with the surrounding container name plural. Similarly, "leaf-list" identifiers **SHOULD** be singular.

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 substatements listed in Table 1 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

Table 1: 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 leafs 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.

Some modules use a "case + when" construct but provide duplicated information (e.g., the "when" statements are constraining a single case in the choice as shown in the example below). Such constructs with duplicated information **SHOULD NOT** be used.

```
leaf type {
  type enumeration {
    enum a;
    enum b;
    enum c;
  }
  mandatory true;
}
choice type-choice {
  case b {
    container type-b {
      when "../type = 'b'";
      leaf foo {
        type string;
      }
    }
  }
  case c {
    container type-c {
      when "../type = 'c'";
      leaf bar {
        mandatory true;
        type string;
      }
    }
  }
}
```

The following example removes the duplicated information:

```
     leaf type {
       type enumeration {
         enum a;
         enum b;
         enum c;
       }
       mandatory true;
     }
     container type-b {
       when "../type = 'b'";
       leaf foo {
         type string;
       }
     }
     container type-c {
       when "../type = 'c'";
       leaf bar {
         mandatory true;
         type string;
       }
     }
```

Note that the use of "case + when" is still useful in cases where complementary modeling constraints should be expressed. See the example provided below:

```
    leaf type {
      type enumeration {
        enum a;
        enum b;
        enum c;
      }
    }
    choice second-type {
      mandatory true;
      case foo {
        container foo {
          presence "When present, indicates type foo";
          leaf foo-attribute {
            type string;
          }
        }
      }
      case b {
        container bar {
          when "../type = 'a' or ../type = 'b'";
          presence "When present, indicates type bar";
          leaf bar-attribute {
            type string;
          }
        }
      }
      case c {
        container baz {
          when "../type = 'c'";
          leaf baz-attribute {
            mandatory true;
            type string;
          }
        }
      }
    }
```

Section 8.1 of [RFC7950] includes provisions for defining constraints on state data and specifies that a constraint must be true in a valid state data tree. However, Section 5.3 of [RFC8342] softens that behavior by allowing semantic constraints to be violated under some circumstances to help to detect anomalies. Relaxing validation constraints on state data is meant to reveal deviations of the observed behavior versus intended behavior of a managed entity and hopefully trigger corrective actions by a management system. From that perspective, it is **RECOMMENDED** to avoid defining constraints on state data that would hinder the detection by a management system of abnormal behaviors of a managed entity.

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 XPath execution environment **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 XPath execution environment **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" statement 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:

```
// assume "ex" is the prefix of the module where the identity
// name-format-null is defined

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

// this is preferred
when "derived-from-or-self(md-name-format, 'ex: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 "derived-from-or-self(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" statement 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:foo1 = 'one' or /f:foo1 = 'three'";
    type string;
  }
```

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

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

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

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 after the publication date 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 9911: 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. Exceptions include (but are not limited to): example modules, IANA-maintained YANG modules, or modules contained in AD-sponsored documents. 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, and **MUST** contain the following statement:

> All revisions of IETF and IANA published modules can be found at the "YANG Parameters" registry group: <https://www.iana.org/assignments/yang-parameters>.

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. For convenience, the description text of a new published revision may summarize any changes made to a module compared to the previous published revision. Typically, that list is a YANG-specific subset of the summary of changes listing any changes made from the RFC being updated or obsoleted as per [ID-Guidelines].

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

```
revision 2010-09-24 {
  description
    "Initial revision.";
reference
  "RFC 6021: Common YANG Data Types";
}
```

The following example shows the "revision" statements for a published YANG module that updates a published module. The new "revision" statement summarizes the changes compared to the previous published revision.

```
revision 2013-07-15 {
  description
    "This revision adds the following new data types:
     - yang:yang-identifier
     - yang:hex-string
     - yang:uuid
     - yang:dotted-quad";
  reference
    "RFC 6991: Common YANG Data Types";
}

revision 2010-09-24 {
  description
    "Initial revision.";
reference
  "RFC 6021: Common YANG Data Types";
}
```

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 "revision" statements for an unpublished update to a published YANG module. The latest "revision" statement of the unpublished module summarizes the changes compared to the previous revision.

```
revision 2025-12-22 {
  description
    "This revision adds the following new data types:
     - yang:date
     - yang:date-no-zone
     - yang:time
     - yang:time-no-zone
     - yang:hours32
     - yang:minutes32
     - yang:seconds32
     - yang:centiseconds32
     - yang:milliseconds32
     - yang:microseconds32
     - yang:microseconds64
     - yang:nanoseconds32
     - yang:nanoseconds64
     - yang:language-tag
     The yang-identifier definition has been aligned with YANG
     1.1 and types representing time support the representation
     of leap seconds.  The representation of time zone offsets
     has been aligned with RFC 9557.  Several description and
     pattern statements have been improved.";
  reference
    "RFC 9911: Common YANG Data Types";
 }

revision 2013-07-15 {
  description
    "This revision adds the following new data types:
     - yang:yang-identifier
     - yang:hex-string
     - yang:uuid
     - yang:dotted-quad";
   reference
     "RFC 6991: Common YANG Data Types";
 }

 revision 2010-09-24 {
   description
     "Initial revision.";
 reference
   "RFC 6021: Common YANG Data Types";
 }
```

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:

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

Until a URI is assigned by the IANA, a proposed namespace URI **MUST** be provided for the "namespace" statement in a YANG module. A value **SHOULD** be selected that is not likely to collide with other YANG namespaces. Standard module names, prefixes, and URI strings already listed in the "YANG Module Names" registry group **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 Section 5.3 of [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 I-Ds from the IETF Stream. 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 (e.g., IANA), then an "enumeration" data type **SHOULD** be used.

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

If distributed extensibility or hierarchical organization 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 [RFC9911] demonstrates the proper use of the "pattern" statement:

```
typedef ipv6-address-no-zone {
  type inet:ipv6-address {
    pattern '[0-9a-fA-F:\.]*';
  }
  ...
}
```

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 [RFC9911] 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 [RFC9911] 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 leafs, but a default statement is often used for plain boolean leafs. 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 [RFC9911], 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:

- Clearly identify the purpose of the grouping in the "description" statement.
- 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.
- Do not reference data outside the grouping in any "path", "must", or "when" statements.
- Do not include a "default" substatement on a leaf or choice unless the value applies on all possible contexts.
- Do not include a "config" substatement on a data node unless the value applies on all possible contexts.
- 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:

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

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

The "anyxml" construct may be useful to represent an HTML banner containing markup elements, such as `"<b>"` and `"</b>"`, 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:

- top-level siblings are not ordered
- top-level siblings are not static and depend on the modules that are loaded
- for subtree filtering, retrieval of a top-level leaf-list will be treated as a content-match node for all top-level-siblings
- 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" statement 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 [RFC8343] 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 sample, it is okay to augment the "interface" entry with "mandatory-leaf" because the augmentation depends on support for "some-new-iftype". The old client does not know about this type, so it would never select this type; 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 Section 7.20.3 of [RFC7950], 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:

- The semantics of the extension **MUST NOT** contradict any YANG statements. Extensions can add semantics not covered by the normal YANG statements.
- 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.
- The extension **MUST** clearly identify where it can be used within other YANG statements.
- 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:

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

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

mirror: create new objects in a new module or the original module, except use 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 leafs 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 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 module 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 module or a module 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 modules, the module should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" module in [RFC7223] has been restructured as an NMDA-compatible module in [RFC8343] (which obsoletes [RFC7223]). The "/interfaces-state" hierarchy has been marked with "status deprecated". Modules 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 modules that augment modules that have not been structured with the NMDA, the modeler will have to consider the structure of the base module and the guidelines listed above. Where possible, such modules should move to new revisions of the base module that are NMDA compatible. When that is not possible, augmenting "state" containers **SHOULD** be avoided, with the expectation that the base module will be re-released with the state containers marked as "deprecated". It is **RECOMMENDED** to augment only the "/foo" hierarchy of the base module. Where this recommendation cannot be followed, 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 module from an NMDA module, the following steps can be taken:

- Change the module name by appending "-state" to the original module name.
- Change the namespace by appending "-state" to the original namespace value.
- Change the prefix by appending "-s" to the original prefix value.
- Add an import to the original module (e.g., for typedef definitions).

- 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.
- 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: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: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: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: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:

- Lists are generally more expensive than containers
- "when" statement evaluation is generally more expensive than "if-feature" or "choice" statements
- "must" statements are generally more expensive than "min-elements", "max-elements", "mandatory", or "unique" statements

- "identityref" leafs are generally more expensive than "enumeration" leafs
- "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 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, an RFC is used for published work, and an I-D is used for unpublished work.

4.28. Defining Standard Tags

[RFC8819] specifies a method for associating tags with YANG modules. Tags may be defined and associated at design time, at implementation time, or via user administrative control. Design-time tags are indicated using the module-tag "extension" statement.

A module **MAY** indicate, using module-tag "extension" statements, a set of tags that are to be automatically associated with it (i.e., not added through configuration).

```
module example-module {
  namespace "https://example.com/yang/example";
  prefix "ex";
  //...
  import module-tags { prefix tags; }

  tags:module-tag "ietf:some-new-tag";
  tags:module-tag "ietf:some-other-tag";
  // ...
}
```

Authors can use existing standard tags or use new tags defined in the model definition, as appropriate. For IETF modules, new tags **MUST** be assigned in the IANA "IETF YANG Module Tags" registry within the "YANG Module Tags" registry group [IANA-TAGS].

4.29. Modeling Abstract Data Structures

For contexts where YANG is used to model abstract data structures (e.g., protocol messages), the use of the "structure" extension statement [RFC8791] is **RECOMMENDED** compared to the "yang-data" extension statement [RFC8040]. Examples of modules that rely upon the "structure" extension statement from [RFC8791] can be found in [RFC9132] or [RFC9195].

Abstract data structures can be augmented using the "augment-structure" statement [RFC8791]. Examples of modules that augment abstract data structures can be found in [RFC9244] and [RFC9362].

4.30. IANA-Maintained YANG Modules

4.30.1. Context

IANA maintains a set of registries that are key for interoperability. The content of these registries is usually available using various formats (e.g., plain text or XML). However, there was some confusion in the past about whether the content of some registries is dependent on a specific representation format. For example, Section 5 of [RFC8892] was published to clarify that MIB and YANG modules are merely additional formats in which the "Interface Types (ifType)" and "Tunnel Types (tunnelType)" registries are available. The MIB [RFC2863] and YANG modules ([RFC7224] [RFC8675]) are not separate registries, and the same values are always present in all formats of the same registry.

A design in which a YANG module includes parameters and values directly in a module that is not maintained by IANA while these are populated in an IANA registry could lead to ambiguity and maintain stale information. Such a design creates another source of information that may deviate from the IANA registry as new values are assigned or some values are deprecated.

For the sake of consistency and the ability to support new values while maintaining IANA registries as the unique authoritative source of information, this document recommends the use of IANA-maintained YANG modules as the single source of information.

The following section provides a set of guidelines for YANG module authors related to the design of IANA-maintained YANG modules. These guidelines are meant to leverage existing IANA registries and use YANG as another format to present the content of these registries when appropriate.

4.30.2. Guidelines for IANA-Maintained YANG Modules

When designing a YANG module for a functionality governed by a protocol for which IANA maintains a registry, it is **RECOMMENDED** to specify an IANA-maintained YANG module that echoes the content of that registry. This is superior to including that content in an IETF-maintained module.

When one or multiple registries are available under the same registry group, it is **RECOMMENDED** to define an IANA-maintained YANG module for each registry. However, module designers **MAY** consider defining one single IANA-maintained YANG module that covers all registries if maintaining that single module is manageable (e.g., very few values are present or expected to be present for each registry). An example of such a module is documented in Section 5.2 of [RFC9132].

An IANA-maintained YANG module may use the "identityref" data type approach (e.g., [RFC8675]) or an "enumeration" data type approach (e.g., [RFC9108]). See Section 4.11.1 for a guidance on which data type to use. The decision about which type to use should be made based upon specifics related to the intended use of the IANA-maintained YANG module. For example, identities are useful if the registry entries are organized hierarchically, possibly including multiple inheritances. The reasoning for the design choice **MUST** be documented in the companion specification that registers an IANA-maintained YANG module. For example, [RFC9244] defines an IANA-maintained YANG module that uses enumerations for the following reason:

> The DOTS telemetry module (Section 11.1) uses "enumerations" rather than "identities" to define units, samples, and intervals because otherwise the namespace identifier "ietf-dots-telemetry" must be included when a telemetry attribute is included (e.g., in a mitigation efficacy update). The use of "identities" is thus suboptimal from the standpoint of message compactness, as message compactness is one of the key requirements for DOTS signal channel messages.

Designers of IANA-maintained YANG modules **MAY** supply the initial full version of the module in a specification document that registers the module or only a script to be used (including by IANA) for generating the module (e.g., an Extensible Stylesheet Language Transformations (XSLT) stylesheet as in Appendix A of [RFC9108] or a Python script as in [RFC9645]). For both cases, the document that defines an IANA-maintained YANG module **MUST** include a note indicating that the document is only documenting the initial version of the module and that the authoritative version is to be retrieved from the IANA registry. Also, the IANA-maintained module **MUST** include the following note indicating the RFC that registered the initial version of the IANA-maintained YANG module:

> The initial version of this YANG module is part of RFC IIII; see the RFC itself for full legal notices.

It is **RECOMMENDED** to include the URL from where to retrieve the recent version of the module. When a script is used, the Internet-Draft that defines an IANA-maintained YANG module has to include an appendix with the full script and **SHOULD** include an appendix with the initial full version of the module. Including such an appendix in Internet-Drafts is meant to assess the correctness of the outcome of the supplied script. The authors **MUST** include a note to the RFC Editor requesting that the appendix with the initial version of the module be removed before publication as RFC and that RFC IIII is replaced with the RFC number that is assigned to the document. Initial versions of IANA-maintained YANG modules that are published in RFCs may be misused despite the appropriate language to refer to the IANA registry to retrieve the up-to-date module. This is problematic for interoperability, e.g., when values are deprecated or are associated with a new meaning.

> Note: [Style] provides XSLT 1.0 stylesheets and other tools for translating IANA registries to YANG modules. The tools can be used to generate up-to-date revisions of an IANA-maintained YANG module based upon the XML representation of an IANA registry.

If an IANA-maintained YANG module is imported by another module, a normative reference with the IANA URL from which to retrieve the IANA-maintained YANG module **SHOULD** be included. Although not encouraged, referencing the RFC that defines the initial version of the IANA module is acceptable in specific cases (e.g., the imported version is specifically the initial version, the RFC includes useful description about the usage of the module).

Examples of IANA URLs from which to retrieve the latest version of an IANA-maintained YANG module are as follows:

- https://www.iana.org/assignments/iana-bgp-l2-encaps,
- https://www.iana.org/assignments/iana-pseudowire-types, and
- https://www.iana.org/assignments/iana-bfd-types.

"IANA_FOO_URL" is used in the following to refer to such URLs. These URLs are expected to be sufficiently permanent and stable.

Whenever referencing a specific version of an IANA-maintained YANG module is needed, then URLs such as the following are used:

- https://www.iana.org/assignments/iana-bgp-l2-encaps@2022-09-20.yang

"IANA_FOO_URL_With_REV" is used in the following to refer to such URLs.

A template for IANA-maintained YANG modules is provided in Appendix C.

4.30.3. Guidance for Writing the IANA Considerations for RFCs Defining IANA-Maintained YANG Modules

In addition to the IANA considerations in Section 3.8, the IANA Considerations section of an RFC that includes an IANA-maintained YANG module **MUST** provide the required instructions for IANA to automatically perform the maintenance of that IANA module. These instructions describe how to proceed with updates to the IANA-maintained YANG module that are triggered by a change to the authoritative registry. Concretely, the IANA Considerations section **SHALL** at least provide the following information:

- A request to IANA to add a note to the page displaying the information about the IANA-maintained YANG module that new values must not be directly added to the module. These values should be added to an authoritative IANA registry.
- A request to IANA to add a note to the authoritative IANA registry to indicate that any change to the registry must be reflected into the corresponding IANA-maintained YANG module. That is, any changes to the registry must be accompanied by an update to the corresponding IANA-maintained YANG module.
- Details about the required actions (e.g., add a new "identity" or "enum" statement) to update the IANA-maintained YANG module to reflect changes to an authoritative IANA registry. Typically, these details have to include the procedure to create a new "identity" statement name and substatements ("base", "status", "description", and "reference") or a new "enum" statement and substatements ("value", "status", "description", and "reference").
  - When creating a new "identity" statement name or a new "enum" statement, it is **RECOMMENDED** to use the same name (if present) as recorded in the IANA registry.
  - If the name in the IANA registry does not comply with the naming conventions listed in Section 4.3.1, the procedure **MUST** detail how IANA can generate legal identifiers from such a name. Specifically, if the name begins with a number, it is **RECOMMENDED** to spell out (i.e., not use a digit) the number when used as an identifier. IANA should be provided with instructions to perform such a task. For example, authors of a module with such identifiers have to indicate whether:
    - "3des-cbc" should be "three-des-cbc" or rather "triple-des-cbc" to be consistent with Section 6.3 of [RFC4253].
    - "6to4" should be "sixToFour" as in [RFC7224] or "sixtofour" as in [RFC8675].

- If a new registration uses an identifier that does not comply with the naming conventions listed in Section 4.3.1, IANA should check if guidance to generate legal identifiers was supplied in the RFC that specified the initial version of the module. If no such guidance is available, IANA should check the latest revision of the IANA-maintained YANG module for similar patterns. If all else fails, IANA should seek advice from relevant registry experts (e.g., designated experts for a registry using the Expert Review policy (Section 4.5 of [RFC8126]) or responsible area director).

The IANA Considerations Section **MAY** also provide the following information if a default action is to be overridden:

- A note whether unassigned or reserved values should be present in the IANA-maintained YANG module. If no instruction is provided, unassigned or reserved values must not be present in the IANA-maintained YANG module.
- An instruction whether experimental values should be included in the IANA-maintained YANG module. If no instruction is provided, experimental values **MUST NOT** be listed in the IANA-maintained YANG module.
- An instruction about how to generate the "revision" statement. If no instruction is provided, default actions provided in Section 5.3 will be followed.

A template for the IANA Considerations is provided in Section 4.30.3.1 for IANA-maintained YANG modules with identities and Section 4.30.3.2 for IANA-maintained YANG modules with enumerations. Authors may modify the template to reflect specifics of their modules (e.g., multiple registries can be listed for a single IANA-maintained YANG module, no explicit description (or name) field is listed under the authoritative IANA registry, or the name does not comply with YANG naming conventions (Section 4.3.1)).

An example of "revision" statements that are generated following the guidance in Section 5.3.2 is provided below:

```
revision 2023-11-27 {
  description
    "Registered RR Type RESINFO 261.";
  reference
    "https://www.iana.org/assignments/yang-parameters/"
  + "iana-dns-class-rr-type@2023-11-27.yang";
}

revision 2023-11-08 {
  description
    "Updated description and replaced draft string reference to
     64 and 65 with RFC 9460: Service Binding and Parameter
     Specification via the DNS (SVCB and HTTPS Resource Records).";
  reference
    "RFC 9460: Service Binding and Parameter Specification via the
               DNS (SVCB and HTTPS Resource Records)
     https://www.iana.org/assignments/yang-parameters/"
  + "iana-dns-class-rr-type@2023-11-08.yang";
}

revision 2023-04-25 {
  description
    "Updated reference for 64 and 65.";
  reference
    "https://www.iana.org/assignments/yang-parameters/"
  + "iana-dns-class-rr-type@2023-04-25.yang";
}

revision 2022-05-30 {
  description
    "Updated description, reference for 64 and 65.";
  reference
    "https://www.iana.org/assignments/yang-parameters/"
  + "iana-dns-class-rr-type@2022-05-30.yang";
}

revision 2021-08-31 {
  description
    "Initial revision.";
  reference
    "RFC 9108: YANG Types for DNS Classes and Resource Record
               Types";
}
```

Duplicating the same reference at the high level and at the level of a new addition might be redundant. For example, the following does not provide access to a specific (OLD) revision of the module when future revisions are made [IANA_Tunnel_Type_URL]:

```
revision 2021-04-23 {
  description
    "Registered tunnelType 19.";
  reference
    "RFC 4301: Security Architecture for the Internet Protocol";
}

revision 2019-11-16 {
  description
    "Initial revision.";
  reference
    "RFC 8675: A YANG Data Model for Tunnel Interface Types";
}

...

identity ipsectunnelmode {
  base ift:tunnel;
  description
    "IpSec tunnel mode encapsulation.";
  reference
    "RFC 4301: Security Architecture for the Internet Protocol";
}
```

The following example shows how to generate the "revision" statements following the guidance in Section 5.3.2:

```
revision 2021-04-23 {
  description
    "Registered tunnelType 19.";
  reference
    "https://www.iana.org/assignments/yang-parameters/"
  + "iana-tunnel-type@2021-04-23.yang
     RFC 4301: Security Architecture for the Internet Protocol";
}

revision 2019-11-16 {
  description
    "Initial revision.";
  reference
    "RFC 8675: A YANG Data Model for Tunnel Interface Types";
}
...
identity ipsectunnelmode {
  base ift:tunnel;
  description
    "IpSec tunnel mode encapsulation.";
  reference
    "RFC 4301: Security Architecture for the Internet Protocol";
}
```

The templates in the following subsections are to be considered in addition to the required information that is provided in Section 3.8.

4.30.3.1. Template for IANA-Maintained YANG Modules with Identities

```
<BEGIN TEMPLATE TEXT>

This document defines the initial version of the IANA-maintained
"iana-foo" YANG module.  The most recent version of the YANG module
is available from the "YANG Parameters" registry group
[IANA-YANG-PARAMETERS].

IANA is requested to add this note to the registry:

   New values must not be directly added to the "iana-foo" YANG
   module.  They must instead be added to the "foo" registry.

IANA is requested to add this note to [reference-to-the-iana-foo-
registry]:

   When this registry is modified, the YANG module "iana-foo"
   [IANA_FOO_URL] must be updated as defined in RFC IIII.

When a value is added to the "foo" registry, a new "identity"
statement needs to be added to the "iana-foo" YANG module.
The name of the "identity" MUST be the name as provided in the
registry.  The "identity" statement should have the following
substatements defined:

 "base":        Contains 'name-base-identity-defined-in-foo'.

 "status":      Include only if a registration has been deprecated or
                obsoleted.  IANA "deprecated" maps to YANG status
                "deprecated", and IANA "obsolete" maps to YANG status
                "obsolete".

 "description":  Replicates the description from the registry.

 "reference":   Replicates the reference(s) from the registry with
                the title of the document(s) added.

-- Optional:

-- Include only text that needs to be customized for the module.
-- Text that does not require customization should be
-- omitted.

-- Notes tagged with "--" include instructions for authors.  These
-- notes must not be copied.

Unassigned and Reserved Values:

-- To be completed only if unassigned and/or reserved values
-- (which may include experimental values) should be included
-- in the module.  These values are typically not included.

Description Substatements:

-- To be completed only if the default actions described in
-- Section 5.3.2 of RFC 9907 are to be overridden.
-- Specify whether instructions apply to "revision" statements,
-- "identity" statements, or both.
```

```
Reference Substatements:

-- To be completed only if the default actions described in
-- Section 5.3.2 of RFC 9907 are to be overridden.
-- Specify whether instructions apply to "revision" statements,
-- "identity" statements, or both.

Naming Considerations:

-- If a name in the IANA registry does not comply with the
-- YANG naming conventions, add details how IANA can generate
-- legal identifiers.  For example, if the name begins with
-- a number, indicate a preference to spell out the number when
-- used as an identifier.

<END TEMPLATE TEXT>
```

4.30.3.2. Template for IANA-Maintained YANG Modules with Enumerations

```
<BEGIN TEMPLATE TEXT>

This document defines the initial version of the IANA-maintained
"iana-foo" YANG module.  The most recent version of the YANG module
is available from the "YANG Parameters" registry group
[IANA-YANG-PARAMETERS].

IANA is requested to add this note to the registry:

    New values must not be directly added to the "iana-foo" YANG
    module.  They must instead be added to the "foo" registry.

When a value is added to the "foo" registry, a new "enum" statement
must be added to the "iana-foo" YANG module.  The "enum" statement,
and substatements thereof, should be defined:

 "enum":        Replicates a name from the registry.

 "value":       Contains the decimal value of the IANA-assigned
                value.

 "status":      Is included only if a registration has been
                deprecated or obsoleted.  IANA "deprecated" maps
                to YANG status "deprecated", and IANA "obsolete"
                maps to YANG status "obsolete".

 "description":  Replicates the description from the registry.

 "reference":   Replicates the reference(s) from the registry.
                References to documents should also inlcude titles.

-- Optional:

-- Include only text that needs to be customized for the module.
-- Text that does not require customization should be
-- omitted.

-- Notes tagged with "--" include instructions for authors.  These
-- notes must not be copied.

Unassigned and Reserved Values:

-- To be completed only if unassigned and/or reserved values
-- (which may include experimental values) should be included
-- in the module.  These values are typically not included.

Description Substatements:

-- To be completed only if the default actions described in
-- Section 5.3.2 of RFC 9907 are to be overridden.
-- Specify whether instructions apply to "revision" statements, "enum"
-- statements, or both.

Reference Substatements:

-- To be completed only if the default actions described in
-- Section 5.3.2 of RFC 9907 are to be overridden.
```

```
-- Specify whether instructions apply to "revision" statements, "enum"
-- statements, or both.

Naming Considerations:

-- If a name in the IANA registry does not comply with the
-- YANG naming conventions, add details how IANA can generate
-- legal identifiers.  For example, if the name begins with
-- a number, indicate a preference to spell out the number when
-- used as an identifier.

<END TEMPLATE TEXT>
```

5. IANA Considerations

5.1. YANG Modules

The following registration in the "ns" registry of the "IETF XML Registry" registry group [RFC3688] was detailed in [RFC8407]. IANA has updated this registration 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.

IANA has registered the following URI in the "ns" registry within the "IETF XML Registry" registry group [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:iana-template
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.

IANA has registered the following YANG modules in the "YANG Module Names" registry [RFC6020] [RFC9890] within the "YANG Parameters" registry group.

Name: ietf-template
Maintained by IANA? N
Namespace: urn:ietf:params:xml:ns:yang:ietf-template
Prefix: temp
Reference: RFC 9907

Name: iana-template
Maintained by IANA? N
Namespace: urn:ietf:params:xml:ns:yang:iana-template
Prefix: iana-foo
Reference: RFC 9907

5.2. Update in YANG Parameters Registry Group

For the references of the "YANG Module Names" registry under the "YANG Parameters" registry group, IANA has updated [RFC8407] to this document, as it contains the template necessary for registration in Appendix B.

5.3. IANA-Maintained YANG Modules

IANA should refer to Section 4.30.3 for information necessary to populate "revision" statements and "identity" and "enum" substatements in IANA-maintained YANG modules.

These considerations cover both the creation and maintenance of an IANA-mainatined YANG module, and they include both instructions applicable to all IANA-maintained YANG modules and instructions that can be customized by module creators.

5.3.1. Requirements for All Modules

In particular, the following instructions should apply to all modules:

- When a YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements. The "revision" statement MUST contain both "description" and "reference" substatements as described in Section 5.3.2.
- When an underlying registration is deprecated or obsoleted, a corresponding "status" substatement should be added to the "identity" or "enum" statement.
- The "reference" substatement in the "revision" statement should point specifically to the published module (i.e., IANA_FOO_URL_With_REV). When the registration is triggered by an RFC, that RFC must also be included in the "reference" substatement. It may also point to an authoritative event triggering the update to the YANG module. In all cases, the event is cited from the underlying IANA registry.
- References to documents should include titles.

In addition, when the module is published, IANA must add the following notes to:

The YANG Module Names registry:
: New values must not be directly added to the "iana-foo" YANG module. They must instead be added to the "foo" registry.

The underlying registry:
: When this registry is modified, the YANG module "iana-foo" [IANA_FOO_URL] must be updated as defined in RFC IIII.

5.3.2. Requirements Subject to Customization

Unless the creators of an IANA-maintained YANG module specify otherwise in their document's IANA Considerations section, the following instructions will apply:

- Unassigned and reserved values (including experimental values) will be omitted from the module.
- The "reference" substatement in an "identity" or "enum" statement should mirror the underlying registry. It may point to contact names as well as documents.
- In a "revision" statement, the "description" substatement captures what changed in the revised version. Typically, the "description" enumerates changes such as updates to existing entries (e.g., update a "description" or a "reference") or notes which identities were added or had their status changed (e.g., deprecated, discouraged, or obsoleted).

  When such a description is not feasible, the description varies in accordance with the trigger for the update.

  If the update is triggered by an RFC, the "description" substatement should include or consist of this text:

  > Applied updates as specified by RFC XXXX.

  If the registration policy for the registry does not require RFC publication (Section 4 of [RFC8126]), insert this text:

  > Applied updates as specified by the registration policy <Some_IANA_policy>.

6. Operational Considerations

Although the document focuses on YANG data modeling language guidance, the document does not define a protocol or a protocol extension. As such, there are no new operations or manageability requirements introduced by this document.

7. Security Considerations

This document defines guidelines for NETCONF or RESTCONF content defined with the YANG data modeling language. It does not introduce any new or increased security risks.

8. References

8.1. Normative References

[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>.

[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>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC8791] Bierman, A., Björklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <https://www.rfc-editor.org/info/rfc8791>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[RFC8819] Hopps, C., Berger, L., and D. Bogdanovic, "YANG Module Tags", RFC 8819, DOI 10.17487/RFC8819, January 2021, <https://www.rfc-editor.org/info/rfc8819>.

[RFC9890] Bierman, A., Boucadair, M., Ed., and Q. Wu, "An Update to YANG Module Names Registration", RFC 9890, DOI 10.17487/RFC9890, October 2025, <https://www.rfc-editor.org/info/rfc9890>.

[W3C.REC-xpath] Clark, J., Ed. and S. DeRose, Ed., "XML Path Language (XPath) Version 1.0", W3C Recommendation, 16 November 1999, <https://www.w3.org/TR/1999/REC-xpath-19991116>.

8.2. Informative References

[Err5693] RFC Errata, Erratum ID 5693, RFC 8407, <https://www.rfc-editor.org/errata/eid5693>.

[Err5800] RFC Errata, Erratum ID 5800, RFC 8407, <https://www.rfc-editor.org/errata/eid5800>.

[Err6899] RFC Errata, Erratum ID 6899, RFC 8407, <https://www.rfc-editor.org/errata/eid6899>.

[Err7416] RFC Errata, Erratum ID 7416, RFC 8407, <https://www.rfc-editor.org/errata/eid7416>.

[IANA-MOD-NAMES] IANA, "YANG Module Names", <https://www.iana.org/assignments/yang-parameters/>.

[IANA-TAGS] IANA, "YANG Module Tags", <https://www.iana.org/assignments/yang-module-tags/>.

[IANA-XML] IANA, "IETF XML Registry", <https://www.iana.org/assignments/xml-registry/>.

[IANA-YANG-PARAMETERS] IANA, "YANG Parameters", <https://www.iana.org/assignments/yang-parameters>.

[IANA_Tunnel_Type_URL] IANA, "iana-tunnel-type YANG Module", <https://www.iana.org/assignments/iana-tunnel-type>.

[ID-Guidelines] IETF, "Content guidelines overview", <https://authors.ietf.org/en/content-guidelines-overview>.

[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>.

[RFC2606] Eastlake 3rd, D. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, DOI 10.17487/RFC2606, June 1999, <https://www.rfc-editor.org/info/rfc2606>.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <https://www.rfc-editor.org/info/rfc2863>.

[RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/info/rfc3849>.

[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>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC5398] Huston, G., "Autonomous System (AS) Number Reservation for Documentation Use", RFC 5398, DOI 10.17487/RFC5398, December 2008, <https://www.rfc-editor.org/info/rfc5398>.

[RFC5612] Eronen, P. and D. Harrington, "Enterprise Number for Documentation Use", RFC 5612, DOI 10.17487/RFC5612, August 2009, <https://www.rfc-editor.org/info/rfc5612>.

[RFC5737] Arkko, J., Cotton, M., and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation", RFC 5737, DOI 10.17487/RFC5737, January 2010, <https://www.rfc-editor.org/info/rfc5737>.

[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>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <https://www.rfc-editor.org/info/rfc7407>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <https://www.rfc-editor.org/info/rfc8299>.

[RFC8309] Wu, Q., Liu, W., and A. Farrel, "Service Models Explained", RFC 8309, DOI 10.17487/RFC8309, January 2018, <https://www.rfc-editor.org/info/rfc8309>.

[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>.

[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>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[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>.

[RFC8466] Wen, B., Fioccola, G., Ed., Xie, C., and L. Jalil, "A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery", RFC 8466, DOI 10.17487/RFC8466, October 2018, <https://www.rfc-editor.org/info/rfc8466>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

[RFC8675] Boucadair, M., Farrer, I., and R. Asati, "A YANG Data Model for Tunnel Interface Types", RFC 8675, DOI 10.17487/RFC8675, November 2019, <https://www.rfc-editor.org/info/rfc8675>.

[RFC8892] Thaler, D. and D. Romascanu, "Guidelines and Registration Procedures for Interface Types and Tunnel Types", RFC 8892, DOI 10.17487/RFC8892, August 2020, <https://www.rfc-editor.org/info/rfc8892>.

[RFC8969] Wu, Q., Ed., Boucadair, M., Ed., Lopez, D., Xie, C., and L. Geng, "A Framework for Automating Service and Network Management with YANG", RFC 8969, DOI 10.17487/RFC8969, January 2021, <https://www.rfc-editor.org/info/rfc8969>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9108] Lhotka, L. and P. Špaček, "YANG Types for DNS Classes and Resource Record Types", RFC 9108, DOI 10.17487/RFC9108, September 2021, <https://www.rfc-editor.org/info/rfc9108>.

[RFC9129] Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "YANG Data Model for the OSPF Protocol", RFC 9129, DOI 10.17487/RFC9129, October 2022, <https://www.rfc-editor.org/info/rfc9129>.

[RFC9132] Boucadair, M., Ed., Shallow, J., and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, DOI 10.17487/RFC9132, September 2021, <https://www.rfc-editor.org/info/rfc9132>.

[RFC9182] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., Munoz, L., and A. Aguado, "A YANG Network Data Model for Layer 3 VPNs", RFC 9182, DOI 10.17487/RFC9182, February 2022, <https://www.rfc-editor.org/info/rfc9182>.

[RFC9195] Lengyel, B. and B. Claise, "A File Format for YANG Instance Data", RFC 9195, DOI 10.17487/RFC9195, February 2022, <https://www.rfc-editor.org/info/rfc9195>.

[RFC9244] Boucadair, M., Ed., Reddy.K, T., Ed., Doron, E., Chen, M., and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Telemetry", RFC 9244, DOI 10.17487/RFC9244, June 2022, <https://www.rfc-editor.org/info/rfc9244>.

[RFC9291] Boucadair, M., Ed., Gonzalez de Dios, O., Ed., Barguil, S., and L. Munoz, "A YANG Network Data Model for Layer 2 VPNs", RFC 9291, DOI 10.17487/RFC9291, September 2022, <https://www.rfc-editor.org/info/rfc9291>.

[RFC9362] Boucadair, M. and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Configuration Attributes for Robust Block Transmission", RFC 9362, DOI 10.17487/RFC9362, February 2023, <https://www.rfc-editor.org/info/rfc9362>.

[RFC9637] Huston, G. and N. Buraglio, "Expanding the IPv6 Documentation Space", RFC 9637, DOI 10.17487/RFC9637, August 2024, <https://www.rfc-editor.org/info/rfc9637>.

[RFC9645] Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", RFC 9645, DOI 10.17487/RFC9645, October 2024, <https://www.rfc-editor.org/info/rfc9645>.

[RFC9911] Schönwälder, J., Ed., "Common YANG Data Types", RFC 9911, DOI 10.17487/RFC9911, December 2025, <https://www.rfc-editor.org/info/rfc9911>.

[Style] "IANA YANG", commit 3a6cb69, December 2021, <https://github.com/llhotka/iana-yang>.

Appendix A. Module Review Checklist

This section is adapted from [RFC4181].

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:

- I-D Boilerplate: Verify that the document contains the required sections (see <https://authors.ietf.org/required-content>).

- 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>.
- 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/>
- Security Considerations section: If none of the modules in the document falls under the exceptions in Section 3.7 (e.g., use YANG data structure), verify that the section is modeled after the latest approved template from the Operations and Management (OPS) area website (see <https://wiki.ietf.org/group/ops/yang-security-guidelines>) and that the guidelines therein have been followed.
- IANA Considerations section: This section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:
  - XML Namespace Registry: Register the YANG module namespace.
  - YANG Module Registry: Register the YANG module name, prefix, namespace, and RFC number according to the rules specified in [RFC6020].
- References: Verify that the references are properly divided between normative and informative references, that 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.
- License: Verify that the document contains the Revised 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/>
- Other Issues: Check for any issues mentioned in <https://www.ietf.org/id-info/checklist.html> that are not covered elsewhere.
- Technical Content: Review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information, including formatting advice, can be found at: <https://wiki.ietf.org/group/netconf> and <https://wiki.ietf.org/group/netmod>

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. Template for IETF Modules

```
<CODE BEGINS> file "ietf-template@2023-07-26.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 your-wg-name (Expanded WG Name) Working Group";

  // update this contact statement with your info

  contact
    "WG Web:   https://datatracker.ietf.org/wg/your-wg-name
     WG List:  YOUR-WG-NAME <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 Revised BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
```

```
      All revisions of IETF and IANA published modules can be found
      at the YANG Parameters registry group
      (https://www.iana.org/assignments/yang-parameters).

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

   // RFC Ed: replace 'date-revision' with the module publication date
   // the format is (YYYY-MM-DD)

   // replace XXXX with actual RFC number and remove
   // this note

   revision date-revision {
     description
       "What changed in this revision.";
     reference
       "RFC XXXX: <Replace With Document Title>";
   }

   // Authors: Replace RFC IIII with the RFC number and title
   // of the RFC that defined the initial version of
   // the module and remove this note

   revision date-initial {
     description
       "Initial version.";
     reference
       "RFC IIII: <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>
```

Appendix C. Template for IANA-Maintained YANG Modules

```
<CODE BEGINS> file "iana-template@2023-12-08.yang"

module iana-template {
  yang-version 1.1;

  // replace this string with a unique namespace URN value

  namespace "urn:ietf:params:xml:ns:yang:iana-template";
```

```
// replace with the assigned prefix

prefix iana-foo;

organization
  "Internet Assigned Numbers Authority (IANA)";

contact
  "Internet Assigned Numbers Authority

   ICANN
   12025 Waterfront Drive, Suite 300
   Los Angeles, CA 90094

   Tel: +1 310 301 5800

   <mailto:iana@iana.org>";

description
  "This module defines a template for IANA-maintained 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 Revised BSD License set
   forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

   All revisions of IETF and IANA published modules can be found
   at the YANG Parameters registry group
   (https://www.iana.org/assignments/yang-parameters).

   The initial version of this YANG module is part of RFC IIII;
   see the RFC itself for full legal notices.

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

// If a script is used, complete with the script information

   This version of this YANG module was generated from the
   corresponding IANA registry using a <script-info>.

// RFC Ed.: replace the IANA_FOO_URL and remove this note

   The latest version of this YANG module is available at
   <IANA_FOO_URL>.";

// replace with the registry name and the URL of the IANA registry

reference
  "Registry Name (URL)";


// replace 'date-revision' with the module publication date
```

```
  // the format is (YYYY-MM-DD)

  revision date-revision {
    description
      "Indicates the list of changes per Section 4.30.3 of RFC 9907";
    reference
      "URL of the latest version of the module
       (if any) list the authoritative event (e.g., RFC) that
       triggered the update to the YANG module";
  }

  // replace 'date-initial' with the module publication date
  // the format is (YYYY-MM-DD)

  revision date-initial {
    description
      "Initial version.";
    reference
      "URL of the published initial version of the module
       RFC IIII: RFC Title";

  // RFC Ed.: Update with the RFC number and title
  // of the RFC that defined the initial version of
  // the module and remove this note
  }

  // identity statements
  // typedef statements
}
<CODE ENDS>
```

Acknowledgments

Thanks to Jürgen Schönwälder and Ladislav Lhotka for the discussion and valuable comments. Special thanks to Ladislav Lhotka for sharing more context that led to the design documented in [RFC9108].

Thanks to Italo Busi, Benoît Claise, Tom Petch, Randy Presuhn, Martin Björklund, Acee Lindem, Dale R. Worley, Kent Watsen, Jan Lindblad, Qiufang Ma, Mahesh Jethanandani, Robert Wilton, and Thomas Fossati for the comments.

Lou Berger suggested to include more details about IANA considerations.

Section 4.28 is inspired by [RFC8819].

Michal Vaško reported an inconsistency in Sections 4.6.2 and 4.6.4 of [RFC8407].

Thanks to Xufeng Liu for reviewing the document, including providing YANGDOCTORS reviews.

Italo Busi provided the examples of "case + when" construct.

Thanks to Rich Salz and Michael Richardson for the SAAG review.

Kent Watsen contributed text to the security and IANA-maintained YANG module templates.

Special thanks to Amanda Baber for the thoughtful and careful review of the document.

Thanks to Qiufang Ma for the careful shepherd review.

Thanks to Acee Lindem for triggering the discussion on data model versus module.

Thanks to Mahesh Jethanandani for the thoughtful AD review.

Thanks to Christer Holmberg for the genart review, Jean Mahoney for the check on RPC implications, Ralf Weber for the dnsdir, Giuseppe Fioccola for the opsdir review, Joseph Touch for the tsvart review, and Yoav Nir for the secdir review.

Thanks Éric Vyncke, Mike Bishop, Roman Danyliw, Orie Steele, Ketan Talaulikar, Deb Cooley, and Gorry Fairhurst for the IESG review.

The author of RFC 8407:

Andy Bierman
YumaWorks
Email: andy@yumaworks.com

Acknowledgments from RFC 8407:

> 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.

Authors' Addresses

Andy Bierman
YumaWorks
United States of America
Email: andy@yumaworks.com

Mohamed Boucadair (EDITOR)
Orange
France
Email: mohamed.boucadair@orange.com

Qin Wu
Huawei
China
Email: bill.wu@huawei.com