

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: October 1, 2017

A. Clemm
Huawei
E. Voit
J. Medved
Cisco Systems
March 30, 2017

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-06.txt

Abstract

This document introduces capabilities that allow YANG datastores to reference and incorporate information from remote datastores. This is accomplished by extending YANG with the ability to define mount points that reference data nodes in another YANG subtree, by subsequently allowing those data nodes to be accessed by client applications as if part of an alternative data hierarchy, and by providing the necessary means to manage and administer those mount points. Two flavors are defined: Alias-Mount allows to mount local subtrees, while Peer-Mount allows subtrees to reside on and be authoritatively owned by a remote server. YANG-Mount facilitates the development of applications that need to access data that transcends individual network devices while improving network-wide object consistency, or that require an aliasing capability to be able to create overlay structures for YANG data.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 1, 2017.

Copyright Notice

Copyright (c) 2017 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
1.1. Overview	3
1.2. Examples	5
2. Definitions and Acronyms	7
3. Example scenarios	7
3.1. Network controller view	8
3.2. Consistent network configuration	10
4. Operating on mounted data	11
4.1. General principles	11
4.2. Data retrieval	12
4.3. Other operations	12
4.4. Other considerations	13
5. Data model structure	14
5.1. YANG mountpoint extensions	14
5.2. YANG structure diagrams	15
5.3. Mountpoint management	15
5.4. Caching	17
5.5. Other considerations	18
5.5.1. Authorization	18

5.5.2. Datastore qualification	18
5.5.3. Mount cascades	18
5.5.4. Implementation considerations	19
5.5.5. Modeling best practices	20
6. Datastore mountpoint YANG module	20
7. Security Considerations	28
8. Acknowledgements	28
9. Normative References	28
Appendix A. Example	30
Authors' Addresses	34

1. Introduction

1.1. Overview

This document introduces a new capability that allows YANG datastores [RFC7950] to incorporate and reference information from other YANG subtrees. The capability allows a client application to retrieve and have visibility of that YANG data as part of an alternative structure. This is provided by introducing a mountpoint concept. This concept allows to declare a YANG data node in a primary datastore to serve as a "mount point" under which a subtree with YANG data can be mounted. This way, data nodes from another subtree can be inserted into an alternative data hierarchy, arranged below local data nodes. To the user, this provides visibility to data from other subtrees, rendered in a way that makes it appear largely as if it were an integral part of the datastore. This enables users to retrieve local "native" as well as mounted data in integrated fashion, using e.g. Netconf [RFC6241] or Restconf [RFC8040] data retrieval primitives. The concept is reminiscent of concepts in a Network File System that allows to mount remote folders and make them appear as if they were contained in the local file system of the user's machine.

Two variants of YANG-Mount are introduced, which build on one another:

- o Alias-Mount allows mountpoints to reference a local YANG subtree residing on the same server. It provides effectively an aliasing capability, allowing for an alternative hierarchy and path for the same YANG data.
- o Peer-Mount allows mountpoints to reference a remote YANG subtree, residing on a different server. It can be thought of as an extension to Alias-Mount, in which a remote server can be specified. Peer-Mount allows a server to effectively provide a federated datastore, including YANG data from across the network.

In each case, mounted data is authoritatively owned by the server that it is a part of. Validation of integrity constraints apply to the authoritative copy; mounting merely provides a different view of the same data. It does not impose additional constraints on that same data; however, mounted data may be referred to from other data nodes. The mountpoint concept applies in principle to operations beyond data retrieval, i.e. to configuration, RPCs, and notifications. However, support for such operations involves additional considerations, for example if support for configuration transactions and locking (which might now apply across the network) were to be provided. While it is conceivable that additional capabilities for operations on mounted information are introduced at some point in time, their specification is beyond the scope of this specification.

YANG does provide means by which modules that have been separately defined can reference and augment one another. YANG also does provide means to specify data nodes that reference other data nodes. However, all the data is assumed to be instantiated as part of the same datastore, for example a datastore provided through a NETCONF server. Existing YANG mechanisms do not account for the possibility that some information that needs to be referred not only resides in a different subtree of the same datastore, or was defined in a separate module that is also instantiated in the same datastore, but that is genuinely part of a different datastore that is provided by a different server.

The ability to mount information from local and remote datastores is new and not covered by existing YANG mechanisms. Until now, management information provided in a datastore has been intrinsically tied to the same server and to a single data hierarchy. In contrast, the capability introduced in this specification allows the server to render alternative data hierarchies, and to represent information from remote systems as if it were its own and contained in its own local data hierarchy.

The capability of allowing the mounting of information from other subtrees is accomplished by a set of YANG extensions that allow to define such mount points. For this purpose, a new YANG module is introduced. The module defines the YANG extensions, as well as a data model that can be used to manage the mountpoints and mounting process itself. Only the mounting module and its server (i.e. the "receivers" or "consumers" of the mounted information) need to be aware of the concepts introduced here. Mounting is transparent to the "providers" of the mounted information and models that are being mounted; any data nodes or subtrees within any YANG model can be mounted.

Alias-Mount and Peer-Mount build on top of each other. It is possible for a server to support Alias-Mount but not Peer-Mount. In essence, Peer-Mount requires an additional parameter that is used to refer to the target system. This parameter does not need to be supported if only Alias-Mount is provided.

Finally, it should be mentioned that Alias-Mount and Peer-Mount are not to be confused with the ability to mount a schema, aka Schema Mount. A Schema Mount allows to instantiate an existing model definition underneath a mount point, not reference a set of YANG data that has already been instantiated somewhere else. In that sense, Schema-Mount resembles more a "grouping" concept that allows to reuse an existing definition in a new context, as opposed to referencing and incorporating existing instance information into a new context.

1.2. Examples

The ability to mount data from remote datastores is useful to address various problems that several categories of applications are faced with.

One category of applications that can leverage this capability are network controller applications that need to present a consolidated view of management information in datastores across a network. Controller applications are faced with the problem that in order to expose information, that information needs to be part of their own datastore. Today, this requires support of a corresponding YANG data module. In order to expose information that concerns other network elements, that information has to be replicated into the controller's own datastore in the form of data nodes that may mirror but are clearly distinct from corresponding data nodes in the network element's datastore. In addition, in many cases, a controller needs to impose its own hierarchy on the data that is different from the one that was defined as part of the original module. An example for this concerns interface data, both operational data (e.g. various types of interface statistics) and configuration data, such as defined in [RFC7223]. This data will be contained in a top-level container ("interfaces", in this particular case) in a network element datastore. The controller may need to provide its clients a view on interface data from multiple devices under its scope of control. One way of to do so would involve organizing the data in a list with separate list elements for each device. However, this in turn would require introduction of redundant YANG modules that effectively replicate the same interface data save for differences in hierarchy.

By directly mounting information from network element datastores, the controller does not need to replicate the same information from

multiple datastores, nor does it need to re-define any network element and system-level abstractions to be able to put them in the context of network abstractions. Instead, the subtree of the remote system is attached to the local mount point. Operations that need to access data below the mount point are in effect transparently redirected to remote system, which is the authoritative owner of the data. The mounting system does not even necessarily need to be aware of the specific data in the remote subtree. Optionally, caching strategies can be employed in which the mounting system prefetches data.

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters. When each network element maintains its own datastore with the same configurable settings, a single global change requires modifying the same information in many network elements across a network. In case of inconsistent configurations, network failures can result that are difficult to troubleshoot. In many cases, what is more desirable is the ability to configure such settings in a single place, then make them available to every network element. Today, this requires in general the introduction of specialized servers and configuration options outside the scope of NETCONF, such as RADIUS [RFC2866] or DHCP [RFC2131]. In order to address this within the scope of NETCONF and YANG, the same information would have to be redundantly modeled and maintained, representing operational data (mirroring some remote server) on some network elements and configuration data on a designated master. Either way, additional complexity ensues.

Instead of replicating the same global parameters across different datastores, the solution presented in this document allows a single copy to be maintained in a subtree of single datastore that is then mounted by every network element that requires awareness of these parameters. The global parameters can be hosted in a controller or a designated network element. This considerably simplifies the management of such parameters that need to be known across elements in a network and require global consistency.

It should be noted that for these and many other applications merely having a view of the remote information is sufficient. It allows to define consolidated views of information without the need for replicating data and models that have already been defined, to audit information, and to validate consistency of configurations across a network. Only retrieval operations are required; no operations that involve configuring remote data are involved.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

DHCP: Dynamic Host Configuration Protocol.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Datastore-push: A mechanism that allows a client to subscribe to updates from a datastore, which are then automatically pushed by the server to the client.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

Mount client: The system at which the mount point resides, into which the remote subtree is mounted.

Mount point: A data node that receives the root node of the remote datastore being mounted.

Mount server: The server with which the mount client communicates and which provides the mount client with access to the mounted information. Can be used synonymously with mount target.

Mount target: A remote server whose datastore is being mounted.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

RADIUS: Remote Authentication Dial In User Service.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Example scenarios

The following example scenarios outline some of the ways in which the ability to mount YANG datastores can be applied. Other mount topologies can be conceived in addition to the ones presented here.

3.1. Network controller view

Network controllers can use the mounting capability to present a consolidated view of management information across the network. This allows network controllers to expose network-wide abstractions, such as topologies or paths, multi-device abstractions, such as VRRP [RFC3768], and network-element specific abstractions, such as information about a network element's interfaces.

While an application on top of a controller could bypass the controller to access network elements directly for their element-specific abstractions, this would come at the expense of added inconvenience for the client application. In addition, it would compromise the ability to provide layered architectures in which access to the network by controller applications is truly channeled through the controller.

Without a mounting capability, a network controller would need to at least conceptually replicate data from network elements to provide such a view, incorporating network element information into its own controller model that is separate from the network element's, indicating that the information in the controller model is to be populated from network elements. This can introduce issues such as data inconsistency and staleness. Equally important, it would lead to the need to define redundant data models: one model that is implemented by the network element itself, and another model to be implemented by the network controller. This leads to poor maintainability, as analogous information has to be redundantly defined and implemented across different data models. In general, controllers cannot simply support the same modules as their network elements for the same information because that information needs to be put into a different context. This leads to "node"-information that needs to be instantiated and indexed differently, because there are multiple instances across different data stores.

For example, "system"-level information of a network element would most naturally be placed into a top-level container at that network element's datastore. At the same time, the same information in the context of the overall network, such as maintained by a controller, might better be provided in a list. For example, the controller might maintain a list with a list element for each network element, underneath which the network element's system-level information is contained. However, the containment structure of data nodes in a module, once defined, cannot be changed. This means that in the context of a network controller, a second module that repeats the same system-level information would need to be defined, implemented, and maintained. Any augmentations that add additional system-level information to the original module will likewise need to be

redundantly defined, once for the "system" module, a second time for the "controller" module.

By allowing a network controller to directly mount information from network element datastores, the controller does not need to replicate the same information from multiple datastores. Perhaps even more importantly, the need to re-define any network element and system-level abstractions just to be able to put them in the context of network abstractions is avoided. In this solution, a network controller's datastore mounts information from many network element datastores. For example, the network controller datastore (the "primary" datastore) could implement a list in which each list element contains a mountpoint. Each mountpoint mounts a subtree from a different network element's datastore. The data from the mounted subtrees is then accessible to clients of the primary datastore using the usual data retrieval operations.

This scenario is depicted in Figure 1. In the figure, M1 is the mountpoint for the datastore in Network Element 1 and M2 is the mountpoint for the datastore in Network Element 2. MDN1 is the mounted data node in Network Element 1, and MDN2 is the mounted data node in Network Element 2.

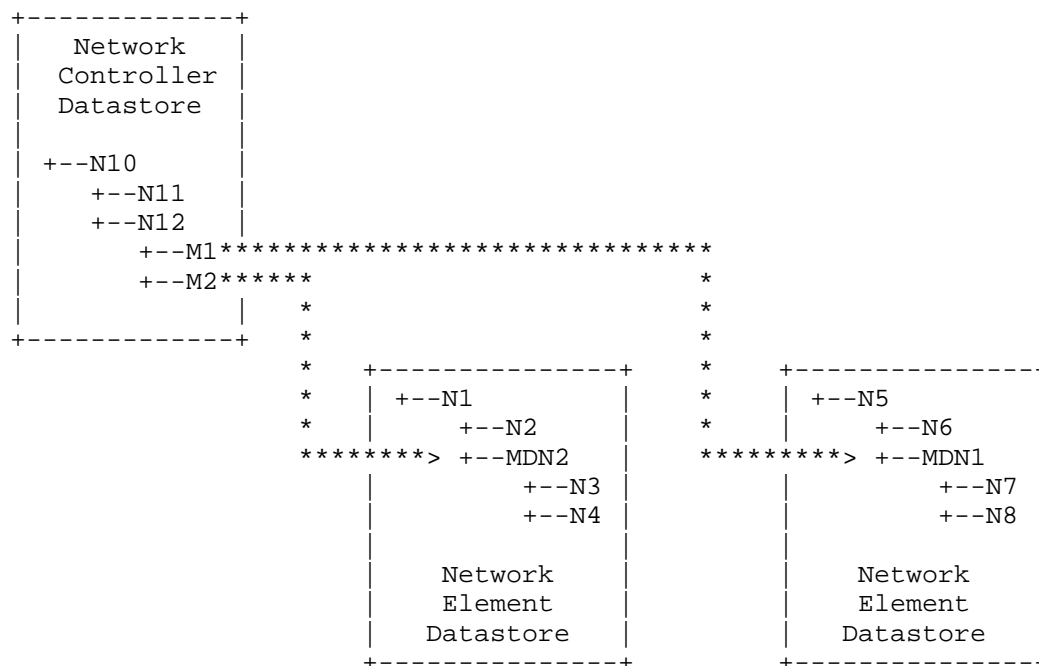


Figure 1: Network controller mount topology

3.2. Consistent network configuration

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters that need to be known across elements in a network. Today, the configuration of such parameters is generally performed on a per network element basis, which is not only redundant but, more importantly, error-prone. Inconsistent configurations lead to erroneous network behavior that can be challenging to troubleshoot.

Using the ability to mount information from remote datastores opens up a new possibility for managing such settings. Instead of replicating the same global parameters across different datastores, a single copy is maintained in a subtree of single datastore. This datastore can be hosted in a controller or a designated network element. The subtree is subsequently mounted by every network element that requires access to these parameters.

In many ways, this category of applications is an inverse of the previous category: Whereas in the network controller case data from many different datastores would be mounted into the same datastore with multiple mountpoints, in this case many elements, each with their own datastore, mount the same remote datastore, which is then mounted by many different systems.

The scenario is depicted in Figure 2. In the figure, M1 is the mountpoint for the Network Controller datastore in Network Element 1 and M2 is the mountpoint for the Network Controller datastore in Network Element 2. MDN is the mounted data node in the Network Controller datastore that contains the data nodes that represent the shared configuration settings. (Note that there is no reason why the Network Controller Datastore in this figure could not simply reside on a network element itself; the division of responsibilities is a logical one.

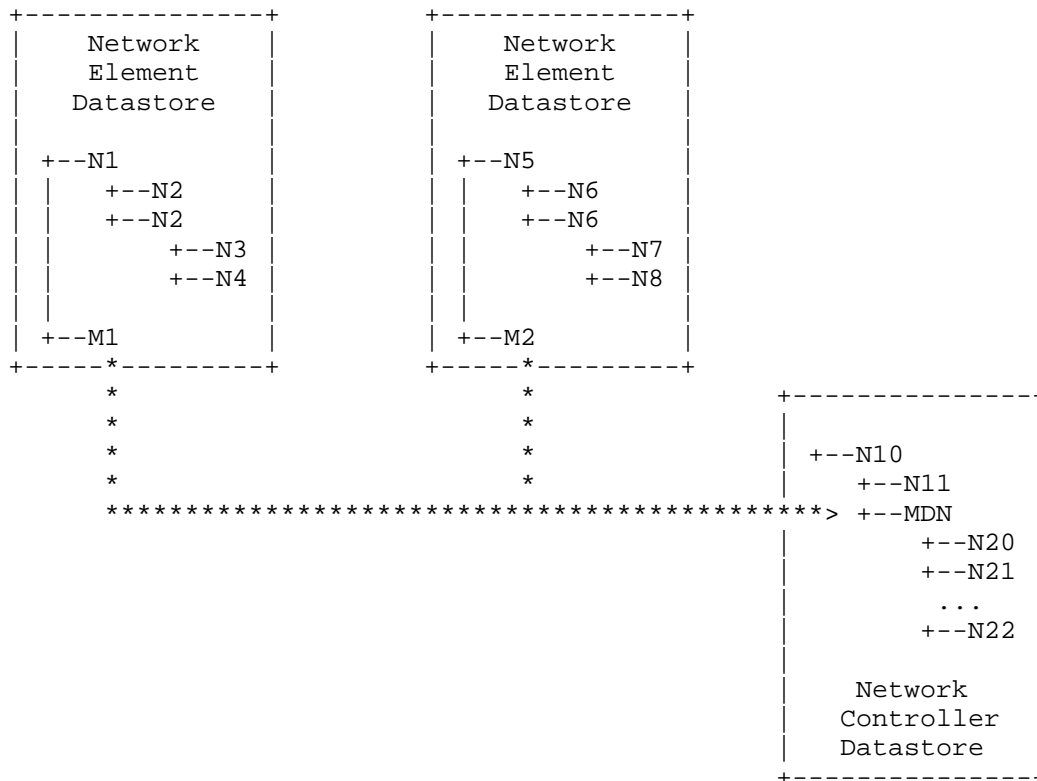


Figure 2: Distributed config settings topology

4. Operating on mounted data

This section provides a rough illustration of the operations flow involving mounted datastores.

4.1. General principles

The first thing that should be noted about these operations flows concerns the fact that a mount client essentially constitutes a special management application that interacts with a subtree to render the data of that subtree as an alternative tree hierarchy. In the case of Alias-Mount, both original and alternative tree are maintained by the same server, which in effect provides alternative paths to the same data. In the case of Peer-Mount, the mount client constitutes in effect another application, with the remote system remaining the authoritative owner of the data. While it is conceivable that the remote system (or an application that proxies for the remote system) provides certain functionality to facilitate

the specific needs of the mount client to make it more efficient, the fact that another system decides to expose a certain "view" of that data is fundamentally not the remote system's concern.

When a client application makes a request to a server that involves data that is mounted from a remote system, the server will effectively act as a proxy to the remote system on the client application's behalf. It will extract from the client application request the portion that involves the mounted subtree from the remote system. It will strip that portion of the local context, i.e. remove any local data paths and insert the data path of the mounted remote subtree, as appropriate. The server will then forward the transposed request to the remote system that is the authoritative owner of the mounted data, acting itself as a client to the remote server. Upon receiving the reply, the server will transpose the results into the local context as needed, for example map the data paths into the local data tree structure, and combine those results with the results of the remainder portion of the original request.

4.2. Data retrieval

Data retrieval operations are the only category of operations that is supported for peer-mounted information. In that case, a Netconf "get" or "get-configuration" operation might be applied on a subtree whose scope includes a mount point. When resolving the mount point, the server issues its own "get" or "get-configuration" request against the remote system's subtree that is attached to the mount point. The returned information is then inserted into the data structure that is in turn returned to the client that originally invoked the request.

4.3. Other operations

The fact that only data retrieval operations are the only category of operations that are supported for peer-mounted information does not preclude other operations to be applied to datastore subtrees that contain mountpoints and peer-mounted information. Peer-mounted information is simply transparent to those operations. When an operation is applied to a subtree which includes mountpoints, mounted information is ignored for purposes of the operation. For example, for a Netconf "edit-config" operation that includes a subtree with a mountpoint, a server will ignore the data under the mountpoint and apply the operation only to the local configuration. Mounted data is "read-only" data. The server does not even need to return an error message that the operation could not be applied to mounted data; the mountpoint is simply ignored.

In principle, it is conceivable that operations other than data-retrieval are applied to mounted data as well. For example, an operation to edit configuration information might expect edits to be applied to remote systems as part of the operation, where the edited subtree involves mounted information. However, editing of information and "writing through" to remote systems potentially involves significant complexity, particularly if transactions and locking across multiple configuration items are involved. Support for such operations will require additional capabilities, specification of which is beyond the scope of this specification.

Likewise, YANG-Mount does not extend towards RPCs that are defined as part of YANG modules whose contents is being mounted. Support for RPCs that involve mounted portions of the datastore, while conceivable, would require introduction of an additional capability, whose definition is outside the scope of this specification.

By the same token, YANG-Mount does not extend towards notifications. It is conceivable to offer such support in the future using a separate capability, definition of which is once again outside the scope of this specification.

4.4. Other considerations

Since mounting of information typically involves communication with a remote system, there is a possibility that the remote system will not respond within a certain amount of time, that connectivity is lost, or that other errors occur. Accordingly, the ability to mount datastores also involves mountpoint management, which includes the ability to configure timeouts, retries, and management of mountpoint state (including dynamic addition removal of mountpoints). Mountpoint management will be discussed in section Section 5.3.

It is expected that some implementations will introduce caching schemes. Caching can increase performance and efficiency in certain scenarios (for example, in the case of data that is frequently read but that rarely changes), but increases implementation complexity. Caching is not required for YANG-mount to work - in which case access to mounted information is "on-demand", in which the authoritative data node always gets accessed. Whether to perform caching is a local implementation decision.

When caching is introduced, it can benefit from the ability to subscribe to updates on remote data by remote servers. Some optimizations to facilitate caching support will be discussed in section Section 5.4.

5. Data model structure

5.1. YANG mountpoint extensions

At the center of the module is a set of YANG extensions that allow to define a mountpoint.

- o The first extension, "mountpoint", is used to declare a mountpoint. The extension takes the name of the mountpoint as an argument.
- o The second extension, "subtree", serves as substatement underneath a mountpoint statement. It takes an argument that defines the root node of the datastore subtree that is to be mounted, specified as string that contains a path expression. This extension is used to define mountpoints for Alias-Mount, as well as Peer-Mount.
- o The third extension, "target", also serves as a substatement underneath a mountpoint statement. It is used for Peer-Mount and takes an argument that identifies the target system. The argument is a reference to a data node that contains the information that is needed to identify and address a remote server, such as an IP address, a host name, or a URI [RFC3986].

A mountpoint **MUST** be contained underneath a container. Future revisions might allow for mountpoints to be contained underneath other data nodes, such as lists, leaf-lists, and cases. However, to keep things simple, at this point mounting is only allowed directly underneath a container.

Only a single data node can be mounted at one time. While the mount target could refer to any data node, it is recommended that as a best practice, the mount target **SHOULD** refer to a container. It is possible to maintain e.g. a list of mount points, with each mount point each of which has a mount target an element of a remote list. However, to avoid unnecessary proliferation of the number of mount points and associated management overhead, when data from lists or leaf-lists is to be mounted, a container containing the list respectively leaf-list **SHOULD** be mounted instead of individual list elements.

It is possible for a mounted datastore to contain another mountpoint, thus leading to several levels of mount indirections. However, mountpoints **MUST NOT** introduce circular dependencies. In particular, a mounted datastore **MUST NOT** contain a mountpoint which specifies the mounting datastore as a target and a subtree which contains as root node a data node that in turn contains the original mountpoint.

Whenever a mount operation is performed, this condition mountpoint.
Whenever a mount operation is performed, this condition MUST be
validated by the mount client.

5.2. YANG structure diagrams

YANG data model structure overviews have proven very useful to convey the "Big Picture". It would be useful to indicate in YANG data model structure overviews the fact that a given data node serves as a mountpoint. We propose for this purpose also a corresponding extension to the structure representation convention. Specifically, we propose to prefix the name of the mounting data node with upper-case 'M'.

```
rw network
+-- rw nodes
  +-- rw node [node-ID]
    +-- rw node-ID
    +-- M node-system-info
```

5.3. Mountpoint management

The YANG module contains facilities to manage the mountpoints themselves.

For this purpose, a list of the mountpoints is introduced. Each list element represents a single mountpoint. It includes an identification of the mount target, i.e. the remote system hosting the remote datastore and a definition of the subtree of the remote data node being mounted. It also includes monitoring information about current status (indicating whether the mount has been successful and is operational, or whether an error condition applies such as the target being unreachable or referring to an invalid subtree).

In addition to the list of mountpoints, a set of global mount policy settings allows to set parameters such as mount retries and timeouts.

Each mountpoint list element also contains a set of the same configuration knobs, allowing administrators to override global mount policies and configure mount policies on a per-mountpoint basis if needed.

There are two ways how mounting occurs: automatic (dynamically performed as part of system operation) or manually (administered by a user or client application). A separate mountpoint-origin object is used to distinguish between manually configured and automatically populated mountpoints.

Whether mounting occurs automatically or needs to be manually configured by a user or an application can depend on the mountpoint being defined, i.e. the semantics of the model.

When configured automatically, mountpoint information is automatically populated by the datastore that implements the mountpoint. The precise mechanisms for discovering mount targets and bootstrapping mount points are provided by the mount client infrastructure and outside the scope of this specification. Likewise, when a mountpoint should be deleted and when it should merely have its mount-status indicate that the target is unreachable is a system-specific implementation decision.

Manual mounting consists of two steps. In a first step, a mountpoint is manually configured by a user or client application through administrative action. Once a mountpoint has been configured, actual mounting occurs through an RPCs that is defined specifically for that purpose. To unmount, a separate RPC is invoked; mountpoint configuration information needs to be explicitly deleted. Manual mounting can also be used to override automatic mounting, for example to allow an administrator to set up or remove a mountpoint.

It should be noted that mountpoint management does not allow users to manually "extend" the model, i.e. simply add a subtree underneath some arbitrary data node into a datastore, without a supporting mountpoint defined in the model to support it. A mountpoint definition is a formal part of the model with well-defined semantics. Accordingly, mountpoint management does not allow users to dynamically "extend" the data model itself. It allows users to populate the datastore and mount structure within the confines of a model that has been defined prior.

The structure of the mountpoint management data model is depicted in the following figure, where brackets enclose list keys, "rw" means configuration, "ro" operational state data, and "?" designates optional nodes. Parentheses enclose choice and case nodes. The figure does not depict all definitions; it is intended to illustrate the overall structure.


```

module: ietf-mount
  +--rw mount-server-mgmt {mount-server-mgmt}?
    +--rw mountpoints
      +--rw mountpoint* [mountpoint-id]
        +--rw mountpoint-id      string
        +--ro mountpoint-origin? enumeration
        +--rw subtree-ref        subtree-ref
        +--rw mount-target
          +--rw (target-address-type)
            +--:(IP)
              | +--rw target-ip?      inet:ip-address
            +--:(URI)
              | +--rw uri?            inet:uri
            +--:(host-name)
              | +--rw hostname?       inet:host
            +--:(node-ID)
              | +--rw node-info-ref?  subtree-ref
            +--:(other)
              | +--rw opaque-target-ID? string
          +--ro mount-status?        mount-status
        +--rw manual-mount?          empty
        +--rw retry-timer?           uint16
        +--rw number-of-retries?     uint8
      +--rw global-mount-policies
        +--rw manual-mount?          empty
        +--rw retry-timer?           uint16
        +--rw number-of-retries?     uint8

```

5.4. Caching

Under certain circumstances, it can be useful to maintain a cache of remote information. Instead of accessing the remote system, requests are served from a copy that is locally maintained. This is particularly advantageous in cases where data is slow changing, i.e. when there are many more "read" operations than changes to the underlying data node, and in cases when a significant delay were incurred when accessing the remote system, which might be prohibitive for certain applications. Examples of such applications are applications that involve real-time control loops requiring response times that are measured in milliseconds. However, as data nodes that are mounted from an authoritative datastore represent the "golden copy", it is important that any modifications are reflected as soon as they are made.

It is a local implementation decision of mount clients whether to cache information once it has been fetched. However, in order to support more powerful caching schemes, it becomes necessary for the mount server to "push" information proactively. For this purpose, it

is useful for the mount client to subscribe for updates to the mounted information at the mount server. A corresponding mechanism that can be leveraged for this purpose is specified in draft-ietf-netconf-yang-push-05.

Note that caching large mountpoints can be expensive. Therefore limiting the amount of data unnecessarily passed when mounting near the top of a YANG subtree is important. For these reasons, an ability to specify a particular caching strategy in conjunction with mountpoints can be desirable, including the ability to exclude certain nodes and subtrees from caching. According capabilities may be introduced in a future version of this draft.

5.5. Other considerations

5.5.1. Authorization

Access to mounted information is subject to authorization rules. To the mounted system, a mounting client will in general appear like any other client. Authorization privileges for remote mounting clients need to be specified through NACM (NETCONF Access Control Model) [RFC6536].

5.5.2. Datastore qualification

It is conceivable to differentiate between different datastores on the remote server, that is, to designate the name of the actual datastore to mount, e.g. "running" or "startup". However, for the purposes of this spec, we assume that the datastore to be mounted is generally implied. Mounted information is treated as analogous to operational data; in general, this means the running or "effective" datastore is the target. That said, the information which targets to mount does constitute configuration and can hence be part of a startup or candidate datastore.

5.5.3. Mount cascades

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint. As part of a mount operation, the mount points of the mounted system need to be checked accordingly.

5.5.4. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, the following considerations apply:

Systems that wish to mount information from remote datastores need to implement a mount client. The mount client communicates with a remote system to access the remote datastore. To do so, there are several options:

- o The mount client acts as a NETCONF client to a remote system. Alternatively, another interface to the remote system can be used, such as a REST API using JSON encodings, as specified in [RFC7951]. --> Either way, to the remote system, the mount client constitutes essentially a client application like any other. The mount client in effect IS a special kind of client application.
- o The mount client communicates with a remote mount server through a separate protocol. The mount server is deployed on the same system as the remote NETCONF datastore and interacts with it through a set of local APIs.
- o The mount client communicates with a remote mount server that acts as a NETCONF client proxy to a remote system, on the client's behalf. The communication between mount client and remote mount server might involve a separate protocol, which is translated into NETCONF operations by the remote mount server.

It is the responsibility of the mount client to manage the association with the target system, e.g. validate it is still reachable by maintaining a permanent association, perform reachability checks in case of a connectionless transport, etc.

It is the responsibility of the mount client to manage the mountpoints. This means that the mount client needs to populate the mountpoint monitoring information (e.g. keep mount-status up to data and determine in the case of automatic mounting when to add and remove mountpoint configuration). In the case of automatic mounting, the mount client also interacts with the mountpoint discovery and bootstrap process.

The mount client needs to also participate in servicing datastore operations involving mounted information. An operation requested involving a mountpoint is relayed by the mounting system's infrastructure to the mount client. For example, a request to retrieve information from a datastore leads to an invocation of an internal mount client API when a mount point is reached. The mount client then relays a corresponding operation to the remote datastore.

It subsequently relays the result along with any responses back to the invoking infrastructure, which then merges the result (e.g. a retrieved subtree with the rest of the information that was retrieved) as needed. Relaying the result may involve the need to transpose error response codes in certain corner cases, e.g. when mounted information could not be reached due to loss of connectivity with the remote server, or when a configuration request failed due to validation error.

5.5.5. Modeling best practices

There is a certain amount of overhead associated with each mount point. The mount point needs to be managed and state maintained. Data subscriptions need to be maintained. Requests including mounted subtrees need to be decomposed and responses from multiple systems combined.

For those reasons, as a general best practice, models that make use of mount points SHOULD be defined in a way that minimizes the number of mountpoints required. Finely granular mounts, in which multiple mountpoints are maintained with the same remote system, each containing only very small data subtrees, SHOULD be avoided. For example, lists SHOULD only contain mountpoints when individual list elements are associated with different remote systems. To mount data from lists in remote datastores, a container node that contains all list elements SHOULD be mounted instead of mounting each list element individually. Likewise, instead of having mount points refer to nodes contained underneath choices, a mountpoint should refer to a container of the choice.

6. Datastore mountpoint YANG module

```
<CODE BEGINS>
file "ietf-mount@2017-03-30.yang"
module ietf-mount {
  namespace "urn:ietf:params:xml:ns:yang:ietf-mount";
  prefix mnt;

  import ietf-inet-types {
    prefix inet;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>
```

WG Chair: Kent Watsen
<mailto:kwatsen@juniper.net>

WG Chair: Lou Berger
<mailto:lberger@labn.net>

Editor: Alexander Clemm
<mailto:ludwig@clemm.org>

Editor: Jan Medved
<mailto:jmedved@cisco.com>

Editor: Eric Voit
<mailto:evoit@cisco.com>";

description

"This module provides a set of YANG extensions and definitions that can be used to mount information from remote datastores.";

```
revision 2017-03-30 {  
  description  
    "Initial revision.";  
  reference  
    "draft-clemm-netmod-mount-06.txt";  
}
```

```
extension mountpoint {  
  argument name;  
  description  
    "This YANG extension is used to mount data from another  
    subtree in place of the node under which this YANG extension  
    statement is used."
```

This extension takes one argument which specifies the name of the mountpoint.

This extension can occur as a substatement underneath a container statement, a list statement, or a case statement. As a best practice, it SHOULD occur as statement only underneath a container statement, but it MAY also occur underneath a list or a case statement.

The extension can take two parameters, target and subtree, each defined as their own YANG extensions.

For Alias-Mount, a mountpoint statement MUST contain a subtree statement for the mountpoint definition to be valid. For Peer-Mount, a mountpoint statement MUST contain both a target and a subtree substatement for the mountpoint

definition to be valid.

The subtree SHOULD be specified in terms of a data node of type 'mnt:subtree-ref'. The targeted data node MUST represent a container.

The target system MAY be specified in terms of a data node that uses the grouping 'mnt:mount-target'. However, it can be specified also in terms of any other data node that contains sufficient information to address the mount target, such as an IP address, a host name, or a URI.

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint.";

}

```
extension target {
  argument target-name;
  description
    "This YANG extension is used to perform a Peer-Mount.
    It is used to specify a remote target system from which to
    mount a datastore subtree. This YANG
    extension takes one argument which specifies the remote
    system. In general, this argument will contain the name of
    a data node that contains the remote system information. It
    is recommended that the reference data node uses the
    mount-target grouping that is defined further below in this
    module.
```

This YANG extension can occur only as a substatement below a mountpoint statement. It MUST NOT occur as a substatement below any other YANG statement.";

}

```
extension subtree {
  argument subtree-path;
  description
    "This YANG extension is used to specify a subtree in a
    datastore that is to be mounted. This YANG extension takes
    one argument which specifies the path to the root of the
    subtree. The root of the subtree SHOULD represent an
    instance of a YANG container. However, it MAY represent
    also another data node.
```

```
    This YANG extension can occur only as a substatement below
    a mountpoint statement. It MUST NOT occur as a substatement
    below any other YANG statement.";
}

feature mount-server-mgmt {
  description
    "Provide additional capabilities to manage remote mount
    points";
}

typedef mount-status {
  type enumeration {
    enum "ok" {
      description
        "Mounted";
    }
    enum "no-target" {
      description
        "The argument of the mountpoint does not define a
        target system";
    }
    enum "no-subtree" {
      description
        "The argument of the mountpoint does not define a
        root of a subtree";
    }
    enum "target-unreachable" {
      description
        "The specified target system is currently
        unreachable";
    }
    enum "mount-failure" {
      description
        "Any other mount failure";
    }
    enum "unmounted" {
      description
        "The specified mountpoint has been unmounted as the
        result of a management operation";
    }
  }
  description
    "This type is used to represent the status of a
    mountpoint.";
}

typedef subtree-ref {
```

```
type string;
description
  "This string specifies a path to a datanode. It corresponds
  to the path substatement of a leafref type statement. Its
  syntax needs to conform to the corresponding subset of the
  XPath abbreviated syntax. Contrary to a leafref type,
  subtree-ref allows to refer to a node in a remote datastore.
  Also, a subtree-ref refers only to a single node, not a list
  of nodes."
}

grouping mount-monitor {
  description
    "This grouping contains data nodes that indicate the
    current status of a mount point."
  leaf mount-status {
    type mount-status;
    config false;
    description
      "Indicates whether a mountpoint has been successfully
      mounted or whether some kind of fault condition is
      present."
  }
}

grouping mount-target {
  description
    "This grouping contains data nodes that can be used to
    identify a remote system from which to mount a datastore
    subtree."
  container mount-target {
    description
      "A container is used to keep mount target information
      together."
    choice target-address-type {
      mandatory true;
      description
        "Allows to identify mount target in different ways,
        i.e. using different types of addresses."
      case IP {
        leaf target-ip {
          type inet:ip-address;
          description
            "IP address identifying the mount target."
        }
      }
      case URI {
        leaf uri {
```



```

        type inet:uri;
        description
            "URI identifying the mount target";
    }
}
case host-name {
    leaf hostname {
        type inet:host;
        description
            "Host name of mount target.";
    }
}
case node-ID {
    leaf node-info-ref {
        type subtree-ref;
        description
            "Node identified by named subtree.";
    }
}
case other {
    leaf opaque-target-ID {
        type string;
        description
            "Catch-all; could be used also for mounting
            of data nodes that are local.";
    }
}
}
}
}

grouping mount-policies {
    description
        "This grouping contains data nodes that allow to configure
        policies associated with mountpoints.";
    leaf manual-mount {
        type empty;
        description
            "When present, a specified mountpoint is not
            automatically mounted when the mount data node is
            created, but needs to be mounted via specific RPC
            invocation.";
    }
    leaf retry-timer {
        type uint16;
        units "seconds";
        description
            "When specified, provides the period after which

```

```
        mounting will be automatically reattempted in case of a
        mount status of an unreachable target";
    }
    leaf number-of-retries {
        type uint8;
        description
            "When specified, provides a limit for the number of
            times for which retries will be automatically
            attempted";
    }
}

rpc mount {
    description
        "This RPC allows an application or administrative user to
        perform a mount operation.  If successful, it will result in
        the creation of a new mountpoint.";
    input {
        leaf mountpoint-id {
            type string {
                length "1..32";
            }
            description
                "Identifier for the mountpoint to be created.
                The mountpoint-id needs to be unique;
                if the mountpoint-id of an existing mountpoint is
                chosen, an error is returned.";
        }
    }
    output {
        leaf mount-status {
            type mount-status;
            description
                "Indicates if the mount operation was successful.";
        }
    }
}

rpc unmount {
    description
        "This RPC allows an application or administrative user to
        unmount information from a remote datastore.  If successful,
        the corresponding mountpoint will be removed from the
        datastore.";
    input {
        leaf mountpoint-id {
            type string {
                length "1..32";
            }
        }
    }
}
```

```
        description
          "Identifies the mountpoint to be unmounted.";
      }
  }
  output {
    leaf mount-status {
      type mount-status;
      description
        "Indicates if the unmount operation was successful.";
    }
  }
}
container mount-server-mgmt {
  if-feature mount-server-mgmt;
  description
    "Contains information associated with managing the
     mountpoints of a datastore.";
  container mountpoints {
    description
      "Keep the mountpoint information consolidated
       in one place.";
    list mountpoint {
      key "mountpoint-id";
      description
        "There can be multiple mountpoints.
         Each mountpoint is represented by its own
         list element.";
      leaf mountpoint-id {
        type string {
          length "1..32";
        }
        description
          "An identifier of the mountpoint.
           RPC operations refer to the mountpoint
           using this identifier.";
      }
      leaf mountpoint-origin {
        type enumeration {
          enum "client" {
            description
              "Mountpoint has been supplied and is
               manually administered by a client";
          }
          enum "auto" {
            description
              "Mountpoint is automatically
               administered by the server";
          }
        }
      }
    }
  }
}
```

```
    }
    config false;
    description
      "This describes how the mountpoint came
       into being.";
  }
  leaf subtree-ref {
    type subtree-ref;
    mandatory true;
    description
      "Identifies the root of the subtree in the
       target system that is to be mounted.";
  }
  uses mount-target;
  uses mount-monitor;
  uses mount-policies;
}
}
container global-mount-policies {
  description
    "Provides mount policies applicable for all mountpoints,
     unless overridden for a specific mountpoint.";
  uses mount-policies;
}
}
```

<CODE ENDS>

7. Security Considerations

TBD

8. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Tony Tkacik, Ambika Tripathy, Robert Varga, Prabhakara Yellai, Shashi Kumar Bansal, Lukas Sedlak, and Benoit Claise.

9. Normative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <<http://www.rfc-editor.org/info/rfc2131>>.

- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <<http://www.rfc-editor.org/info/rfc2866>>.
- [RFC3768] Hinden, R., Ed., "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, DOI 10.17487/RFC3768, April 2004, <<http://www.rfc-editor.org/info/rfc3768>>.
- [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, <<http://www.rfc-editor.org/info/rfc3986>>.
- [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, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<http://www.rfc-editor.org/info/rfc7923>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

Appendix A. Example

In the following example, we are assuming the use case of a network controller that wants to provide a controller network view to its client applications. This view needs to include network abstractions that are maintained by the controller itself, as well as certain information about network devices where the network abstractions tie in with element-specific information. For this purpose, the network controller leverages the mount capability specified in this document and presents a fictitious Controller Network YANG Module that is depicted in the outlined structure below. The example illustrates how mounted information is leveraged by the mounting datastore to provide an additional level of information that ties together network and device abstractions, which could not be provided otherwise without introducing a (redundant) model to replicate those device abstractions

```
rw controller-network
+-- rw topologies
|   +-- rw topology [topo-id]
|       +-- rw topo-id          node-id
|       +-- rw nodes
|           +-- rw node [node-id]
|               +-- rw node-id          node-id
|               +-- rw supporting-ne    network-element-ref
|               +-- rw termination-points
|                   +-- rw term-point [tp-id]
|                       +-- tp-id        tp-id
|                       +-- ifref         mountedIfRef
|       +-- rw links
|           +-- rw link [link-id]
|               +-- rw link-id          link-id
|               +-- rw source            tp-ref
|               +-- rw dest              tp-ref
+-- rw network-elements
    +-- rw network-element [element-id]
        +-- rw element-id              element-id
        +-- rw element-address
        |   +-- ...
        +-- M interfaces
```

The controller network model consists of the following key components:

- o A container with a list of topologies. A topology is a graph representation of a network at a particular layer, for example, an IS-IS topology, an overlay topology, or an Openflow topology. Specific topology types can be defined in their own separate YANG

modules that augment the controller network model. Those augmentations are outside the scope of this example

- o An inventory of network elements, along with certain information that is mounted from each element. The information that is mounted in this case concerns interface configuration information. For this purpose, each list element that represents a network element contains a corresponding mountpoint. The mountpoint uses as its target the network element address information provided in the same list element
- o Each topology in turn contains a container with a list of nodes. A node is a network abstraction of a network device in the topology. A node is hosted on a network element, as indicated by a network-element leafref. This way, the "logical" and "physical" aspects of a node in the network are cleanly separated.
- o A node also contains a list of termination points that terminate links. A termination point is implemented on an interface. Therefore, it contains a leafref that references the corresponding interface configuration which is part of the mounted information of a network element. Again, the distinction between termination points and interfaces provides a clean separation between logical concepts at the network topology level and device-specific concepts that are instantiated at the level of a network element. Because the interface information is mounted from a different datastore and therefore occurs at a different level of the containment hierarchy than it would if it were not mounted, it is not possible to use the interface-ref type that is defined in YANG data model for interface management [] to allow the termination point refer to its supporting interface. For this reason, a new type definition "mountedIfRef" is introduced that allows to refer to interface information that is mounted and hence has a different path.
- o Finally, a topology also contains a container with a list of links. A link is a network abstraction that connects nodes via node termination points. In the example, directional point-to-point links are depicted in which one node termination point serves as source, another as destination.

The following is a YANG snippet of the module definition which makes use of the mountpoint definition.

```

<CODE BEGINS>
module controller-network {
    namespace "urn:cisco:params:xml:ns:yang:controller-network";
    // example only, replace with IANA namespace when assigned
    prefix cn;
    import mount {
        prefix mnt;
    }
    import interfaces {
        prefix if;
    }
    ...
    typedef mountedIfRef {
        type leafref {
            path "/cn:controller-network/cn:network-elements/"
              + "cn:network-element/cn:interfaces/if:interface/if:name";
            // cn:interfaces corresponds to the mountpoint
        }
    }
    ...
    list termination-point {
        key "tp-id";
        ...
        leaf ifref {
            type mountedIfRef;
        }
        ...
        list network-element {
            key "element-id";
            leaf element-id {
                type element-ID;
            }
            container element-address {
                ... // choice definition that allows to specify
                // host name,
                // IP addresses, URIs, etc
            }
            mnt:mountpoint "interfaces" {
                mnt:target "./element-address";
                mnt:subtree "/if:interfaces";
            }
            ...
        }
    }
    ...
<CODE ENDS>

```

Finally, the following contains an XML snippet of instantiated YANG information. We assume three datastores: NE1 and NE2 each have a

datastore (the mount targets) that contains interface configuration data, which is mounted into NC's datastore (the mount client).

Interface information from NE1 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/1</name>
    <name>ethernetCsmacd</type>
    <location>1/1</location>
  </interface>
</interfaces>
```

Interface information from NE2 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/2</name>
    <name>ethernetCsmacd</type>
    <location>1/2</location>
  </interface>
</interfaces>
```

NC datastore with mounted interface information from NE1 and NE2:

```
<controller-network>
...
<network-elements>
  <network-element>
    <element-id>NE1</element-id>
    <element-address> .... </element-address>
    <interfaces>
      <if:interface>
        <if:name>fastethernet-1/0</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/0</if:location>
      </if:interface>
      <if:interface>
        <if:name>fastethernet-1/1</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/1</if:location>
      </if:interface>
    </interfaces>
  </network-element>
  <network-element>
    <element-id>NE2</element-id>
    <element-address> .... </element-address>
    <interfaces>
      <if:interface>
        <if:name>fastethernet-1/0</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/0</if:location>
      </if:interface>
      <if:interface>
        <if:name>fastethernet-1/2</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/2</if:location>
      </if:interface>
    </interfaces>
  </network-element>
</network-elements>
...
</controller-network>
```

Authors' Addresses

Alexander Clemm
Huawei

E-Mail: ludwig@clemm.org

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

Jan Medved
Cisco Systems

EMail: jmedved@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: November 22, 2019

A. Clemm
Futurewei
E. Voit
Cisco Systems
May 21, 2019

Subscription to YANG Datastores
draft-ietf-netconf-yang-push-25

Abstract

This document describes a mechanism that allows subscriber applications to request a continuous and customized stream of updates from a YANG datastore. Providing such visibility into updates enables new capabilities based on the remote mirroring and monitoring of configuration and operational state.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 22, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Definitions and Acronyms	3
3. Solution Overview	5
3.1. Subscription Model	5
3.2. Negotiation of Subscription Policies	6
3.3. On-Change Considerations	7
3.4. Reliability Considerations	8
3.5. Data Encodings	9
3.6. Defining the Selection with a Datastore	10
3.7. Streaming Updates	11
3.8. Subscription Management	13
3.9. Receiver Authorization	15
3.10. On-Change Notifiable Datastore Nodes	16
3.11. Other Considerations	17
4. A YANG Data Model for Management of Datastore Push Subscriptions	18
4.1. Overview	18
4.2. Subscription Configuration	24
4.3. YANG Notifications	25
4.4. YANG RPCs	26
5. YANG Module	31
6. IANA Considerations	48
7. Security Considerations	48
8. Acknowledgments	50
9. Contributors	50
10. References	50
10.1. Normative References	50
10.2. Informative References	51
Appendix A. Appendix A: Subscription Errors	52
A.1. RPC Failures	52
A.2. Notifications of Failure	54

Appendix B. Changes Between Revisions	54
Authors' Addresses	58

1. Introduction

Traditional approaches to provide visibility into managed entities from a remote system have been built on polling. With polling, data is periodically requested and retrieved by a client from a server to stay up-to-date. However, there are issues associated with polling-based management:

- o Polling incurs significant latency. This latency prohibits many types of application.
- o Polling cycles may be missed and requests may be delayed or get lost, often when the network is under stress and the need for the data is the greatest.
- o Polling requests may undergo slight fluctuations, resulting in intervals of different lengths. The resulting data is difficult to calibrate and compare.
- o For applications that monitor for changes, many remote polling cycles place unwanted and ultimately wasteful load on the network, devices, and applications, particularly when changes occur only infrequently.

A more effective alternative to polling is for an application to receive automatic and continuous updates from a targeted subset of a datastore. Accordingly, there is a need for a service that allows applications to subscribe to updates from a datastore and that enables the server (also referred to as publisher) to push and in effect stream those updates. The requirements for such a service have been documented in [RFC7923].

This document defines a corresponding solution that is built on top of "Custom Subscription to Event Streams" [I-D.draft-ietf-netconf-subscribed-notifications]. Supplementing that work are YANG data model augmentations, extended RPCs, and new datastore specific update notifications. Transport options for [I-D.draft-ietf-netconf-subscribed-notifications] will work seamlessly with this solution.

2. Definitions and Acronyms

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.

This document uses the terminology defined in [RFC7950], [RFC8341], [RFC8342], and [I-D.draft-ietf-netconf-subscribed-notifications]. In addition, the following terms are introduced:

- o Datastore node: A node in the instantiated YANG data tree associated with a datastore. In this document, datastore nodes are often also simply referred to as "objects"
- o Datastore node update: A data item containing the current value of a datastore node at the time the datastore node update was created, as well as the path to the datastore node.
- o Datastore subscription: A subscription to a stream of datastore node updates.
- o Datastore subtree: A datastore node and all its descendant datastore nodes
- o On-change subscription: A datastore subscription with updates that are triggered when changes in subscribed datastore nodes are detected.
- o Periodic subscription: A datastore subscription with updates that are triggered periodically according to some time interval.
- o Selection filter: Evaluation and/or selection criteria, which may be applied against a targeted set of objects.
- o Update record: A representation of one or more datastore node updates. In addition, an update record may contain which type of update led to the datastore node update (e.g., whether the datastore node was added, changed, deleted). Also included in the update record may be other metadata, such as a subscription id of the subscription as part of which the update record was generated. In this document, update records are often also simply referred to as "updates".
- o Update trigger: A mechanism that determines when an update record needs to be generated.
- o YANG-Push: The subscription and push mechanism for datastore updates that is specified in this document.

3. Solution Overview

This document specifies a solution that provides a subscription service for updates from a datastore. This solution supports dynamic as well as configured subscriptions to updates of datastore nodes. Subscriptions specify when notification messages (also referred to as "push updates") should be sent and what data to include in update records. Datastore node updates are subsequently pushed from the publisher to the receiver per the terms of the subscription.

3.1. Subscription Model

YANG-push subscriptions are defined using a YANG data model. This model enhances the subscription model defined in [I-D.draft-ietf-netconf-subscribed-notifications] with capabilities that allow subscribers to subscribe to datastore node updates, specifically to specify the update triggers defining when to generate update records as well as what to include in an update record. Key enhancements include:

- o Specification of selection filters which identify targeted YANG datastore nodes and/or datastore subtrees for which updates are to be pushed.
- o Specification of update policies contain conditions which trigger the generation and pushing of new update records. There are two types of subscriptions, distinguished by how updates are triggered: periodic and on-change.
 - * For periodic subscriptions, the update trigger is specified by two parameters that define when updates are to be pushed. These parameters are the period interval with which to report updates, and an "anchor time", i.e. a reference point in time that can be used to calculate at which points in time periodic updates need to be assembled and sent.
 - * For on-change subscriptions, an update trigger occurs whenever a change in the subscribed information is detected. Included are additional parameters that include:
 - + Dampening period: In an on-change subscription, detected object changes should be sent as quickly as possible. However it may be undesirable to send a rapid series of object changes. Such behavior has the potential to exhaust resources in the publisher or receiver. In order to protect against that, a dampening period MAY be used to specify the interval which has to pass before successive update records for the same subscription are generated for a receiver. The

dampening period collectively applies to the set of all datastore nodes selected by a single subscription. This means that when there is a change to one or more subscribed objects, an update record containing those objects is created immediately (when no dampening period is in effect) or at the end of a dampening period (when a dampening period is in fact in effect). If multiple changes to a single object occur during a dampening period, only the value that is in effect at the time when the update record is created is included. The dampening period goes into effect every time an update record completes assembly.

- + Change type: This parameter can be used to reduce the types of datastore changes for which updates are sent (e.g., you might only send an update when an object is created or deleted, but not when an object value changes).
- + Sync on start: defines whether or not a complete push-update of all subscribed data will be sent at the beginning of a subscription. Such early synchronization establishes the frame of reference for subsequent updates.
- o An encoding (using anydata) for the contents of periodic and on-change push updates.

3.2. Negotiation of Subscription Policies

A dynamic subscription request SHOULD be declined if a publisher's assessment is that it may be unable to provide update records meeting the terms of an "establish-subscription" or "modify-subscription" RPC request. In this case, a subscriber may quickly follow up with a new RPC request using different parameters.

Random guessing of different parameters by a subscriber is to be discouraged. Therefore, in order to minimize the number of subscription iterations between subscriber and publisher, a dynamic subscription supports a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is in the form of supplemental information which should be inserted within error responses to a failed RPC request. This returned error response information, when considered, should increase the likelihood of success for subsequent RPC requests. Such hints include suggested periodic time intervals, acceptable dampening periods, and size estimates for the number or objects which would be returned from a proposed selection filter. However, there are no guarantees that subsequent requests which consider these hints will be accepted.

3.3. On-Change Considerations

On-change subscriptions allow receivers to receive updates whenever changes to targeted objects occur. As such, on-change subscriptions are particularly effective for data that changes infrequently, yet for which applications need to be quickly notified whenever a change does occur with minimal delay.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

Whether or not to accept or reject on-change subscription requests when the scope of the subscription contains objects for which on-change is not supported is up to the publisher implementation. A publisher MAY accept an on-change subscription even when the scope of the subscription contains objects for which on-change is not supported. In that case, updates are sent only for those objects within the scope that do support on-change updates, whereas other objects are excluded from update records, even if their values change. In order for a subscriber to determine whether objects support on-change subscriptions, objects are marked accordingly on a publisher. Accordingly, when subscribing, it is the responsibility of the subscriber to ensure it is aware of which objects support on-change and which do not. For more on how objects are so marked, see Section 3.10.

Alternatively, a publisher MAY decide to simply reject an on-change subscription in case the scope of the subscription contains objects for which on-change is not supported. In case of a configured subscription, the publisher MAY suspend the subscription.

To avoid flooding receivers with repeated updates for subscriptions containing fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update record for a given object is generated, no other updates for this particular subscription will be created until the end of the dampening period. Values sent at the end of the dampening period are the values that are current at the end of the dampening period of all changed objects. Changed objects include those which were deleted or newly created during that dampening period. If an object has returned to its original value (or even has been created and then deleted) during the dampening-period, that value (and not the interim change) will still be sent. This will indicate churn is occurring on that object.

On-change subscriptions can be refined to let users subscribe only to certain types of changes. For example, a subscriber might only want

object creations and deletions, but not modifications of object values.

Putting it all together, following is the conceptual process for creating an update record as part of an on-change subscription:

1. Just before a change, or at the start of a dampening period, evaluate any filtering and any access control rules to ensure receiver is authorized to view all subscribed datastore nodes (filtering out any nodes for which this is not the case). The result is a set "A" of datastore nodes and subtrees.
2. Just after a change, or at the end of a dampening period, evaluate any filtering and any (possibly new) access control rules. The result is a set "B" of datastore nodes and subtrees.
3. Construct an update record, which takes the form of YANG patch record [RFC8072] for going from A to B.
4. If there were any changes made between A and B which canceled each other out, insert into the YANG patch record the last change made, even if the new value is no different from the original value (since changes that were made in the interim were canceled out). In case the changes involve creating a new datastore node, then deleting it, the YANG patch record will indicate deletion of the datastore node. Similarly, in case the changes involve deleting a new datastore node, then recreating it, the YANG patch record will indicate creation of the datastore node.
5. If the resulting patch record is non-empty, send it to the receiver.

Note: In cases where a subscriber wants to have separate dampening periods for different objects, the subscriber has the option to create multiple subscriptions with different selection filters.

3.4. Reliability Considerations

A subscription to updates from a datastore is intended to obviate the need for polling. However, in order to do so, it is critical that subscribers can rely on the subscription and have confidence that they will indeed receive the subscribed updates without having to worry about updates being silently dropped. In other words, a subscription constitutes a promise on the side of the publisher to provide the receivers with updates per the terms of the subscription.

Now, there are many reasons why a publisher may at some point no longer be able to fulfill the terms of the subscription, even if the

subscription had been entered into with good faith. For example, the volume of datastore nodes may be larger than anticipated, the interval may prove too short to send full updates in rapid succession, or an internal problem may prevent objects from being collected. For this reason, the solution that is defined in this document mandates that a publisher notifies receivers immediately and reliably whenever it encounters a situation in which it is unable to keep the terms of the subscription, and provides the publisher with the option to suspend the subscription in such a case. This includes indicating the fact that an update is incomplete as part of a push-update or push-change-update notification, as well as emitting a subscription-suspended notification as applicable. This is described further in Section 3.11.1.

A publisher SHOULD reject a request for a subscription if it is unlikely that the publisher will be able to fulfill the terms of that subscription request. In such cases, it is preferable to have a subscriber request a less resource intensive subscription than to deal with frequently degraded behavior.

The solution builds on [I-D.draft-ietf-netconf-subscribed-notifications]. As defined there, any loss of underlying transport connection will be detected and result in subscription termination (in case of dynamic subscriptions) or suspension (in case of configured subscriptions), ensuring that situations will not occur in which the loss of update notifications would go unnoticed.

3.5. Data Encodings

3.5.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update record corresponds to data that could have been read using a retrieval operation.

3.5.2. On-Change Subscriptions

In an on-change subscription, update records need to indicate not only values of changed datastore nodes but also the types of changes that occurred since the last update. Therefore, encoding rules for data in on-change updates will generally follow YANG-patch operation as specified in [RFC8072]. The YANG-patch will describe what needs to be applied to the earlier state reported by the preceding update, to result in the now-current state. Note that contrary to [RFC8072], objects encapsulated are not restricted to only configuration objects.

A publisher indicates the type of change to a datastore node using the different YANG patch operations: the "create" operation is used for newly created objects (except entries in a user-ordered list), the "delete" operation is used for deleted objects (including in user-ordered lists), the "replace" operation is used when only the object value changes, the "insert" operation is used when a new entry is inserted in a list, and the "move" operation is used when an existing entry in a user-ordered list is moved.

However, a patch must be able to do more than just describe the delta from the previous state to the current state. As per Section 3.3, it must also be able to identify whether transient changes have occurred on an object during a dampening period. To support this, it is valid to encode a YANG patch operation so that its application would result in no change between the previous and current state. This indicates that some churn has occurred on the object. An example of this would be a patch that indicates a "create" operation for a datastore node where the receiver believes one already exists, or a "replace" operation which replaces a previous value with the same value. Note that this means that the "create" and "delete" errors described in [RFC8072] section 2.5 are not errors, and are valid operations with YANG-Push.

3.6. Defining the Selection with a Datastore

A subscription must specify both the selection filters and the datastore against which these selection filters will be applied. This information is used to choose and subsequently push data from the publisher's datastore to the receivers.

Only a single selection filter can be applied to a subscription at a time. An RPC request proposing a new selection filter replaces any existing filter. The following selection filter types are included in the YANG-push data model, and may be applied against a datastore:

- o subtree: A subtree selection filter identifies one or more datastore subtrees. When specified, update records will only come from the datastore nodes of selected datastore subtree(s). The syntax and semantics correspond to that specified for [RFC6241] section 6.
- o xpath: An "xpath" selection filter is an XPath expression that returns a node set. (XPath is a query language for selecting nodes in an XML document.) When specified, updates will only come from the selected datastore nodes.

These filters are intended to be used as selectors that define which objects are within the scope of a subscription. A publisher **MUST** support at least one type of selection filter.

XPath itself provides powerful filtering constructs and care must be used in filter definition. Consider an XPath filter which only passes a datastore node when an interface is up. It is up to the receiver to understand implications of the presence or absence of objects in each update.

When the set of selection filtering criteria is applied for a periodic subscription, then they are applied whenever a periodic update record is constructed, and only datastore nodes that pass the filter and to which a receiver has access are provided to that receiver. If the same filtering criteria is applied to an on-change subscription, only the subset of those datastore nodes supporting on-change is provided. A datastore node which doesn't support on-change is never sent as part of an on-change subscription's "push-update" or "push-change-update" (see Section 3.7).

3.7. Streaming Updates

Contrary to traditional data retrieval requests, datastore subscription enables an unbounded series of update records to be streamed over time. Two generic YANG notifications for update records have been defined for this: "push-update" and "push-change-update".

A "push-update" notification defines a complete, filtered update of the datastore per the terms of a subscription. This type of YANG notification is used for continuous updates of periodic subscriptions. A "push-update" notification can also be used for the on-change subscriptions in two cases. First, it **MUST** be used as the initial "push-update" if there is a need to synchronize the receiver at the start of a new subscription. It also **MAY** be sent if the publisher later chooses to resync an on-change subscription. The "push-update" update record contains an instantiated datastore subtree with all of the subscribed contents. The content of the update record is equivalent to the contents that would be obtained had the same data been explicitly retrieved using a datastore retrieval operation using the same transport with the same filters applied.

A "push-change-update" notification is the most common type of update for on-change subscriptions. The update record in this case contains the set of changes that datastore nodes have undergone since the last notification message. In other words, this indicates which datastore nodes have been created, deleted, or have had changes to their

values. In cases where multiple changes have occurred over the course of a dampening period and the object has not been deleted, the object's most current value is reported. (In other words, for each object, only one change is reported, not its entire history. Doing so would defeat the purpose of the dampening period.)

"Push-update" and "push-change-update" are encoded and placed within notification messages, and ultimately queued for egress over the specified transport.

The following is an example of a notification message for a subscription tracking the operational status of a single Ethernet interface (per [RFC8343]). This notification message is encoded XML over NETCONF as per [I-D.draft-ietf-netconf-netconf-event-notifications].

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2017-10-25T08:00:11.22Z</eventTime>
  <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>1011</id>
    <datastore-contents>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>eth0</name>
          <oper-status>up</oper-status>
        </interface>
      </interfaces>
    </datastore-contents>
  </push-update>
</notification>
```

Figure 1: Push example

The following is an example of an on-change notification message for the same subscription.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2017-10-25T08:22:33.44Z</eventTime>
  <push-change-update
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>89</id>
    <datastore-changes>
      <yang-patch>
        <patch-id>0</patch-id>
        <edit>
          <edit-id>edit1</edit-id>
          <operation>replace</operation>
          <target>/ietf-interfaces:interfaces</target>
          <value>
            <interfaces
              xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
                <interface>
                  <name>eth0</name>
                  <oper-status>down</oper-status>
                </interface>
              </interfaces>
            </value>
          </edit>
        </yang-patch>
      </datastore-changes>
    </push-change-update>
  </notification>
```

Figure 2: Push example for on change

Of note in the above example is the 'patch-id' with a value of '0'. Per [RFC8072], the 'patch-id' is an arbitrary string. With YANG Push, the publisher SHOULD put into the 'patch-id' a counter starting at '0' which increments with every 'push-change-update' generated for a subscription. If used as a counter, this counter MUST be reset to '0' anytime a resynchronization occurs (i.e., with the sending of a 'push-update'). Also if used as a counter, the counter MUST be reset to '0' after passing a maximum value of '4294967295' (i.e. maximum value that can be represented using uint32 data type). Such a mechanism allows easy identification of lost or out-of-sequence update records.

3.8. Subscription Management

The RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] have been enhanced to support datastore subscription negotiation. Also, new error codes have been added that are able to indicate why a datastore subscription attempt has failed, along with new YANG-data that MAY be

used to include details on input parameters that might result in a successful subsequent RPC invocation.

The establishment or modification of a datastore subscription can be rejected for multiple reasons. This includes a too large subtree request, or the inability of the publisher to push update records as frequently as requested. In such cases, no subscription is established. Instead, the subscription-result with the failure reason is returned as part of the RPC response. As part of this response, a set of alternative subscription parameters MAY be returned that would likely have resulted in acceptance of the subscription request. The subscriber may consider these as part of future subscription attempts.

In the case of a rejected request for an establishment of a datastore subscription, if there are hints, the hints SHOULD be transported within a YANG-data "establish-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "establish-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "establish-subscription-datastore-error-info". All tree diagrams used in this document follow the notation defined in [RFC8340]

```

YANG-data establish-subscription-datastore-error-info
  +--ro establish-subscription-datastore-error-info
    +--ro reason?                identityref
    +--ro period-hint?           centiseconds
    +--ro filter-failure-hint?   string
    +--ro object-count-estimate? uint32
    +--ro object-count-limit?   uint32
    +--ro kilobytes-estimate?    uint32
    +--ro kilobytes-limit?       uint32

```

Figure 3: Tree diagram for establish-subscription-datastore-error-info

Similarly, in the case of a rejected request for modification of a datastore subscription, if there are hints, the hints SHOULD be transported within a YANG-data "modify-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "modify-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "modify-subscription-datastore-error-info".

```

YANG-data modify-subscription-datastore-error-info
  +--ro modify-subscription-datastore-error-info
    +--ro reason?                identityref
    +--ro period-hint?           centiseconds
    +--ro filter-failure-hint?   string
    +--ro object-count-estimate? uint32
    +--ro object-count-limit?   uint32
    +--ro kilobytes-estimate?   uint32
    +--ro kilobytes-limit?      uint32

```

Figure 4: Tree diagram for modify-subscription-datastore-error-info

3.9. Receiver Authorization

A receiver of subscription data MUST only be sent updates for which it has proper authorization. A publisher MUST ensure that no non-authorized data is included in push updates. To do so, it needs to apply all corresponding checks applicable at the time of a specific pushed update and if necessary silently remove any non-authorized data from datastore subtrees. This enables YANG data pushed based on subscriptions to be authorized equivalently to a regular data retrieval (get) operation.

Each "push-update" and "push-change-update" MUST have access control applied, as is depicted in the following diagram. This includes validating that read access is permitted for any new objects selected since the last notification message was sent to a particular receiver. To accomplish this, implementations SHOULD support the conceptual authorization model of [RFC8341], specifically section 3.2.4.

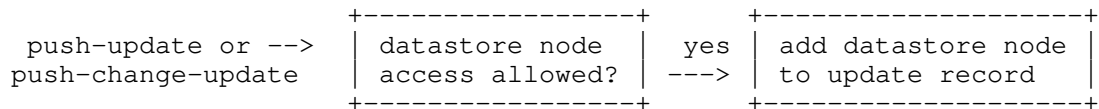


Figure 5: Updated [RFC8341] access control for push updates

A publisher MUST allow for the possibility that a subscription's selection filter references non-existent data or data that a receiver is not allowed to access. Such support permits a receiver the ability to monitor the entire lifecycle of some datastore tree without needing to explicitly enumerate every individual datastore node. If, after access control has been applied, there are no objects remaining in an update record, then (in case of a periodic subscription) only a single empty "push-update" notification MUST be sent. Empty "push-change-update" messages (in case of an on-change subscription) MUST NOT be sent. This is required to ensure that clients cannot

surreptitiously monitor objects that they do not have access to via carefully crafted selection filters. By the same token, changes to objects that are filtered MUST NOT affect any dampening intervals.

A publisher MAY choose to reject an establish-subscription request which selects non-existent data or data that a receiver is not allowed to access. As reason, the error identity "unchanging-selection" SHOULD be returned. In addition, a publisher MAY choose to terminate a dynamic subscription or suspend a configured receiver when the authorization privileges of a receiver change, or the access controls for subscribed objects change. In that case, the publisher SHOULD include the error identity "unchanging-selection" as reason when sending the "subscription-terminated" respectively "subscription-suspended" notification. Such a capability enables the publisher to avoid having to support continuous and total filtering of a subscription's content for every update record. It also reduces the possibility of leakage of access-controlled objects.

If read access into previously accessible nodes has been lost due to a receiver permissions change, this SHOULD be reported as a patch "delete" operation for on-change subscriptions. If not capable of handling such receiver permission changes with such a "delete", publisher implementations MUST force dynamic subscription re-establishment or configured subscription re-initialization so that appropriate filtering is installed.

3.10. On-Change Notifiable Datastore Nodes

In some cases, a publisher supporting on-change notifications may not be able to push on-change updates for some object types. Reasons for this might be that the value of the datastore node changes frequently (e.g., [RFC8343]'s in-octets counter), that small object changes are frequent and meaningless (e.g., a temperature gauge changing 0.1 degrees), or that the implementation is not capable of on-change notification for a particular object.

In those cases, it will be important for client applications to have a way to identify for which objects on-change notifications are supported and for which ones they are not supported. Otherwise client applications will have no way of knowing whether they can indeed rely on their on-change subscription to provide them with the change updates that they are interested in. In other words, if implementations do not provide a solution and do not support comprehensive on-change notifiability, clients of those implementations will have no way of knowing what their on-change subscription actually covers.

Implementations are therefore strongly advised to provide a solution to this problem. One solution might involve making discoverable to clients which objects are on-change notifiable, specified using another YANG data model. Such a solution is specified in [I-D.draft-ietf-netconf-notification-capabilities]. Until this solution is standardized, implementations SHOULD provide their own solution.

3.11. Other Considerations

3.11.1. Robustness and reliability

Particularly in the case of on-change updates, it is important that these updates do not get lost. In case the loss of an update is unavoidable, it is critical that the receiver is notified accordingly.

Update records for a single subscription MUST NOT be resequenced prior to transport.

It is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update record the full set of objects desired per the terms of a subscription. In this case, the publisher MUST act as follows.

- o The publisher MUST set the "incomplete-update" flag on any update record which is known to be missing information.
- o The publisher MAY choose to suspend the subscription as per [I-D.draft-ietf-netconf-subscribed-notifications]. If the publisher does not create an update record at all, it MUST suspend the subscription.
- o When resuming an on-change subscription, the publisher SHOULD generate a complete patch from the previous update record. If this is not possible and the "sync-on-start" option is true for the subscription, then the full datastore contents MAY be sent via a "push-update" instead (effectively replacing the previous contents). If neither of these are possible, then an "incomplete-update" flag MUST be included on the next "push-change-update".

Note: It is perfectly acceptable to have a series of "push-change-update" notifications (and even "push update" notifications) serially queued at the transport layer awaiting transmission. It is not required for the publisher to merge pending update records sent at the same time.

On the receiver side, what action to take when a record with an incomplete-update flag is received depends on the application. It could simply choose to wait and do nothing. It could choose to resynch, actively retrieving all subscribed information. It could also choose to tear down the subscription and start a new one, perhaps with a lesser scope that contains less objects.

3.11.2. Publisher capacity

It is far preferable to decline a subscription request than to accept such a request when it cannot be met.

Whether or not a subscription can be supported will be determined by a combination of several factors such as the subscription update trigger (on-change or periodic), the period in which to report changes (one second periods will consume more resources than one hour periods), the amount of data in the datastore subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.

4. A YANG Data Model for Management of Datastore Push Subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figures. The tree diagram that is used follows the notation defined in [RFC8340]. New schema objects defined here (i.e., beyond those from [I-D.draft-ietf-netconf-subscribed-notifications]) are identified with "yp". For the reader's convenience, in order to compact the tree representation, some nodes that are defined in ietf-subscribed-notifications and that are not essential to the understanding of the data model defined here have been removed. This is indicated by "... " in the diagram where applicable.

Because the tree diagram is quite large, its depiction is broken up into several figures. The first figure depicts the augmentations that are introduced in module ietf-yang-push to subscription configuration specified in module ietf-subscribed-notifications.

```

module: ietf-subscribed-notifications
...
+--rw filters
|
|   ...
+--rw yp:selection-filter* [filter-id]
|   +--rw yp:filter-id                string
|   +--rw (yp:filter-spec)?
|       +--:(yp:datastore-subtree-filter)
|           +--rw yp:datastore-subtree-filter?    <anydata>
|               {sn:subtree}?
|       +--:(yp:datastore-xpath-filter)
|           +--rw yp:datastore-xpath-filter?      yang:xpath1.0
|               {sn:xpath}?
+--rw subscriptions
|   +--rw subscription* [id]
|       |
|       |   ...
+--rw (target)
|   +--:(stream)
|       |
|       |   ...
+--:(yp:datastore)
|   +--rw yp:datastore                identityref
|   +--rw (yp:selection-filter)?
|       +--:(yp:by-reference)
|           +--rw yp:selection-filter-ref
|               selection-filter-ref
|       +--:(yp:within-subscription)
|           +--rw (yp:filter-spec)?
|               +--:(yp:datastore-subtree-filter)
|                   +--rw yp:datastore-subtree-filter?
|                       <anydata> {sn:subtree}?
|               +--:(yp:datastore-xpath-filter)
|                   +--rw yp:datastore-xpath-filter?
|                       yang:xpath1.0 {sn:xpath}?
|       |
|       |   ...
+--rw (yp:update-trigger)
|   +--:(yp:periodic)
|       |
|       |   +--rw yp:periodic!
|       |       +--rw yp:period            centiseconds
|       |       +--rw yp:anchor-time?      yang:date-and-time
|       +--:(yp:on-change) {on-change}?
|           +--rw yp:on-change!
|               +--rw yp:dampening-period?  centiseconds
|               +--rw yp:sync-on-start?     boolean
|               +--rw yp:excluded-change*   change-type

```

Figure 6: Model structure: subscription configuration

The next figure depicts the augmentations of module `ietf-yang-push` made to RPCs specified in module `ietf-subscribed-notifications`. Specifically, these augmentations concern the `establish-subscription` and `modify-subscription` RPCs, which are augmented with parameters that are needed to specify datastore push subscriptions.

```

rpcs:
  +---x establish-subscription
  |   +---w input
  |   |   ...
  |   |   +---w (target)
  |   |   |   +---:(stream)
  |   |   |   |   ...
  |   |   |   +---:(yp:datastore)
  |   |   |   |   +---w yp:datastore
  |   |   |   |   |   identityref
  |   |   |   |   +---w (yp:selection-filter)?
  |   |   |   |   |   +---:(yp:by-reference)
  |   |   |   |   |   |   +---w yp:selection-filter-ref
  |   |   |   |   |   |   |   selection-filter-ref
  |   |   |   |   +---:(yp:within-subscription)
  |   |   |   |   |   +---w (yp:filter-spec)?
  |   |   |   |   |   |   +---:(yp:datastore-subtree-filter)
  |   |   |   |   |   |   |   +---w yp:datastore-subtree-filter?
  |   |   |   |   |   |   |   |   <anydata> {sn:subtree}?
  |   |   |   |   |   +---:(yp:datastore-xpath-filter)
  |   |   |   |   |   |   +---w yp:datastore-xpath-filter?
  |   |   |   |   |   |   |   yang:xpath1.0 {sn:xpath}?
  |   |   |   |   ...
  |   |   +---w (yp:update-trigger)
  |   |   |   +---:(yp:periodic)
  |   |   |   |   +---w yp:periodic!
  |   |   |   |   |   +---w yp:period
  |   |   |   |   |   |   centiseconds
  |   |   |   |   |   +---w yp:anchor-time?
  |   |   |   |   |   |   yang:date-and-time
  |   |   |   +---:(yp:on-change) {on-change}?
  |   |   |   |   +---w yp:on-change!
  |   |   |   |   |   +---w yp:dampening-period?
  |   |   |   |   |   |   centiseconds
  |   |   |   |   |   +---w yp:sync-on-start?
  |   |   |   |   |   |   boolean
  |   |   |   |   +---w yp:excluded-change*
  |   |   |   |   |   change-type
  |   +---ro output
  |   |   +---ro id
  |   |   |   subscription-id
  |   |   +---ro replay-start-time-revision?
  |   |   |   yang:date-and-time
  |   |   |   {replay}?
  +---x modify-subscription
  |   +---w input
  |   |   ...
  |   |   +---w (target)
  |   |   |   ...

```

```

|      +---:(yp:datastore)
|      |      +---w yp:datastore                                identityref
|      |      +---w (yp:selection-filter)?
|      |      |      +---:(yp:by-reference)
|      |      |      |      +---w yp:selection-filter-ref
|      |      |      |      |      selection-filter-ref
|      |      |      +---:(yp:within-subscription)
|      |      |      |      +---w (yp:filter-spec)?
|      |      |      |      |      +---:(yp:datastore-subtree-filter)
|      |      |      |      |      |      +---w yp:datastore-subtree-filter?
|      |      |      |      |      |      |      <anydata> {sn:subtree}?
|      |      |      |      |      +---:(yp:datastore-xpath-filter)
|      |      |      |      |      |      +---w yp:datastore-xpath-filter?
|      |      |      |      |      |      |      yang:xpath1.0 {sn:xpath}?
|      |      |      |      +---w ...
|      |      +---w (yp:update-trigger)
|      |      |      +---:(yp:periodic)
|      |      |      |      +---w yp:periodic!
|      |      |      |      |      +---w yp:period                                centiseconds
|      |      |      |      |      +---w yp:anchor-time?                    yang:date-and-time
|      |      |      +---:(yp:on-change) {on-change}?
|      |      |      |      +---w yp:on-change!
|      |      |      |      +---w yp:dampening-period?    centiseconds
|      +---x delete-subscription
|      |      ...
|      +---x kill-subscription
|      |      ...

```

YANG-data (for placement into rpc error responses)

...

Figure 7: Model structure: RPCs

The next figure depicts augmentations of module `ietf-yang-push` to the notifications that are specified in module `ietf-subscribed-notifications`. The augmentations allow the inclusion of subscription configuration parameters that are specific to datastore push subscriptions as part of subscription-started and subscription-modified notifications.

```

notifications:
|      +---n replay-completed {replay}?
|      |      ...
|      +---n subscription-completed
|      |      ...
|      +---n subscription-started {configured}?
|      |      |      ...
|      |      +---ro (target)

```



```

...
+---:(yp:datastore)
+--ro yp:datastore                                identityref
+--ro (yp:selection-filter)?
+---:(yp:by-reference)
|   +--ro yp:selection-filter-ref
|       selection-filter-ref
+---:(yp:within-subscription)
+--ro (yp:filter-spec)?
+---:(yp:datastore-subtree-filter)
|   +--ro yp:datastore-subtree-filter?
|       <anydata> {sn:subtree}?
+---:(yp:datastore-xpath-filter)
+--ro yp:datastore-xpath-filter?
|   yang:xpath1.0 {sn:xpath}?
...
+--ro (yp:update-trigger)
+---:(yp:periodic)
|   +--ro yp:periodic!
|       +--ro yp:period            centiseconds
|       +--ro yp:anchor-time?    yang:date-and-time
+---:(yp:on-change) {on-change}?
+--ro yp:on-change!
|   +--ro yp:dampening-period?    centiseconds
|   +--ro yp:sync-on-start?       boolean
|   +--ro yp:excluded-change*     change-type
+---n subscription-resumed
...
+---n subscription-modified
...
+--ro (target)
|   ...
+---:(yp:datastore)
+--ro yp:datastore                                identityref
+--ro (yp:selection-filter)?
+---:(yp:by-reference)
|   +--ro yp:selection-filter-ref
|       selection-filter-ref
+---:(yp:within-subscription)
+--ro (yp:filter-spec)?
+---:(yp:datastore-subtree-filter)
|   +--ro yp:datastore-subtree-filter?
|       <anydata> {sn:subtree}?
+---:(yp:datastore-xpath-filter)
+--ro yp:datastore-xpath-filter?
|   yang:xpath1.0 {sn:xpath}?
...
+--ro (yp:update-trigger)?

```

```

|      +---:(yp:periodic)
|      |      +---ro yp:periodic!
|      |      |      +---ro yp:period          centiseconds
|      |      |      +---ro yp:anchor-time?    yang:date-and-time
|      +---:(yp:on-change) {on-change}?
|      |      +---ro yp:on-change!
|      |      |      +---ro yp:dampening-period?    centiseconds
|      |      |      +---ro yp:sync-on-start?      boolean
|      |      |      +---ro yp:excluded-change*    change-type
+---n subscription-terminated
|      ...
+---n subscription-suspended
|      ...

```

Figure 8: Model structure: Notifications

The final figure in this section depicts the parts of module `ietf-yang-push` that are not simply augmentations to another module, but that are newly introduced.

module: `ietf-yang-push`

rpcs:

```

+---x resync-subscription {on-change}?
|      +---w input
|      |      +---w id          sn:subscription-id

```

YANG-data: (for placement into rpc error responses)

```

+--- resync-subscription-error
|      +---ro reason?          identityref
|      +---ro period-hint?     centiseconds
|      +---ro filter-failure-hint? string
|      +---ro object-count-estimate? uint32
|      +---ro object-count-limit?   uint32
|      +---ro kilobytes-estimate?   uint32
|      +---ro kilobytes-limit?      uint32
+--- establish-subscription-error-datastore
|      +---ro reason?          identityref
|      +---ro period-hint?     centiseconds
|      +---ro filter-failure-hint? string
|      +---ro object-count-estimate? uint32
|      +---ro object-count-limit?   uint32
|      +---ro kilobytes-estimate?   uint32
|      +---ro kilobytes-limit?      uint32
+--- modify-subscription-error-datastore
|      +---ro reason?          identityref

```

```

    +--ro period-hint?           centiseconds
    +--ro filter-failure-hint?   string
    +--ro object-count-estimate? uint32
    +--ro object-count-limit?    uint32
    +--ro kilobytes-estimate?    uint32
    +--ro kilobytes-limit?       uint32

notifications:
  +---n push-update
  |   +--ro id?                  sn:subscription-id
  |   +--ro datastore-contents? <anydata>
  |   +--ro incomplete-update?   empty
  +---n push-change-update {on-change}?
  |   +--ro id?                  sn:subscription-id
  |   +--ro datastore-changes
  |   |   +--ro yang-patch
  |   |   |   +--ro patch-id      string
  |   |   |   +--ro comment?     string
  |   |   |   +--ro edit* [edit-id]
  |   |   |       +--ro edit-id    string
  |   |   |       +--ro operation  enumeration
  |   |   |       +--ro target     target-resource-offset
  |   |   |       +--ro point?    target-resource-offset
  |   |   |       +--ro where?    enumeration
  |   |   |       +--ro value?    <anydata>
  |   +--ro incomplete-update?   empty

```

Figure 9: Model structure (non-augmentation portions)

Selected components of the model are summarized below.

4.2. Subscription Configuration

Both configured and dynamic subscriptions are represented within the list "subscription". New parameters extending the basic subscription data model in [I-D.draft-ietf-netconf-subscribed-notifications] include:

- o The targeted datastore from which the selection is being made. The potential datastores include those from [RFC8341]. A platform may also choose to support a custom datastore.
- o A selection filter identifying YANG nodes of interest within a datastore. Filter contents are specified via a reference to an existing filter, or via an in-line definition for only that subscription. Referenced filters allows an implementation to avoid evaluating filter acceptability during a dynamic

subscription request. The case statement differentiates the options.

- o For periodic subscriptions, triggered updates will occur at the boundaries of a specified time interval. These boundaries can be calculated from the periodic parameters:
 - * a "period" which defines the duration between push updates.
 - * an "anchor-time"; update intervals fall on the points in time that are a multiple of a "period" from an "anchor-time". If "anchor-time" is not provided, then the "anchor-time" MUST be set with the creation time of the initial update record.
- o For on-change subscriptions, assuming any dampening period has completed, triggering occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that are guided by their own set of parameters:
 - * a "dampening-period" specifies the interval that must pass before a successive update for the subscription is sent. If no dampening period is in effect, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed for this subscription.
 - * an "excluded-change" parameter which allows restriction of the types of changes for which updates should be sent (e.g., only add to an update record on object creation).
 - * a "sync-on-start" specifies whether a complete update with all the subscribed data is to be sent at the beginning of a subscription.

4.3. YANG Notifications

4.3.1. State Change Notifications

Subscription state notifications and mechanism are reused from [I-D.draft-ietf-netconf-subscribed-notifications]. Notifications "subscription-started" and "subscription-modified" have been augmented to include the datastore specific objects.

4.3.2. Notifications for Subscribed Content

Along with the subscribed content, there are other objects which might be part of a "push-update" or "push-change-update" notification.

An "id" (that identifies the subscription) MUST be transported along with the subscribed contents. This allows a receiver to differentiate which subscription resulted in a particular update record.

A "time-of-update" which represents the time an update record snapshot was generated. A receiver MAY assume that at this point in time a publisher's objects had the values that were pushed.

An "incomplete-update" leaf. This leaf indicates that not all changes which have occurred since the last update are actually included with this update. In other words, the publisher has failed to fulfill its full subscription obligations. (For example a datastore was unable to provide the full set of datastore nodes to a publisher process.) To facilitate re-synchronization of on-change subscriptions, a publisher MAY subsequently send a "push-update" containing a full selection snapshot of subscribed data.

4.4. YANG RPCs

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.1. Establish-subscription RPC

The subscriber sends an establish-subscription RPC with the parameters in section 3.1. An example might look like:

```

<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
      xmlns:ex="http://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
    </establish-subscription>
  </netconf:rpc>

```

Figure 10: Establish-subscription RPC

A positive response includes the "id" of the accepted subscription. In that case a publisher may respond:

```

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    52
  </id>
</rpc-reply>

```

Figure 11: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, no capacity to serve the subscription at the publisher, or the inability of the publisher to select datastore content at the requested cadence.

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error hints which help a subscriber understand subscription parameters might have been accepted for the request. These hints would be included within the YANG-data structure "establish-subscription-error-datastore". However even with these hints, there are no guarantee that subsequent requests will in fact be accepted.

The specific parameters to be returned as part of the RPC error response depend on the specific transport that is used to manage the subscription. For NETCONF, those parameters are defined in [I-D.draft-ietf-netconf-netconf-event-notifications]. For example, for the following NETCONF request:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns=
      "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
      xmlns:ex="http://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:on-change>
      <yp:dampening-period>100</yp:dampening-period>
    </yp:on-change>
    </establish-subscription>
  </rpc>
```

Figure 12: Establish-subscription request example 2

a publisher that cannot serve on-change updates but that can serve periodic updates might return the following NETCONF response:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-path>/yp:periodic/yp:period</error-path>
    <error-info>
      <yp:establish-subscription-error-datastore>
        <yp:reason>yp:on-change-unsupported</yp:reason>
      </yp:establish-subscription-error-datastore>
    </error-info>
  </rpc-error>
</rpc-reply>
```

Figure 13: Establish-subscription error response example 2

4.4.2. Modify-subscription RPC

The subscriber MAY invoke the "modify-subscription" RPC for a subscription it previously established. The subscriber will include newly desired values in the "modify-subscription" RPC. Parameters not included MUST remain unmodified. Below is an example where a subscriber attempts to modify the period and datastore XPath filter of a subscription using NETCONF.

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns=
      "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>1011</id>
    <yp:datastore
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
      xmlns:ex="http://example.com/sample-data/1.0">
      /ex:bar
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>250</yp:period>
    </yp:periodic>
    </modify-subscription>
  </rpc>
```

Figure 14: Modify subscription request

The publisher MUST respond to the subscription modification request. If the request is rejected, the existing subscription is left unchanged, and the publisher MUST send an RPC error response. This response might have hints encapsulated within the YANG-data structure "modify-subscription-error-datastore". A subscription MAY be modified multiple times.

The specific parameters to be returned as part of the RPC error response depend on the specific transport that is used to manage the subscription. For NETCONF, those parameters are specified in [I-D.draft-ietf-netconf-netconf-event-notifications].

A configured subscription cannot be modified using "modify-subscription" RPC. Instead, the configuration needs to be edited as needed.

4.4.3. Delete-subscription RPC

To stop receiving updates from a subscription and effectively delete a subscription that had previously been established using an "establish-subscription" RPC, a subscriber can send a "delete-subscription" RPC, which takes as only input the subscription's "id". This RPC is unmodified from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.4. Resync-subscription RPC

This RPC is supported only for on-change subscriptions previously established using an "establish-subscription" RPC. For example:

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <resync-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push"
    <id>1011</id>
  </resync-subscription>
</netconf:rpc>
```

Figure 15: Resync subscription

On receipt, a publisher must either accept the request and quickly follow with a "push-update", or send an appropriate error within an rpc error response. Within an error response, the publisher MAY include supplemental information about the reasons within the YANG-data structure "resync-subscription-error".

4.4.5. YANG Module Synchronization

To make subscription requests, the subscriber needs to know the YANG datastore schemas used by the publisher, which are available via the YANG Library module, `ietf-yang-library.yang` from [RFC8525]. The receiver is expected to know the YANG library information before starting a subscription.

The set of modules, revisions, features, and deviations can change at run-time (if supported by the publisher implementation). For this purpose, the YANG library provides a simple "yang-library-change" notification that informs the subscriber that the library has changed. In this case, a subscription may need to be updated to take the updates into account. The receiver may also need to be informed of module changes in order to process updates regarding datastore nodes from changed modules correctly.

5. YANG Module

This YANG module imports typedefs from [RFC6991], identities from [RFC8342], the YANG-data extension from [RFC8040], and the yang-patch grouping from [RFC8072]. In addition, it imports and augments many definitions from [I-D.draft-ietf-netconf-subscribed-notifications].

```
<CODE BEGINS> file "ietf-yang-push@2019-05-21.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "draft-ietf-netconf-subscribed-notifications:
      Customized Subscriptions to a Publisher's Event Streams
      NOTE TO RFC Editor: Please replace above reference to
      draft-ietf-netconf-subscribed-notifications with RFC number
      when published (i.e. RFC xxxx).";
  }
  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA)";
  }
  import ietf-restconf {
    prefix rc;
    reference
      "RFC 8040: RESTCONF Protocol";
  }
  import ietf-yang-patch {
    prefix ypatch;
    reference
      "RFC 8072: YANG Patch Media Type";
  }

  organization
    "IETF NETCONF Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/netconf/>"

```

```
WG List:  <mailto:netconf@ietf.org>

Editor:   Alexander Clemm
          <mailto:ludwig@clemm.org>
Editor:   Eric Voit
          <mailto:evoit@cisco.com>
Editor:   Alberto Gonzalez Prieto
          <mailto:agonzalezpri@vmware.com>
Editor:   Ambika Prasad Tripathy
          <mailto:ambtripa@cisco.com>
Editor:   Einar Nilsen-Nygaard
          <mailto:einarnn@cisco.com>
Editor:   Andy Bierman
          <mailto:andy@yumaworks.com>
Editor:   Balazs Lengyel
          <mailto:balazs.lengyel@ericsson.com>";
description
  "This module contains YANG specifications for YANG push.

  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 (RFC 2119) (RFC 8174) when, and only when,
  they appear in all capitals, as shown here.

  Copyright (c) 2019 IETF Trust and the persons identified as
  authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject to
  the license terms contained in, the Simplified BSD License set
  forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

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

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

revision 2019-05-21 {
  description
    "Initial revision.
    NOTE TO RFC EDITOR:
    (1) Please replace the above revision date to
    the date of RFC publication when published.
    (2) Please replace the date in the file name
```

```
        (ietf-yang-push@2019-05-21.yang) to the date of RFC
        publication.
        (3) Please replace the following reference to
        draft-ietf-netconf-yang-push-25 with RFC number when
        published (i.e. RFC xxxx).";
    reference
        "draft-ietf-netconf-yang-push-25";
}

/*
 * FEATURES
 */

feature on-change {
    description
        "This feature indicates that on-change triggered subscriptions
        are supported.";
}

/*
 * IDENTITIES
 */

/* Error type identities for datastore subscription */

identity resync-subscription-error {
    description
        "Problem found while attempting to fulfill an
        'resync-subscription' RPC request.";
}

identity cant-exclude {
    base sn:establish-subscription-error;
    description
        "Unable to remove the set of 'excluded-changes'. This means
        the publisher is unable to restrict 'push-change-update's to
        just the change types requested for this subscription.";
}

identity datastore-not-subscribable {
    base sn:establish-subscription-error;
    base sn:subscription-terminated-reason;
    description
        "This is not a subscribable datastore.";
}

identity no-such-subscription-resync {
    base resync-subscription-error;
```

```
description
  "Referenced subscription doesn't exist. This may be as a result
  of a non-existent subscription ID, an ID which belongs to
  another subscriber, or an ID for configured subscription.";
}

identity on-change-unsupported {
  base sn:establish-subscription-error;
  description
    "On-change is not supported for any objects which are
    selectable by this filter.";
}

identity on-change-sync-unsupported {
  base sn:establish-subscription-error;
  description
    "Neither sync on start nor resynchronization are supported for
    this subscription. This error will be used for two
    reasons. First if an 'establish-subscription' RPC includes
    'sync-on-start', yet the publisher can't support sending a
    'push-update' for this subscription for reasons other than
    'on-change-unsupported' or 'sync-too-big'. And second, if the
    'resync-subscription' RPC is invoked either for an existing
    periodic subscription, or for an on-change subscription which
    can't support resynchronization.";
}

identity period-unsupported {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base sn:subscription-suspended-reason;
  description
    "Requested time period or dampening-period is too short. This
    can be for both periodic and on-change subscriptions (with or
    without dampening.) Hints suggesting alternative periods may
    be returned as supplemental information.";
}

identity update-too-big {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base sn:subscription-suspended-reason;
  description
    "Periodic or on-change push update datatrees exceed a maximum
    size limit. Hints on estimated size of what was too big may
    be returned as supplemental information.";
}
```

```
identity sync-too-big {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base resync-subscription-error;
  base sn:subscription-suspended-reason;
  description
    "Sync-on-start or resynchronization datatree exceeds a maximum
    size limit. Hints on estimated size of what was too big may
    be returned as supplemental information.";
}

identity unchanging-selection {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base sn:subscription-terminated-reason;
  description
    "Selection filter is unlikely to ever select datatree nodes.
    This means that based on the subscriber's current access
    rights, the publisher recognizes that the selection filter is
    unlikely to ever select datatree nodes which change. Examples
    for this might be that node or subtree doesn't exist, read
    access is not permitted for a receiver, or static objects that
    only change at reboot have been chosen.";
}

/*
 * TYPE DEFINITIONS
 */

typedef change-type {
  type enumeration {
    enum create {
      description
        "A change that refers to the creation of a new datastore
        node.";
    }
    enum delete {
      description
        "A change that refers to the deletion of a datastore
        node.";
    }
    enum insert {
      description
        "A change that refers to the insertion of a new
        user-ordered datastore node.";
    }
    enum move {
      description
```

```
        "A change that refers to a reordering of the target
        datastore node.";
    }
    enum replace {
        description
            "A change that refers to a replacement of the target
            datastore node's value.";
    }
}
description
    "Specifies different types of datastore changes.

    This type is based on the edit operations defined for YANG
    Patch, with the difference that it is valid for a receiver to
    process an update record which performs a create operation on
    a datastore node the receiver believes exists, or to process a
    delete on a datastore node the receiver believes is missing.";
reference
    "RFC 8072: YANG Patch Media Type, section 2.5";
}

typedef selection-filter-ref {
    type leafref {
        path "/sn:filters/yp:selection-filter/yp:filter-id";
    }
    description
        "This type is used to reference a selection filter.";
}

typedef centiseconds {
    type uint32;
    description
        "A period of time, measured in units of 0.01 seconds.";
}

/*
 * GROUP DEFINITIONS
 */

grouping datastore-criteria {
    description
        "A grouping to define criteria for which selected objects
        from a targeted datastore should be included in push
        updates.";
    leaf datastore {
        type identityref {
            base ds:datastore;
        }
    }
}
```

```
    }
    mandatory true;
    description
      "Datastore from which to retrieve data.";
  }
  uses selection-filter-objects;
}
```

grouping selection-filter-types {
 description
 "This grouping defines the types of selectors for objects
 from a datastore.";
 choice filter-spec {
 description
 "The content filter specification for this request.";
 anydata datastore-subtree-filter {
 if-feature "sn:subtree";
 description
 "This parameter identifies the portions of the
 target datastore to retrieve.";
 reference
 "RFC 6241: Network Configuration Protocol, Section 6.";
 }
 leaf datastore-xpath-filter {
 if-feature "sn:xpath";
 type yang:xpath1.0;
 description
 "This parameter contains an XPath expression identifying
 the portions of the target datastore to retrieve."

If the expression returns a node-set, all nodes in the node-set are selected by the filter. Otherwise, if the expression does not return a node-set, the filter doesn't select any nodes.

The expression is evaluated in the following XPath context:

- o The set of namespace declarations is the set of prefix and namespace pairs for all YANG modules implemented by the server, where the prefix is the YANG module name and the namespace is as defined by the 'namespace' statement in the YANG module.

If the leaf is encoded in XML, all namespace declarations in scope on the 'stream-xpath-filter' leaf element are added to the set of namespace declarations. If a prefix found in the XML is

already present in the set of namespace declarations, the namespace in the XML is used.

- o The set of variable bindings is empty.
- o The function library is the core function library, and the XPath functions defined in section 10 in RFC 7950.
- o The context node is the root node of the target datastore.";

```

    }
  }
}

grouping selection-filter-objects {
  description
    "This grouping defines a selector for objects from a
    datastore.";
  choice selection-filter {
    description
      "The source of the selection filter applied to the
      subscription. This will come either referenced from a global
      list, or be provided within the subscription itself.";
    case by-reference {
      description
        "Incorporate a filter that has been configured
        separately.";
      leaf selection-filter-ref {
        type selection-filter-ref;
        mandatory true;
        description
          "References an existing selection filter which is to be
          applied to the subscription.";
      }
    }
    case within-subscription {
      description
        "Local definition allows a filter to have the same
        lifecycle as the subscription.";
      uses selection-filter-types;
    }
  }
}

```

```

grouping update-policy-modifiable {
  description
    "This grouping describes the datastore specific subscription
    conditions that can be changed during the lifetime of the

```

```
    subscription.";
choice update-trigger {
  description
    "Defines necessary conditions for sending an event record to
    the subscriber.";
  case periodic {
    container periodic {
      presence "indicates a periodic subscription";
      description
        "The publisher is requested to notify periodically the
        current values of the datastore as defined by the
        selection filter.";
      leaf period {
        type centiseconds;
        mandatory true;
        description
          "Duration of time which should occur between periodic
          push updates, in one hundredths of a second.";
      }
      leaf anchor-time {
        type yang:date-and-time;
        description
          "Designates a timestamp before or after which a series
          of periodic push updates are determined. The next
          update will take place at a whole multiple interval
          from the anchor time. For example, for an anchor time
          is set for the top of a particular minute and a period
          interval of a minute, updates will be sent at the top
          of every minute this subscription is active.";
      }
    }
  }
}
case on-change {
  if-feature "on-change";
  container on-change {
    presence "indicates an on-change subscription";
    description
      "The publisher is requested to notify changes in values
      in the datastore subset as defined by a selection
      filter.";
    leaf dampening-period {
      type centiseconds;
      default "0";
      description
        "Specifies the minimum interval between the assembly of
        successive update records for a single receiver of a
        subscription. Whenever subscribed objects change, and
        a dampening period interval (which may be zero) has
```

```

        elapsed since the previous update record creation for
        a receiver, then any subscribed objects and properties
        which have changed since the previous update record
        will have their current values marshalled and placed
        into a new update record.";
    }
}
}
}

grouping update-policy {
  description
    "This grouping describes the datastore-specific subscription
    conditions of a subscription.";
  uses update-policy-modifiable {
    augment "update-trigger/on-change/on-change" {
      description
        "Includes objects not modifiable once subscription is
        established.";
      leaf sync-on-start {
        type boolean;
        default "true";
        description
          "When this object is set to false, it restricts an
          on-change subscription from sending push-update
          notifications. When false, pushing a full selection per
          the terms of the selection filter MUST NOT be done for
          this subscription. Only updates about changes,
          i.e. only push-change-update notifications are sent.
          When true (default behavior), in order to facilitate a
          receiver's synchronization, a full update is sent when
          the subscription starts using a push-update
          notification. After that, push-change-update
          notifications are exclusively sent unless the publisher
          chooses to resync the subscription via a new push-update
          notification.";
      }
      leaf-list excluded-change {
        type change-type;
        description
          "Use to restrict which changes trigger an update. For
          example, if modify is excluded, only creation and
          deletion of objects is reported.";
      }
    }
  }
}
}

```

```
grouping hints {
  description
    "Parameters associated with some error for a subscription
    made upon a datastore.";
  leaf period-hint {
    type centiseconds;
    description
      "Returned when the requested time period is too short. This
      hint can assert a viable period for either a periodic push
      cadence or an on-change dampening interval.";
  }
  leaf filter-failure-hint {
    type string;
    description
      "Information describing where and/or why a provided filter
      was unsupportable for a subscription.";
  }
  leaf object-count-estimate {
    type uint32;
    description
      "If there are too many objects which could potentially be
      returned by the selection filter, this identifies the
      estimate of the number of objects which the filter would
      potentially pass.";
  }
  leaf object-count-limit {
    type uint32;
    description
      "If there are too many objects which could be returned by
      the selection filter, this identifies the upper limit of
      the publisher's ability to service for this subscription.";
  }
  leaf kilobytes-estimate {
    type uint32;
    description
      "If the returned information could be beyond the capacity
      of the publisher, this would identify the data size which
      could result from this selection filter.";
  }
  leaf kilobytes-limit {
    type uint32;
    description
      "If the returned information would be beyond the capacity
      of the publisher, this identifies the upper limit of the
      publisher's ability to service for this subscription.";
  }
}
```

```
/*
 * RPCs
 */

rpc resync-subscription {
  if-feature "on-change";
  description
    "This RPC allows a subscriber of an active on-change
    subscription to request a full push of objects.

    A successful invocation results in a push-update of all
    datastore nodes that the subscriber is permitted to access.
    This RPC can only be invoked on the same session on which the
    subscription is currently active. In case of an error, a
    resync-subscription-error is sent as part of an error
    response.";
  input {
    leaf id {
      type sn:subscription-id;
      mandatory true;
      description
        "Identifier of the subscription that is to be resynced.";
    }
  }
}

rc:yang-data resync-subscription-error {
  container resync-subscription-error {
    description
      "If a 'resync-subscription' RPC fails, the subscription is
      not resynced and the RPC error response MUST indicate the
      reason for this failure. This YANG-data MAY be inserted as
      structured data within a subscription's RPC error response
      to indicate the failure reason.";
    leaf reason {
      type identityref {
        base resync-subscription-error;
      }
      mandatory true;
      description
        "Indicates the reason why the publisher has declined a
        request for subscription resynchronization.";
    }
    uses hints;
  }
}

augment "/sn:establish-subscription/sn:input" {
```

```
    description
      "This augmentation adds additional subscription parameters
       that apply specifically to datastore updates to RPC input.";
    uses update-policy;
  }

  augment "/sn:establish-subscription/sn:input/sn:target" {
    description
      "This augmentation adds the datastore as a valid target
       for the subscription to RPC input.";
    case datastore {
      description
        "Information specifying the parameters of an request for a
         datastore subscription.";
      uses datastore-criteria;
    }
  }

  rc:yang-data establish-subscription-datastore-error-info {
    container establish-subscription-datastore-error-info {
      description
        "If any 'establish-subscription' RPC parameters are
         unsupportable against the datastore, a subscription is not
         created and the RPC error response MUST indicate the reason
         why the subscription failed to be created.  This YANG-data
         MAY be inserted as structured data within a subscription's
         RPC error response to indicate the failure reason.  This
         YANG-data MUST be inserted if hints are to be provided back
         to the subscriber.";
      leaf reason {
        type identityref {
          base sn:establish-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
           be created to a targeted datastore.";
      }
      uses hints;
    }
  }

  augment "/sn:modify-subscription/sn:input" {
    description
      "This augmentation adds additional subscription parameters
       specific to datastore updates.";
    uses update-policy-modifiable;
  }
```

```
augment "/sn:modify-subscription/sn:input/sn:target" {
  description
    "This augmentation adds the datastore as a valid target
    for the subscription to RPC input.";
  case datastore {
    description
      "Information specifying the parameters of an request for a
      datastore subscription.";
    uses datastore-criteria;
  }
}

rc:yang-data modify-subscription-datastore-error-info {
  container modify-subscription-datastore-error-info {
    description
      "This YANG-data MAY be provided as part of a subscription's
      RPC error response when there is a failure of a
      'modify-subscription' RPC which has been made against a
      datastore. This YANG-data MUST be used if hints are to be
      provides back to the subscriber.";
    leaf reason {
      type identityref {
        base sn:modify-subscription-error;
      }
      description
        "Indicates the reason why the subscription has failed to
        be modified.";
    }
    uses hints;
  }
}

/*
 * NOTIFICATIONS
 */

notification push-update {
  description
    "This notification contains a push update, containing data
    subscribed to via a subscription. This notification is sent
    for periodic updates, for a periodic subscription. It can
    also be used for synchronization updates of an on-change
    subscription. This notification shall only be sent to
    receivers of a subscription. It does not constitute a
    general-purpose notification that would be subscribable as
    part of the NETCONF event stream by any receiver.";
  leaf id {
    type sn:subscription-id;
  }
}
```

```
    description
      "This references the subscription which drove the
        notification to be sent.";
  }
  anydata datastore-contents {
    description
      "This contains the updated data. It constitutes a snapshot
        at the time-of-update of the set of data that has been
        subscribed to. The snapshot corresponds to the same
        snapshot that would be returned in a corresponding get
        operation with the same selection filter parameters
        applied.";
  }
  leaf incomplete-update {
    type empty;
    description
      "This is a flag which indicates that not all datastore
        nodes subscribed to are included with this update. In
        other words, the publisher has failed to fulfill its full
        subscription obligations, and despite its best efforts is
        providing an incomplete set of objects.";
  }
}

notification push-change-update {
  if-feature "on-change";
  description
    "This notification contains an on-change push update. This
      notification shall only be sent to the receivers of a
      subscription. It does not constitute a general-purpose
      notification that would be subscribable as part of the
      NETCONF event stream by any receiver.";
  leaf id {
    type sn:subscription-id;
    description
      "This references the subscription which drove the
        notification to be sent.";
  }
  container datastore-changes {
    description
      "This contains the set of datastore changes of the target
        datastore starting at the time of the previous update, per
        the terms of the subscription.";
    uses ypatch:yang-patch;
  }
  leaf incomplete-update {
    type empty;
    description
```



```
        "The presence of this object indicates not all changes which
        have occurred since the last update are included with this
        update.  In other words, the publisher has failed to
        fulfill its full subscription obligations, for example in
        cases where it was not able to keep up with a change
        burst.";
    }
}

augment "/sn:subscription-started" {
    description
        "This augmentation adds datastore-specific objects to
        the notification that a subscription has started.";
    uses update-policy;
}

augment "/sn:subscription-started/sn:target" {
    description
        "This augmentation allows the datastore to be included as
        part of the notification that a subscription has started.";
    case datastore {
        uses datastore-criteria {
            refine "selection-filter/within-subscription" {
                description
                    "Specifies the selection filter and where it originated
                    from.  If the 'selection-filter-ref' is populated, the
                    filter within the subscription came from the 'filters'
                    container.  Otherwise it is populated in-line as part of
                    the subscription itself.";
            }
        }
    }
}

augment "/sn:subscription-modified" {
    description
        "This augmentation adds datastore-specific objects to
        the notification that a subscription has been modified.";
    uses update-policy;
}

augment "/sn:subscription-modified/sn:target" {
    description
        "This augmentation allows the datastore to be included as
        part of the notification that a subscription has been
        modified.";
    case datastore {
        uses datastore-criteria {
```

```
    refine "selection-filter/within-subscription" {
        description
            "Specifies the selection filter and where it originated
            from. If the 'selection-filter-ref' is populated, the
            filter within the subscription came from the 'filters'
            container. Otherwise it is populated in-line as part of
            the subscription itself.";
    }
}
}
}

/*
 * DATA NODES
 */

augment "/sn:filters" {
    description
        "This augmentation allows the datastore to be included as part
        of the selection filtering criteria for a subscription.";
    list selection-filter {
        key "filter-id";
        description
            "A list of pre-configured filters that can be applied
            to datastore subscriptions.";
        leaf filter-id {
            type string;
            description
                "An identifier to differentiate between selection
                filters.";
        }
        uses selection-filter-types;
    }
}

augment "/sn:subscriptions/sn:subscription" {
    when 'yp:datastore';
    description
        "This augmentation adds many datastore specific objects to a
        subscription.";
    uses update-policy;
}

augment "/sn:subscriptions/sn:subscription/sn:target" {
    description
        "This augmentation allows the datastore to be included as
        part of the selection filtering criteria for a subscription.";
    case datastore {
```

```
        uses datastore-criteria;
    }
}
}

<CODE ENDS>
```

6. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

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

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

Name: ietf-yang-push
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-push
Prefix: yp
Reference: draft-ietf-netconf-yang-push-21.txt (RFC form)

7. Security Considerations

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

The 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.

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

[I-D.draft-ietf-netconf-subscribed-notifications]. All security considerations that are listed there are relevant also for datastore subscriptions. In the following, we focus on the data nodes that are newly introduced here.)

- o Subtree "selection-filter" under container "filters": This subtree allows to specify which objects or subtrees to include in a datastore subscription. An attacker could attempt to modify the filter. For example, the filter might be modified to result in very few objects being filtered in order to attempt to overwhelm the receiver. Alternatively, the filter might be modified to result in certain objects to be excluded from updates, in order to have certain changes go unnoticed.
- o Subtree "datastore" in choice "target" in list "subscription": Analogous to "selection filter", an attacker might attempt to modify the objects being filtered in order to overwhelm a receiver with a larger volume of object updates than expected, or to have certain changes go unnoticed.
- o Choice "update-trigger" in list "subscription": By modifying the update trigger, an attacker might alter the updates that are being sent in order to confuse a receiver, to withhold certain updates to be sent to the receiver, and/or to overwhelm a receiver. For example, an attacker might modify the period with which updates are reported for a periodic subscription, or it might modify the dampening period for an on-change subscription, resulting in greater delay of successive updates (potentially affecting responsiveness of applications that depend on the updates) or in a high volume of updates (to exhaust receiver resources).
- o RPC "resync-subscription": This RPC allows a subscriber of an on-change subscription to request a full push of objects in the subscription's scope. This can result in a large volume of data. An attacker could attempt to use this RPC to exhaust resources on the server to generate the data, and attempt to overwhelm a receiver with the resulting data volume.

NACM provides one means to mitigate these threats on the publisher side. In order to address those threats as a subscriber, a subscriber could monitor the subscription configuration for any unexpected changes. For this, it can subscribe to updates to the YANG datastore nodes that represent his datastore subscriptions. As this data volume is small, a paranoid subscriber could even revert to occasional polling to guard against a compromised subscription against subscription configuration updates itself.

8. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Martin Bjorklund, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Guangying Zheng, Tom Petch, Henk Birkholz, Reshad Rahman, Qin Wu, Rohit Ranade, and Rob Wilton.

9. Contributors

Alberto Gonzalez Prieto
Microsoft
albgonz@microsoft.com

Ambika Prasad Tripathy
Cisco Systems
ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems
einarnn@cisco.com

Andy Bierman
YumaWorks
andy@yumaworks.com

Balazs Lengyel
Ericsson
balazs.lengyel@ericsson.com

10. References

10.1. Normative References

- [I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A.,
and E. Nilsen-Nygaard, "Subscription to YANG Event
Notifications", draft-ietf-netconf-subscribed-
notifications-24 (work in progress), April 2019.
- [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>>.

- [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>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <<https://www.rfc-editor.org/info/rfc8072>>.
- [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>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.

10.2. Informative References

- [I-D.draft-ietf-netconf-netconf-event-notifications] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Dynamic subscription to YANG Events and Datastores over NETCONF", April 2019.

- [I-D.draft-ietf-netconf-notification-capabilities]
Lengyel, B. and A. Clemm, "YangPush Notification Capabilities", March 2019.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<https://www.rfc-editor.org/info/rfc7923>>.
- [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>>.
- [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>>.

Appendix A. Appendix A: Subscription Errors

A.1. RPC Failures

Rejection of an RPC for any reason is indicated by via RPC error response from the publisher. Valid RPC errors returned include both existing transport layer RPC error codes, such as those seen with NETCONF in [RFC6241], as well as subscription specific errors such as those defined within the YANG model. As a result, how subscription errors are encoded within an RPC error response is transport dependent.

References to specific identities in the ietf-subscribed-notifications YANG model or the ietf-yang-push YANG model may be returned as part of the error responses resulting from failed attempts at datastore subscription. For errors defined as part of ietf-subscribed-notifications, please refer to

[I-D.draft-ietf-netconf-subscribed-notifications]. The errors introduced in this document, grouped per RPC, are as follows:

establish-subscription	modify-subscription
-----	-----
cant-exclude	period-unsupported
datastore-not-subscribable	update-too-big
on-change-unsupported	sync-too-big
on-change-sync-unsupported	unchanging-selection
period-unsupported	
update-too-big	resync-subscription
sync-too-big	-----
unchanging-selection	no-such-subscription-resync
	sync-too-big

There is one final set of transport independent RPC error elements included in the YANG model. These are the following four YANG-data structures for failed datastore subscriptions:

1. YANG-data establish-subscription-error-datastore
This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints are included.
2. YANG-data modify-subscription-error-datastore
This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "modifiy-subscription" RPC response. This MUST be sent if hints are included.
3. YANG-data sn:delete-subscription-error
This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.
4. YANG-data resync-subscription-error
This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "resync-subscription" RPC response.

A.2. Notifications of Failure

A subscription may be unexpectedly terminated or suspended independent of any RPC or configuration operation. In such cases, indications of such a failure MUST be provided. To accomplish this, a number of errors can be returned as part of the corresponding subscription state change notification. For this purpose, the following error identities have been introduced in this document, in addition to those that were already defined in [I-D.draft-ietf-netconf-subscribed-notifications]:

subscription-terminated -----	subscription-suspended -----
datastore-not-subscribable unchanging-selection	period-unsupported update-too-big synchronization-size

Appendix B. Changes Between Revisions

(To be removed by RFC editor prior to publication)

v24 - v25

- o Minor updates to address IESG review comment regarding referencing the draft which addresses the notification capabilities problem.

v23 - v24

- o Minor updates to address IESG review comments. Moving five of the coauthors to contributors as requested.

v22 - v23

- o Minor updates to address IESG review comments.

v21 - v22

- o Minor updates per Martin Bjorklund's YANG doctor review.

v20 - v21

- o Minor updates, simplifying RPC input conditions.

v19 - v20

- o Minor updates per WGLC comments.

v18 - v19

- o Minor updates per WGLC comments.

v17 - v18

- o Minor updates per WGLC comments.

v16 - v17

- o Minor updates to YANG module, incorporating comments from Tom Petch.
- o Updated references.

v15 - v16

- o Updated security considerations.
- o Updated references.
- o Addressed comments from last call review, specifically comments received from Martin Bjorklund.

v14 - v15

- o Minor text fixes. Includes a fix to on-change update calculation to cover churn when an object changes to and from a value during a dampening period.

v13 - v14

- o Minor text fixes.

v12 - v13

- o Hint negotiation models now show error examples.
- o yang-data structures for rpc errors.

v11 - v12

- o Included Martin's review clarifications.
- o QoS moved to subscribed-notifications
- o time-of-update removed as it is redundant with RFC5277's eventTime, and other times from notification-messages.

- o Error model moved to match existing implementations
- o On-change notifiable removed, how to do this is implementation specific.
- o NMDA model supported. Non NMDA version at <https://github.com/netconf-wg/yang-push/>

v10 - v11

- o Promise model reference added.
- o Error added for no-such-datastore
- o Inherited changes from subscribed notifications (such as optional feature definitions).
- o scrubbed the examples for proper encodings

v09 - v10

- o Returned to the explicit filter subtyping of v00-v05
- o identityref to ds:datastore made explicit
- o Returned ability to modify a selection filter via RPC.

v08 - v09

- o Minor tweaks cleaning up text, removing appendices, and making reference to revised-datastores.
- o Subscription-id (now:id) optional in push updates, except when encoded in RFC5277, Section 4 one-way notification.
- o Finished adding the text describing the resync subscription RPC.
- o Removed relationships to other drafts and future technology appendices as this work is being explored elsewhere.
- o Deferred the multi-line card issue to new drafts
- o Simplified the NACM interactions.

v07 - v08

- o Updated YANG models with minor tweaks to accommodate changes of ietf-subscribed-notifications.

v06 - v07

- o Clarifying text tweaks.
- o Clarification that filters act as selectors for subscribed datastore nodes; support for value filters not included but possible as a future extension
- o Filters don't have to be matched to existing YANG objects

v05 - v06

- o Security considerations updated.
- o Base YANG model in [subscribe] updated as part of move to identities, YANG augmentations in this doc matched up
- o Terms refined and text updates throughout
- o Appendix talking about relationship to other drafts added.
- o Datastore replaces stream
- o Definitions of filters improved

v04 to v05

- o Referenced based subscription document changed to Subscribed Notifications from 5277bis.
- o Getting operational data from filters
- o Extension notifiable-on-change added
- o New appendix on potential futures. Moved text into there from several drafts.
- o Subscription configuration section now just includes changed parameters from Subscribed Notifications
- o Subscription monitoring moved into Subscribed Notifications
- o New error and hint mechanisms included in text and in the YANG model.
- o Updated examples based on the error definitions
- o Groupings updated for consistency

- o Text updates throughout v03 to v04
- o Updates-not-sent flag added
- o Not notifiable extension added
- o Dampening period is for whole subscription, not single objects
- o Moved start/stop into rfc5277bis
- o Client and Server changed to subscriber, publisher, and receiver
- o Anchor time for periodic
- o Message format for synchronization (i.e. sync-on-start)
- o Material moved into 5277bis
- o QoS parameters supported, by not allowed to be modified by RPC
- o Text updates throughout

Authors' Addresses

Alexander Clemm
Futurewei

Email: ludwig@clemm.org

Eric Voit
Cisco Systems

Email: evoit@cisco.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: May 10, 2019

M. Jethanandani
VMware
S. Agarwal
Cisco Systems, Inc.
L. Huang

D. Blair
November 6, 2018

Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-21

Abstract

This document defines a data model for Access Control List (ACL). An ACL is a user-ordered set of rules, used to configure the forwarding behavior in device. Each rule is used to find a match on a packet, and define actions that will be performed on the packet.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 10, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Definitions and Acronyms	4
1.2. Terminology	4
1.3. Tree Diagram	4
2. Problem Statement	4
3. Understanding ACL's Filters and Actions	5
3.1. ACL Modules	6
4. ACL YANG Models	10
4.1. IETF Access Control List module	10
4.2. IETF Packet Fields module	24
4.3. ACL Examples	37
4.4. Port Range Usage and Other Examples	39
5. Security Considerations	43
6. IANA Considerations	44
6.1. URI Registration	44
6.2. YANG Module Name Registration	44
7. Acknowledgements	45
8. References	45
8.1. Normative References	45
8.2. Informative References	47
Appendix A. Extending ACL model examples	48
A.1. A company proprietary module example	48
A.2. Linux nftables	51
A.3. Ethertypes	52
Authors' Addresses	60

1. Introduction

Access Control List (ACL) is one of the basic elements used to configure device forwarding behavior. It is used in many networking technologies such as Policy Based Routing (PBR), firewalls etc.

An ACL is an user-ordered set of rules, that is used to filter traffic on a networking device. Each rule is represented by an Access Control Entry (ACE).

Each ACE has a group of match criteria and a group of actions.

The match criteria allow for definition of packet headers and metadata, the contents of which must match the definitions.

- o Packet header matches apply to fields visible in the packet such as address or Class of Service (CoS) or port numbers.
- o In case a vendor supports it, metadata matches apply to fields associated with the packet but not in the packet header such as input interface or length of the packet as received over the wire.

The actions specify what to do with the packet when the matching criteria are met. These actions are any operations that would apply to the packet, such as counting, policing, or simply forwarding. The list of potential actions is unbounded depending on the capabilities of the networking devices.

Access Control List is also widely known as ACL (pronounce as [ak-uh l]) or Access List. In this document, Access Control List, ACL and Access List are used interchangeably.

The matching of filters and actions in an ACE/ACL are triggered only after the application/attachment of the ACL to an interface, VRF, vty/tty session, QoS policy, or routing protocols, amongst various other configuration attachment points. Once attached, it is used for filtering traffic using the match criteria in the ACEs and taking appropriate action(s) that have been configured against that ACE. In order to apply an ACL to any attachment point other than an interface, vendors would have to augment the ACL YANG model.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. Please note that no other RFC Editor instructions are specified anywhere else in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements

- o "XXXX" --> the assigned RFC value for this draft both in this draft and in the YANG models under the revision statement.
- o Revision date in model, in the format 2018-11-06 needs to get updated with the date the draft gets approved. The date also needs to get reflected on the line with <CODE BEGINS>.

1.1. Definitions and Acronyms

ACE: Access Control Entry

ACL: Access Control List

CoS: Class of Service

DSCP: Differentiated Services Code Point

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

MAC: Media Access Control

PBR: Policy Based Routing

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

1.2. Terminology

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.

1.3. Tree Diagram

For a reference to the annotations used in tree diagrams included in this draft, please see YANG Tree Diagrams [RFC8340].

2. Problem Statement

This document defines a YANG 1.1 [RFC7950] data model for the configuration of ACLs. The model defines matching rules for commonly used protocols such as, Ethernet, IPv4, IPv6, TCP, UDP and ICMP. If more protocols need to be supported in the future, this base model can be augmented. An example of such an augmentation can be seen in the Appendix.

ACL implementations in every device may vary greatly in terms of the filter constructs and actions that they support. Therefore, this draft proposes a model that can be augmented by standard extensions and vendor proprietary models.

3. Understanding ACL's Filters and Actions

Although different vendors have different ACL data models, there is a common understanding of what Access Control List (ACL) is. A network system usually has a list of ACLs, and each ACL contains an ordered list of rules, also known as Access Control Entries (ACE). Each ACE has a group of match criteria and a group of actions. The match criteria allow for definition of contents of the packet headers or metadata, if supported by the vendor. Packet header matching applies to fields visible in the packet such as address or CoS or port numbers. Metadata matching applies to fields associated with the packet, but not in the packet header, such as input interface, packet length, or source or destination prefix length. The actions can be any sort of operation from logging to rate limiting or dropping to simply forwarding. Actions on the first matching ACE are applied with no processing of subsequent ACEs.

The model also includes a container to hold overall operational state for each ACL and operational state for each ACE. One ACL can be applied to multiple targets within the device, such as interface of a networking device, applications or features running in the device, etc. When applied to interfaces of a networked device, distinct ACLs are defined for the ingress (input) or egress (output) interface.

This draft tries to address the commonalities between all vendors and create a common model, which can be augmented with proprietary models. The base model is simple in design, and we hope to achieve enough flexibility for each vendor to extend the base model.

The use of feature statements in the model allows vendors to advertise match rules they are capable and willing to support. There are two sets of feature statements a device needs to advertise. The first set of feature statements specify the capability of the device. These include features such as "Device can support matching on Ethernet headers" or "Device can support matching on IPv4 headers". The second set of feature statements specify the combinations of headers the device is willing to support. These include features such as "Plain IPv6 ACL supported" or "Ethernet, IPv4 and IPv6 ACL combinations supported".

3.1. ACL Modules

There are two YANG modules in the model. The first module, "ietf-access-control-list", defines generic ACL aspects which are common to all ACLs regardless of their type or vendor. In effect, the module can be viewed as providing a generic ACL "superclass". It imports the second module, "ietf-packet-fields". The match container in "ietf-access-control-list" uses groupings in "ietf-packet-fields" to specify match fields such as port numbers or protocol. The combination of 'if-feature' checks and 'must' statements allow for the selection of relevant match fields that a user can define rules for.

If there is a need to define a new "matches" choice, such as IPFIX [RFC7011], the container "matches" can be augmented.

```

module: ietf-access-control-list
  +--rw acls
    +--rw acl* [name]
      +--rw name      string
      +--rw type?     acl-type
      +--rw aces
        +--rw ace* [name]
          +--rw name      string
          +--rw matches
            +--rw (l2)?
              +--:(eth)
                +--rw eth {match-on-eth}?
                  +--rw destination-mac-address?
                    |      yang:mac-address
                  +--rw destination-mac-address-mask?
                    |      yang:mac-address
                  +--rw source-mac-address?
                    |      yang:mac-address
                  +--rw source-mac-address-mask?
                    |      yang:mac-address
                  +--rw ethertype?
                    |      eth:ethertype
            +--rw (l3)?
              +--:(ipv4)
                +--rw ipv4 {match-on-ipv4}?
                  +--rw dscp?
                    |      inet:dscp
                  +--rw ecn?
                    |      uint8
                  +--rw length?
                    |      uint16
                  +--rw ttl?

```

```

|         uint8
+--rw protocol?
|         uint8
+--rw ihl?
|         uint8
+--rw flags?
|         bits
+--rw offset?
|         uint16
+--rw identification?
|         uint16
+--rw (destination-network)?
|   +--:(destination-ipv4-network)
|     +--rw destination-ipv4-network?
|       inet:ipv4-prefix
+--rw (source-network)?
|   +--:(source-ipv4-network)
|     +--rw source-ipv4-network?
|       inet:ipv4-prefix
+--:(ipv6)
+--rw ipv6 {match-on-ipv6}?
+--rw dscp?
|   inet:dscp
+--rw ecn?
|   uint8
+--rw length?
|   uint16
+--rw ttl?
|   uint8
+--rw protocol?
|   uint8
+--rw (destination-network)?
|   +--:(destination-ipv6-network)
|     +--rw destination-ipv6-network?
|       inet:ipv6-prefix
+--rw (source-network)?
|   +--:(source-ipv6-network)
|     +--rw source-ipv6-network?
|       inet:ipv6-prefix
+--rw flow-label?
|   inet:ipv6-flow-label
+--rw (l4)?
+--:(tcp)
+--rw tcp {match-on-tcp}?
+--rw sequence-number?      uint32
+--rw acknowledgement-number?  uint32
+--rw data-offset?          uint8
+--rw reserved?             uint8

```

```

+--rw flags?                               bits
+--rw window-size?                         uint16
+--rw urgent-pointer?                      uint16
+--rw options?                             binary
+--rw source-port
|   +--rw (source-port)?
|   |   +--:(range-or-operator)
|   |   |   +--rw (port-range-or-operator)?
|   |   |   |   +--:(range)
|   |   |   |   |   +--rw lower-port
|   |   |   |   |   |   inet:port-number
|   |   |   |   |   +--rw upper-port
|   |   |   |   |   |   inet:port-number
|   |   |   |   +--:(operator)
|   |   |   |   |   +--rw operator?      operator
|   |   |   |   |   +--rw port
|   |   |   |   |   |   inet:port-number
|   |   +--rw destination-port
|   |   |   +--rw (destination-port)?
|   |   |   |   +--:(range-or-operator)
|   |   |   |   |   +--rw (port-range-or-operator)?
|   |   |   |   |   |   +--:(range)
|   |   |   |   |   |   |   +--rw lower-port
|   |   |   |   |   |   |   |   inet:port-number
|   |   |   |   |   |   |   +--rw upper-port
|   |   |   |   |   |   |   |   inet:port-number
|   |   |   |   |   +--:(operator)
|   |   |   |   |   |   +--rw operator?      operator
|   |   |   |   |   |   +--rw port
|   |   |   |   |   |   |   inet:port-number
+--:(udp)
+--rw udp {match-on-udp}?
+--rw length?                             uint16
+--rw source-port
|   +--rw (source-port)?
|   |   +--:(range-or-operator)
|   |   |   +--rw (port-range-or-operator)?
|   |   |   |   +--:(range)
|   |   |   |   |   +--rw lower-port
|   |   |   |   |   |   inet:port-number
|   |   |   |   |   +--rw upper-port
|   |   |   |   |   |   inet:port-number
|   |   |   |   +--:(operator)
|   |   |   |   |   +--rw operator?      operator
|   |   |   |   |   +--rw port
|   |   |   |   |   |   inet:port-number
+--rw destination-port
+--rw (destination-port)?

```

```

      +---:(range-or-operator)
      +---rw (port-range-or-operator)?
      +---:(range)
      |   +---rw lower-port
      |   |       inet:port-number
      |   +---rw upper-port
      |       inet:port-number
      +---:(operator)
      +---rw operator?      operator
      +---rw port
      |       inet:port-number
      +---:(icmp)
      +---rw icmp {match-on-icmp}?
      |   +---rw type?      uint8
      |   +---rw code?      uint8
      |   +---rw rest-of-header?  binary
      +---rw egress-interface?  if:interface-ref
      +---rw ingress-interface? if:interface-ref
      +---rw actions
      |   +---rw forwarding  identityref
      |   +---rw logging?    identityref
      +---ro statistics {acl-aggregate-stats}?
      +---ro matched-packets?  yang:counter64
      +---ro matched-octets?   yang:counter64
+---rw attachment-points
+---rw interface* [interface-id] {interface-attachment}?
+---rw interface-id  if:interface-ref
+---rw ingress
+---rw acl-sets
+---rw acl-set* [name]
+---rw name          -> /acls/acl/name
+---ro ace-statistics* [name] {interface-stats}?
+---ro name
+---ro name          -> /acls/acl/aces/ace/name
+---ro matched-packets?  yang:counter64
+---ro matched-octets?   yang:counter64
+---rw egress
+---rw acl-sets
+---rw acl-set* [name]
+---rw name          -> /acls/acl/name
+---ro ace-statistics* [name] {interface-stats}?
+---ro name
+---ro name          -> /acls/acl/aces/ace/name
+---ro matched-packets?  yang:counter64
+---ro matched-octets?   yang:counter64

```

4. ACL YANG Models

4.1. IETF Access Control List module

"ietf-access-control-list" module defines the "acls" container that has a list of "acl". Each "acl" has information identifying the access list by a name ("name") and a list ("aces") of rules associated with the "name". Each of the entries in the list ("aces"), indexed by the string "name", has containers defining "matches" and "actions".

The model defines several ACL types and actions in the form of identities and features. Features are used by implementors to select the ACL types the system can support and identities are used to validate the types that have been selected. These types are implicitly inherited by the "ace", thus safeguarding against misconfiguration of "ace" types in an "acl".

The "matches" define criteria used to identify patterns in "ietf-packet-fields". The choice statements within the match container allow for selection of one header within each of "l2", "l3", or "l4" headers. The "actions" define behavior to undertake once a "match" has been identified. In addition to permit and deny for actions, a logging option allows for a match to be logged that can later be used to determine which rule was matched upon. The model also defines the ability for ACLs to be attached to a particular interface.

Statistics in the ACL can be collected for an "ace" or for an "interface". The feature statements defined for statistics can be used to determine whether statistics are being collected per "ace", or per "interface".

This module imports definitions from Common YANG Data Types [RFC6991], and A YANG Data Model for Interface Management [RFC8343].

<CODE BEGINS> file "ietf-access-control-list@2018-11-06.yang"

```
module ietf-access-control-list {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-access-control-list";
  prefix acl;

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

```
import ietf-packet-fields {
  prefix pf;
  reference
    "RFC XXXX - Network ACL YANG Model.";
}

import ietf-interfaces {
  prefix if;
  reference
    "RFC 8343 - A YANG Data Model for Interface Management.";
}

organization
  "IETF NETMOD (Network Modeling Language)
   Working Group";

contact
  "WG Web: http://tools.ietf.org/wg/netmod/
   WG List: netmod@ietf.org

   Editor: Mahesh Jethanandani
           mjethanandani@gmail.com
   Editor: Lisa Huang
           lyihuang16@gmail.com
   Editor: Sonal Agarwal
           sagarwall12@gmail.com
   Editor: Dana Blair
           dblair@cisco.com";

description
  "This YANG module defines a component that describe the
   configuration and monitoring of Access Control Lists (ACLs).

   Copyright (c) 2018 IETF Trust and the persons identified as
   the document authors. All rights reserved.
   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD
   License set forth in Section 4.c of the IETF Trust's Legal
   Provisions Relating to IETF Documents
   (http://trustee.ietf.org/license-info).

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

revision 2018-11-06 {
  description
    "Initial version.";
```



```
    reference
      "RFC XXX: Network Access Control List (ACL) YANG Data Model.";
  }

  /*
   * Identities
   */

  /*
   * Forwarding actions for a packet
   */
  identity forwarding-action {
    description
      "Base identity for actions in the forwarding category";
  }

  identity accept {
    base forwarding-action;
    description
      "Accept the packet";
  }

  identity drop {
    base forwarding-action;
    description
      "Drop packet without sending any ICMP error message";
  }

  identity reject {
    base forwarding-action;
    description
      "Drop the packet and send an ICMP error message to the source";
  }

  /*
   * Logging actions for a packet
   */
  identity log-action {
    description
      "Base identity for defining the destination for logging actions";
  }

  identity log-syslog {
    base log-action;
    description
      "System log (syslog) the information for the packet";
  }
```

```
identity log-none {
  base log-action;
  description
    "No logging for the packet";
}

/*
 * ACL type identities
 */
identity acl-base {
  description
    "Base Access Control List type for all Access Control List type
    identifiers.";
}

identity ipv4-acl-type {
  base acl:acl-base;
  if-feature "ipv4";
  description
    "An ACL that matches on fields from the IPv4 header
    (e.g. IPv4 destination address) and layer 4 headers (e.g. TCP
    destination port). An acl of type ipv4 does not contain
    matches on fields in the ethernet header or the IPv6 header.";
}

identity ipv6-acl-type {
  base acl:acl-base;
  if-feature "ipv6";
  description
    "An ACL that matches on fields from the IPv6 header
    (e.g. IPv6 destination address) and layer 4 headers (e.g. TCP
    destination port). An acl of type ipv6 does not contain
    matches on fields in the ethernet header or the IPv4 header.";
}

identity eth-acl-type {
  base acl:acl-base;
  if-feature "eth";
  description
    "An ACL that matches on fields in the ethernet header,
    like 10/100/1000baseT or WiFi Access Control List. An acl of
    type ethernet does not contain matches on fields in the IPv4
    header, IPv6 header or layer 4 headers.";
}

identity mixed-eth-ipv4-acl-type {
  base "acl:eth-acl-type";
  base "acl:ipv4-acl-type";
}
```

```
    if-feature "mixed-eth-ipv4";
    description
        "An ACL that contains a mix of entries that
        match on fields in ethernet headers,
        entries that match on IPv4 headers.
        Matching on layer 4 header fields may also exist in the
        list.";
}

identity mixed-eth-ipv6-acl-type {
    base "acl:eth-acl-type";
    base "acl:ipv6-acl-type";
    if-feature "mixed-eth-ipv6";
    description
        "ACL that contains a mix of entries that
        match on fields in ethernet headers, entries
        that match on fields in IPv6 headers. Matching on
        layer 4 header fields may also exist in the list.";
}

identity mixed-eth-ipv4-ipv6-acl-type {
    base "acl:eth-acl-type";
    base "acl:ipv4-acl-type";
    base "acl:ipv6-acl-type";
    if-feature "mixed-eth-ipv4-ipv6";
    description
        "ACL that contains a mix of entries that
        match on fields in ethernet headers, entries
        that match on fields in IPv4 headers, and entries
        that match on fields in IPv6 headers. Matching on
        layer 4 header fields may also exist in the list.";
}

/*
 * Features
 */

/*
 * Features supported by device
 */
feature match-on-eth {
    description
        "The device can support matching on ethernet headers.";
}

feature match-on-ipv4 {
    description
        "The device can support matching on IPv4 headers.";
```

```
    }

    feature match-on-ipv6 {
        description
            "The device can support matching on IPv6 headers.";
    }

    feature match-on-tcp {
        description
            "The device can support matching on TCP headers.";
    }

    feature match-on-udp {
        description
            "The device can support matching on UDP headers.";
    }

    feature match-on-icmp {
        description
            "The device can support matching on ICMP (v4 and v6) headers.";
    }

    /*
     * Header classifications combinations supported by
     * device
     */
    feature eth {
        if-feature "match-on-eth";
        description
            "Plain Ethernet ACL supported";
    }

    feature ipv4 {
        if-feature "match-on-ipv4";
        description
            "Plain IPv4 ACL supported";
    }

    feature ipv6 {
        if-feature "match-on-ipv6";
        description
            "Plain IPv6 ACL supported";
    }

    feature mixed-eth-ipv4 {
        if-feature "match-on-eth and match-on-ipv4";
        description
            "Ethernet and IPv4 ACL combinations supported";
    }
```

```
}

feature mixed-eth-ipv6 {
  if-feature "match-on-eth and match-on-ipv6";
  description
    "Ethernet and IPv6 ACL combinations supported";
}

feature mixed-eth-ipv4-ipv6 {
  if-feature "match-on-eth and match-on-ipv4
    and match-on-ipv6";
  description
    "Ethernet, IPv4 and IPv6 ACL combinations supported.";
}

/*
 * Stats Features
 */
feature interface-stats {
  description
    "ACL counters are available and reported only per interface";
}

feature acl-aggregate-stats {
  description
    "ACL counters are aggregated over all interfaces, and reported
    only per ACL entry";
}

/*
 * Attachment point features
 */
feature interface-attachment {
  description
    "ACLs are set on interfaces.";
}

/*
 * Typedefs
 */
typedef acl-type {
  type identityref {
    base acl-base;
  }
  description
    "This type is used to refer to an Access Control List
    (ACL) type";
}
```

```
/*
 * Groupings
 */
grouping acl-counters {
  description
    "Common grouping for ACL counters";

  leaf matched-packets {
    type yang:counter64;
    config false;
    description
      "Count of the number of packets matching the current ACL
      entry.

      An implementation should provide this counter on a
      per-interface per-ACL-entry basis if possible.

      If an implementation only supports ACL counters on a per
      entry basis (i.e., not broken out per interface), then the
      value should be equal to the aggregate count across all
      interfaces.

      An implementation that provides counters on a per entry per
      interface basis is not required to also provide an aggregate
      count, e.g., per entry -- the user is expected to be able
      implement the required aggregation if such a count is
      needed.";
  }

  leaf matched-octets {
    type yang:counter64;
    config false;
    description
      "Count of the number of octets (bytes) matching the current
      ACL entry.

      An implementation should provide this counter on a
      per-interface per-ACL-entry if possible.

      If an implementation only supports ACL counters per entry
      (i.e., not broken out per interface), then the value
      should be equal to the aggregate count across all interfaces.

      An implementation that provides counters per entry per
      interface is not required to also provide an aggregate count,
      e.g., per entry -- the user is expected to be able implement
      the required aggregation if such a count is needed.";
  }
}
```

```
}

/*
 * Configuration and monitoring data nodes
 */
container acls {
  description
    "This is a top level container for Access Control Lists.
    It can have one or more acl nodes.";
  list acl {
    key "name";
    description
      "An Access Control List (ACL) is an ordered list of
      Access Control Entries (ACE). Each ACE has a
      list of match criteria and a list of actions.
      Since there are several kinds of Access Control Lists
      implemented with different attributes for
      different vendors, this model accommodates customizing
      Access Control Lists for each kind and, for each vendor.";
    leaf name {
      type string {
        length "1..64";
      }
      description
        "The name of access list. A device MAY restrict the length
        and value of this name, possibly space and special
        characters are not allowed.";
    }
    leaf type {
      type acl-type;
      description
        "Type of access control list. Indicates the primary intended
        type of match criteria (e.g. ethernet, IPv4, IPv6, mixed,
        etc) used in the list instance.";
    }
  }
  container aces {
    description
      "The aces container contains one or more ace nodes.";
    list ace {
      key "name";
      ordered-by user;
      description
        "List of Access Control Entries (ACEs)";
      leaf name {
        type string {
          length "1..64";
        }
        description

```

```
    "A unique name identifying this Access Control
      Entry (ACE).";
  }

  container matches {
    description
      "The rules in this set determine what fields will be
       matched upon before any action is taken on them.
       The rules are selected based on the feature set
       defined by the server and the acl-type defined.
       If no matches are defined in a particular container,
       then any packet will match that container. If no
       matches are specified at all in an ACE, then any
       packet will match the ACE.";

    choice l2 {
      container eth {
        when "derived-from-or-self(/acls/acl/type, " +
          "'acl:eth-acl-type')";
        if-feature match-on-eth;
        uses pf:acl-eth-header-fields;
        description
          "Rule set that matches ethernet headers.";
      }
      description
        "Match layer 2 headers, for example ethernet
         header fields.";
    }

    choice l3 {
      container ipv4 {
        when "derived-from-or-self(/acls/acl/type, " +
          "'acl:ipv4-acl-type')";
        if-feature match-on-ipv4;
        uses pf:acl-ip-header-fields;
        uses pf:acl-ipv4-header-fields;
        description
          "Rule set that matches IPv4 headers.";
      }

      container ipv6 {
        when "derived-from-or-self(/acls/acl/type, " +
          "'acl:ipv6-acl-type')";
        if-feature match-on-ipv6;
        uses pf:acl-ip-header-fields;
        uses pf:acl-ipv6-header-fields;
        description
          "Rule set that matches IPv6 headers.";
      }
    }
  }
}
```



```
    }
    description
      "Choice of either ipv4 or ipv6 headers";
  }

  choice l4 {
    container tcp {
      if-feature match-on-tcp;
      uses pf:acl-tcp-header-fields;
      container source-port {
        choice source-port {
          case range-or-operator {
            uses pf:port-range-or-operator;
            description
              "Source port definition from range or
              operator.";
          }
          description
            "Choice of source port definition using
            range/operator or a choice to support future
            'case' statements, such as one enabling a
            group of source ports to be referenced.";
        }
        description
          "Source port definition.";
      }
      container destination-port {
        choice destination-port {
          case range-or-operator {
            uses pf:port-range-or-operator;
            description
              "Destination port definition from range or
              operator.";
          }
          description
            "Choice of destination port definition using
            range/operator or a choice to support future
            'case' statements, such as one enabling a
            group of destination ports to be referenced.";
        }
        description
          "Destination port definition.";
      }
      description
        "Rule set that matches TCP headers.";
    }

    container udp {
```

```
if-feature match-on-udp;
uses pf:acl-udp-header-fields;
container source-port {
  choice source-port {
    case range-or-operator {
      uses pf:port-range-or-operator;
      description
        "Source port definition from range or
        operator.";
    }
    description
      "Choice of source port definition using
      range/operator or a choice to support future
      'case' statements, such as one enabling a
      group of source ports to be referenced.";
  }
  description
    "Source port definition.";
}
container destination-port {
  choice destination-port {
    case range-or-operator {
      uses pf:port-range-or-operator;
      description
        "Destination port definition from range or
        operator.";
    }
    description
      "Choice of destination port definition using
      range/operator or a choice to support future
      'case' statements, such as one enabling a
      group of destination ports to be referenced.";
  }
  description
    "Destination port definition.";
}
description
  "Rule set that matches UDP headers.";
}

container icmp {
  if-feature match-on-icmp;
  uses pf:acl-icmp-header-fields;
  description
    "Rule set that matches ICMP headers.";
}
description
  "Choice of TCP, UDP or ICMP headers.";
```

```
    }

    leaf egress-interface {
        type if:interface-ref;
        description
            "Egress interface. This should not be used if this ACL
            is attached as an egress ACL (or the value should
            equal the interface to which the ACL is attached).";
    }

    leaf ingress-interface {
        type if:interface-ref;
        description
            "Ingress interface. This should not be used if this ACL
            is attached as an ingress ACL (or the value should
            equal the interface to which the ACL is attached)";
    }
}

container actions {
    description
        "Definitions of action for this ace entry";
    leaf forwarding {
        type identityref {
            base forwarding-action;
        }
        mandatory true;
        description
            "Specifies the forwarding action per ace entry";
    }

    leaf logging {
        type identityref {
            base log-action;
        }
        default log-none;
        description
            "Specifies the log action and destination for
            matched packets. Default value is not to log the
            packet.";
    }
}

container statistics {
    if-feature "acl-aggregate-stats";
    config false;
    description
        "Statistics gathered across all attachment points for the
        given ACL.";
```

```
        uses acl-counters;
    }
}
}
container attachment-points {
  description
    "Enclosing container for the list of
    attachment-points on which ACLs are set";

  /*
   * Groupings
   */
  grouping interface-acl {
    description
      "Grouping for per-interface ingress ACL data";

    container acl-sets {
      description
        "Enclosing container the list of ingress ACLs on the
        interface";

      list acl-set {
        key "name";
        ordered-by user;
        description
          "List of ingress ACLs on the interface";

        leaf name {
          type leafref {
            path "/acls/acl/name";
          }
          description
            "Reference to the ACL name applied on ingress";
        }

        list ace-statistics {
          if-feature "interface-stats";
          key "name";
          config false;
          description
            "List of Access Control Entries (ACEs)";
          leaf name {
            type leafref {
              path "/acls/acl/aces/ace/name";
            }
            description
              "The ace name";
          }
        }
      }
    }
  }
}
```

```

        }
        uses acl-counters;
    }
}

list interface {
    if-feature interface-attachment;
    key "interface-id";
    description
        "List of interfaces on which ACLs are set";

    leaf interface-id {
        type if:interface-ref;
        description
            "Reference to the interface id list key";
    }

    container ingress {
        uses interface-acl;
        description
            "The ACLs applied to ingress interface";
    }
    container egress {
        uses interface-acl;
        description
            "The ACLs applied to egress interface";
    }
}
}
}
}

```

<CODE ENDS>

4.2. IETF Packet Fields module

The packet fields module defines the necessary groups for matching on fields in the packet including ethernet, ipv4, ipv6, and transport layer fields. The "type" node determines which of these fields get included for any given ACL with the exception of TCP, UDP and ICMP header fields. Those fields can be used in conjunction with any of the above layer 2 or layer 3 fields.

Since the number of match criteria are very large, the base draft does not include these directly but references them by 'uses' statement to keep the base module simple. In case more match

conditions are needed, those can be added by augmenting choices within container "matches" in ietf-access-control-list.yang model.

This module imports definitions from Common YANG Data Types [RFC6991] and references IP [RFC0791], ICMP [RFC0792], TCP [RFC0793], Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers [RFC2474], The Addition of Explicit Congestion Notification (ECN) to IP [RFC3168], , IPv6 Scoped Address Architecture [RFC4007], IPv6 Addressing Architecture [RFC4291], A Recommendation for IPv6 Address Text Representation [RFC5952], IPv6 [RFC8200].

<CODE BEGINS> file "ietf-packet-fields@2018-11-06.yang"

```
module ietf-packet-fields {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-packet-fields";
  prefix packet-fields;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991 - Common YANG Data Types.";
  }

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

  import ietf-ethertypes {
    prefix eth;
    reference
      "RFC XXXX - Network ACL YANG Model.";
  }

  organization
    "IETF NETMOD (Network Modeling Language) Working
    Group";

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
    WG List: netmod@ietf.org

    Editor: Mahesh Jethanandani
            mjethanandani@gmail.com
    Editor: Lisa Huang
            lyihuang16@gmail.com
```

Editor: Sonal Agarwal
sagarwall12@gmail.com
Editor: Dana Blair
dblair@cisco.com";

description

"This YANG module defines groupings that are used by ietf-access-control-list YANG module. Their usage is not limited to ietf-access-control-list and can be used anywhere as applicable.

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>).

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

```
revision 2018-11-06 {  
  description  
    "Initial version.";  
  reference  
    "RFC XXX: Network Access Control List (ACL) YANG Data Model.";  
}
```

```
/*  
 * Typedefs  
 */  
typedef operator {  
  type enumeration {  
    enum lte {  
      description  
        "Less than or equal.";  
    }  
    enum gte {  
      description  
        "Greater than or equal.";  
    }  
    enum eq {  
      description  
        "Equal to.";  
    }  
    enum neq {
```

```
        description
            "Not equal to.";
    }
}
description
    "The source and destination port range definitions
    can be further qualified using an operator. An
    operator is needed only if lower-port is specified
    and upper-port is not specified. The operator
    therefore further qualifies lower-port only.";
}

/*
 * Groupings
 */
grouping port-range-or-operator {
    choice port-range-or-operator {
        case range {
            leaf lower-port {
                type inet:port-number;
                must ". <= ../upper-port" {
                    error-message
                        "The lower-port must be less than or equal to
                        upper-port.";
                }
                mandatory true;
                description
                    "Lower boundry for a port.";
            }
            leaf upper-port {
                type inet:port-number;
                mandatory true;
                description
                    "Upper boundry for port.";
            }
        }
    }
    case operator {
        leaf operator {
            type operator;
            default eq;
            description
                "Operator to be applied on the port below.";
        }
        leaf port {
            type inet:port-number;
            mandatory true;
            description
                "Port number along with operator on which to
```



```
        match.";
    }
}
description
    "Choice of specifying a port range or a single
    port along with an operator.";
}
description
    "Grouping for port definitions in the form of a
    choice statement.";
}

grouping acl-ip-header-fields {
    description
        "IP header fields common to ipv4 and ipv6";
    reference
        "RFC 791: Internet Protocol.";

    leaf dscp {
        type inet:dscp;
        description
            "Differentiated Services Code Point.";
        reference
            "RFC 2474: Definition of Differentiated services field
            (DS field) in the IPv4 and IPv6 headers.";
    }

    leaf ecn {
        type uint8 {
            range 0..3;
        }
        description
            "Explicit Congestion Notification.";
        reference
            "RFC 3168: Explicit Congestion Notification.";
    }

    leaf length {
        type uint16;
        description
            "In IPv4 header field, this field is known as the Total Length.
            Total Length is the length of the datagram, measured in octets,
            including internet header and data.

            In IPv6 header field, this field is known as the Payload
            Length, the length of the IPv6 payload, i.e. the rest of
            the packet following the IPv6 header, in octets.";
        reference
    }
}
```

```
        "RFC 791: Internet Protocol,
        RFC 8200: Internet Protocol, Version 6 (IPv6) Specification.";
    }

    leaf ttl {
        type uint8;
        description
            "This field indicates the maximum time the datagram is allowed
            to remain in the internet system.  If this field contains the
            value zero, then the datagram must be dropped.

            In IPv6, this field is known as the Hop Limit.";
        reference
            "RFC 791: Internet Protocol,
            RFC 8200: Internet Protocol, Version 6 (IPv6) Specification.";
    }

    leaf protocol {
        type uint8;
        description
            "Internet Protocol number. Refers to the protocol of the
            payload. In IPv6, this field is known as 'next-header',
            and if extension headers are present, the protocol is
            present in the 'upper-layer' header.";
        reference
            "RFC 791: Internet Protocol,
            RFC 8200: Internet Protocol, Version 6 (IPv6) Specification.";
    }
}

grouping acl-ipv4-header-fields {
    description
        "Fields in IPv4 header.";

    leaf ihl {
        type uint8 {
            range "5..60";
        }
        description
            "An IPv4 header field, the Internet Header Length (IHL) is
            the length of the internet header in 32 bit words, and
            thus points to the beginning of the data. Note that the
            minimum value for a correct header is 5.";
    }

    leaf flags {
        type bits {
            bit reserved {
```

```
        position 0;
        description
            "Reserved. Must be zero.";
    }
    bit fragment {
        position 1;
        description
            "Setting value to 0 indicates may fragment, while setting
            the value to 1 indicates do not fragment.";
    }
    bit more {
        position 2;
        description
            "Setting the value to 0 indicates this is the last fragment,
            and setting the value to 1 indicates more fragments are
            coming.";
    }
}
description
    "Bit definitions for the flags field in IPv4 header.";
}

leaf offset {
    type uint16 {
        range "20..65535";
    }
    description
        "The fragment offset is measured in units of 8 octets (64 bits).
        The first fragment has offset zero. The length is 13 bits";
}

leaf identification {
    type uint16;
    description
        "An identifying value assigned by the sender to aid in
        assembling the fragments of a datagram.";
}

choice destination-network {
    case destination-ipv4-network {
        leaf destination-ipv4-network {
            type inet:ipv4-prefix;
            description
                "Destination IPv4 address prefix.";
        }
    }
    description
        "Choice of specifying a destination IPv4 address or
```

```
        referring to a group of IPv4 destination addresses.";
    }
    choice source-network {
        case source-ipv4-network {
            leaf source-ipv4-network {
                type inet:ipv4-prefix;
                description
                    "Source IPv4 address prefix.";
            }
        }
        description
            "Choice of specifying a source IPv4 address or
             referring to a group of IPv4 source addresses.";
    }
}

grouping acl-ipv6-header-fields {
    description
        "Fields in IPv6 header";

    choice destination-network {
        case destination-ipv6-network {
            leaf destination-ipv6-network {
                type inet:ipv6-prefix;
                description
                    "Destination IPv6 address prefix.";
            }
        }
        description
            "Choice of specifying a destination IPv6 address
             or referring to a group of IPv6 destination
             addresses.";
    }

    choice source-network {
        case source-ipv6-network {
            leaf source-ipv6-network {
                type inet:ipv6-prefix;
                description
                    "Source IPv6 address prefix.";
            }
        }
        description
            "Choice of specifying a source IPv6 address or
             referring to a group of IPv6 source addresses.";
    }
}

leaf flow-label {
```

```
    type inet:ipv6-flow-label;
    description
      "IPv6 Flow label.";
  }
  reference
    "RFC 4291: IP Version 6 Addressing Architecture
     RFC 4007: IPv6 Scoped Address Architecture
     RFC 5952: A Recommendation for IPv6 Address Text
     Representation";
}

grouping acl-eth-header-fields {
  description
    "Fields in Ethernet header.";

  leaf destination-mac-address {
    type yang:mac-address;
    description
      "Destination IEEE 802 MAC address.";
  }
  leaf destination-mac-address-mask {
    type yang:mac-address;
    description
      "Destination IEEE 802 MAC address mask.";
  }
  leaf source-mac-address {
    type yang:mac-address;
    description
      "Source IEEE 802 MAC address.";
  }
  leaf source-mac-address-mask {
    type yang:mac-address;
    description
      "Source IEEE 802 MAC address mask.";
  }
  leaf ethertype {
    type eth:ethertype;
    description
      "The Ethernet Type (or Length) value represented
       in the canonical order defined by IEEE 802.
       The canonical representation uses lowercase
       characters.";
    reference
      "IEEE 802-2014 Clause 9.2";
  }
  reference
    "IEEE 802: IEEE Standard for Local and Metropolitan
     Area Networks: Overview and Architecture.";
}
```

```
}

grouping acl-tcp-header-fields {
  description
    "Collection of TCP header fields that can be used to
    setup a match filter.";

  leaf sequence-number {
    type uint32;
    description
      "Sequence number that appears in the packet.";
  }

  leaf acknowledgement-number {
    type uint32;
    description
      "The acknowledgement number that appears in the
      packet.";
  }

  leaf data-offset {
    type uint8 {
      range "5..15";
    }
    description
      "Specifies the size of the TCP header in 32-bit
      words. The minimum size header is 5 words and
      the maximum is 15 words thus giving the minimum
      size of 20 bytes and maximum of 60 bytes,
      allowing for up to 40 bytes of options in the
      header.";
  }

  leaf reserved {
    type uint8;
    description
      "Reserved for future use.";
  }

  leaf flags {
    type bits {
      bit cwr {
        position 1;
        description
          "Congestion Window Reduced (CWR) flag is set by
          the sending host to indicate that it received
          a TCP segment with the ECE flag set and had
          responded in congestion control mechanism.";
      }
    }
  }
}
```

```
reference
  "RFC 3168: The Addition of Explicit Congestion
    Notification (ECN) to IP.";
}
bit ece {
  position 2;
  description
    "ECN-Echo has a dual role, depending on the value
    of the SYN flag. It indicates:
    If the SYN flag is set (1), that the TCP peer is ECN
    capable. If the SYN flag is clear (0), that a packet
    with Congestion Experienced flag set (ECN=11) in IP
    header was received during normal transmission
    (added to header by RFC 3168). This serves as an
    indication of network congestion (or impending
    congestion) to the TCP sender.";
  reference
    "RFC 3168: The Addition of Explicit Congestion
      Notification (ECN) to IP.";
}
bit urg {
  position 3;
  description
    "Indicates that the Urgent pointer field is significant.";
}
bit ack {
  position 4;
  description
    "Indicates that the Acknowledgment field is significant.
    All packets after the initial SYN packet sent by the
    client should have this flag set.";
}
bit psh {
  position 5;
  description
    "Push function. Asks to push the buffered data to the
    receiving application.";
}
bit rst {
  position 6;
  description
    "Reset the connection.";
}
bit syn {
  position 7;
  description
    "Synchronize sequence numbers. Only the first packet
    sent from each end should have this flag set. Some
```

```
        other flags and fields change meaning based on this
        flag, and some are only valid for when it is set,
        and others when it is clear.";
    }
    bit fin {
        position 8;
        description
            "Last package from sender.";
    }
}
description
    "Also known as Control Bits. Contains 9 1-bit flags.";
reference
    "RFC 793: Transmission Control Protocol (TCP).";
}

leaf window-size {
    type uint16;
    units "bytes";
    description
        "The size of the receive window, which specifies
        the number of window size units beyond the segment
        identified by the sequence number in the acknowledgment
        field that the sender of this segment is currently
        willing to receive.";
}

leaf urgent-pointer {
    type uint16;
    description
        "This field is an offset from the sequence number
        indicating the last urgent data byte.";
}

leaf options {
    type binary {
        length "1..40";
    }
    description
        "The length of this field is determined by the
        data offset field. Options have up to three
        fields: Option-Kind (1 byte), Option-Length
        (1 byte), Option-Data (variable). The Option-Kind
        field indicates the type of option, and is the
        only field that is not optional. Depending on
        what kind of option we are dealing with,
        the next two fields may be set: the Option-Length
        field indicates the total length of the option,
```



```
        and the Option-Data field contains the value of
        the option, if applicable.";
    }
}

grouping acl-udp-header-fields {
    description
        "Collection of UDP header fields that can be used
        to setup a match filter.";

    leaf length {
        type uint16;
        description
            "A field that specifies the length in bytes of
            the UDP header and UDP data. The minimum
            length is 8 bytes because that is the length of
            the header. The field size sets a theoretical
            limit of 65,535 bytes (8 byte header + 65,527
            bytes of data) for a UDP datagram. However the
            actual limit for the data length, which is
            imposed by the underlying IPv4 protocol, is
            65,507 bytes (65,535 minus 8 byte UDP header
            minus 20 byte IP header).

            In IPv6 jumbograms it is possible to have
            UDP packets of size greater than 65,535 bytes.
            RFC 2675 specifies that the length field is set
            to zero if the length of the UDP header plus
            UDP data is greater than 65,535.";
    }
}

grouping acl-icmp-header-fields {
    description
        "Collection of ICMP header fields that can be
        used to setup a match filter.";

    leaf type {
        type uint8;
        description
            "Also known as Control messages.";
        reference
            "RFC 792: Internet Control Message Protocol (ICMP),
            RFC 4443: Internet Control Message Protocol (ICMPv6)
            for Internet Protocol Version 6 (IPv6)
            Specification.";
    }
}
```

```
leaf code {
  type uint8;
  description
    "ICMP subtype. Also known as Control messages.";
  reference
    "RFC 792: Internet Control Message Protocol (ICMP),
     RFC 4443: Internet Control Message Protocol (ICMPv6)
     for Internet Protocol Version 6 (IPv6)
     Specifciation.";
}

leaf rest-of-header {
  type binary;
  description
    "Unbounded in length, the contents vary based on the
     ICMP type and code. Also referred to as 'Message Body'
     in ICMPv6.";
  reference
    "RFC 792: Internet Control Message Protocol (ICMP),
     RFC 4443: Internet Control Message Protocol (ICMPv6)
     for Internet Protocol Version 6 (IPv6)
     Specifciation.";
}
}
```

<CODE ENDS>

4.3. ACL Examples

Requirement: Deny tcp traffic from 192.0.2.0/24, destined to 198.51.100.0/24.

Here is the acl configuration xml for this Access Control List:

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

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-ipv4-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <ipv4>
              <protocol>6</protocol>
              <destination-ipv4-network>198.51.100.0/24</destination-
-ipv4-network>
              <source-ipv4-network>192.0.2.0/24</source-ipv4-network\
>
            </ipv4>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

The acl and aces can be described in CLI as the following:

```
acl ipv4 sample-ipv4-acl
deny tcp 192.0.2.0/24 198.51.100.0/24
```

Requirement: Accept all DNS traffic destined for 2001:db8::/32 on port 53.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>allow-dns-packets</name>
      <type>ipv6-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <ipv6>
              <destination-ipv6-network>2001:db8::/32</destination-ipv6-network>
            </ipv6>
            <udp>
              <destination-port>
                <operator>eq</operator>
                <port>53</port>
              </destination-port>
            </udp>
          </matches>
          <actions>
            <forwarding>accept</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

4.4. Port Range Usage and Other Examples

When a lower-port and an upper-port are both present, it represents a range between lower-port and upper-port with both the lower-port and upper-port included. When only a port is present, it represents a port, with the operator specifying the range.

The following XML example represents a configuration where TCP traffic from source ports 16384, 16385, 16386, and 16387 is dropped.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-port-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <tcp>
              <source-port>
                <lower-port>16384</lower-port>
                <upper-port>16387</upper-port>
              </source-port>
            </tcp>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

The following XML example represents a configuration where all IPv4 ICMP echo requests are dropped.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-icmp-acl</name>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <ipv4>
              <protocol>1</protocol>
            </ipv4>
            <icmp>
              <type>8</type>
              <code>0</code>
            </icmp>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

The following XML example represents a configuration of a single port, port 21 that accepts TCP traffic.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-ipv4-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <tcp>
              <destination-port>
                <operator>eq</operator>
                <port>21</port>
              </destination-port>
            </tcp>
          </matches>
          <actions>
            <forwarding>accept</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

The following XML example represents a configuration specifying all ports that are not equal to 21, that will drop TCP packets destined for those ports.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-ipv4-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <tcp>
              <destination-port>
                <operator>neq</operator>
                <port>21</port>
              </destination-port>
            </tcp>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

5. Security Considerations

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

The NETCONF Access Control Model (NACM [RFC8341]) provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

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

These are the subtrees and data nodes and their sensitivity/vulnerability:

/acls/acl/aces: This list specifies all the configured access control entries on the device. Unauthorized write access to this list can allow intruders to modify the entries so as to permit traffic that should not be permitted, or deny traffic that should be permitted. The former may result in a DoS attack, or compromise the device. The latter may result in a DoS attack. The impact of an unauthorized read access of the list will allow the attacker to determine which rules are in effect, to better craft an attack.

/acls/acl/aces/ace/actions/logging: This node specifies ability to log packets that match this ace entry. Unauthorized write access to this node can allow intruders to enable logging on one or many ace entries, overwhelming the server in the process. Unauthorized read access of this node can allow intruders to access logging information, which could be used to craft an attack the server.

6. IANA Considerations

This document registers three URIs and three YANG modules.

6.1. URI Registration

This document registers three URIs in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-access-control-list
URI: urn:ietf:params:xml:ns:yang:ietf-packet-fields
URI: urn:ietf:params:xml:ns:yang:ietf-ethertypes

Registrant Contact: The IESG.

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

6.2. YANG Module Name Registration

This document registers three YANG module in the YANG Module Names registry YANG [RFC6020].

```
name: ietf-access-control-list
namespace: urn:ietf:params:xml:ns:yang:ietf-access-control-list
prefix: acl
reference: RFC XXXX

name: ietf-packet-fields
namespace: urn:ietf:params:xml:ns:yang:ietf-packet-fields
prefix: packet-fields
reference: RFC XXXX

name: ietf-ethertypes
namespace: urn:ietf:params:xml:ns:yang:ietf-ethertypes
prefix: ethertypes
reference: RFC XXXX
```

7. Acknowledgements

Alex Clemm, Andy Bierman and Lisa Huang started it by sketching out an initial IETF draft in several past IETF meetings. That draft included an ACL YANG model structure and a rich set of match filters, and acknowledged contributions by Louis Fourie, Dana Blair, Tula Kraiser, Patrick Gili, George Serpa, Martin Bjorklund, Kent Watsen, and Phil Shafer. Many people have reviewed the various earlier drafts that made the draft went into IETF charter.

Dean Bogdanovic, Kiran Agrahara Sreenivasa, Lisa Huang, and Dana Blair each evaluated the YANG model in previous drafts separately, and then worked together to created a ACL draft that was supported by different vendors. That draft removed vendor specific features, and gave examples to allow vendors to extend in their own proprietary ACL. The earlier draft was superseded with this updated draft and received more participation from many vendors.

Authors would like to thank Jason Sterne, Lada Lhotka, Juergen Schoenwalder, David Bannister, Jeff Haas, Kristian Larsson and Einar Nilsen-Nygaard for their review of and suggestions to the draft.

8. References

8.1. Normative References

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<https://www.rfc-editor.org/info/rfc792>>.

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [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>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <<https://www.rfc-editor.org/info/rfc4007>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<https://www.rfc-editor.org/info/rfc5952>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [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>>.
- [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>>.

- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [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>>.

8.2. Informative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <<https://www.rfc-editor.org/info/rfc7011>>.
- [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>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.

[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>>.

Appendix A. Extending ACL model examples

A.1. A company proprietary module example

Module "example-newco-acl" is an example of company proprietary model that augments "ietf-acl" module. It shows how to use 'augment' with an XPath expression to add additional match criteria, actions, and default actions for when no ACE matches are found. All these are company proprietary extensions or system feature extensions. "example-newco-acl" is just an example and it is expected that vendors will create their own proprietary models.

```
module example-newco-acl {  
  
  yang-version 1.1;  
  
  namespace "http://example.com/ns/example-newco-acl";  
  
  prefix example-newco-acl;  
  
  import ietf-access-control-list {  
    prefix "acl";  
  }  
  
  organization  
    "Newco model group.";  
  
  contact  
    "abc@newco.com";  
  description  
    "This YANG module augments IETF ACL Yang.";  
  
  revision 2018-11-06 {  
    description  
      "Creating NewCo proprietary extensions to ietf-acl model";  
  
    reference  
      "RFC XXXX: Network Access Control List (ACL)  
      YANG Data Model";  
  }  
  
  augment "/acl:acls/acl:acl/" +  
    "acl:aces/acl:ace/" +  
    "acl:matches" {  
    
```

```
description "Newco proprietary simple filter matches";
choice protocol-payload-choice {
  description "Newco proprietary payload match condition";
  list protocol-payload {
    key value-keyword;
    ordered-by user;
    description "Match protocol payload";
    uses match-simple-payload-protocol-value;
  }
}

choice metadata {
  description "Newco proprietary interface match condition";
  leaf packet-length {
    type uint16;
    description "Match on packet length";
  }
}
}

augment "/acl:acls/acl:acl/" +
  "acl:aces/acl:ace/" +
  "acl:actions" {
  description "Newco proprietary simple filter actions";
  choice action {
    description "";
    case count {
      description "Count the packet in the named counter";
      leaf count {
        type uint32;
        description "Count";
      }
    }
    case policer {
      description "Name of policer to use to rate-limit traffic";
      leaf policer {
        type string;
        description "Name of the policer";
      }
    }
    case hierarchical-policer {
      leaf hierarchitacl-policer {
        type string;
        description
          "Name of the hierarchical policer.";
      }
    }
  }
  description
    "Name of hierarchical policer to use to
```

```

        rate-limit traffic";
    }
}

augment "/acl:acls/acl:acl" +
    "/acl:aces/acl:ace/" +
    "acl:actions" {
    leaf default-action {
        type identityref {
            base acl:forwarding-action;
        }
        default acl:drop;
        description
            "Actions that occur if no ace is matched.";
    }
    description
        "Newco proprietary default action";
}

grouping match-simple-payload-protocol-value {
    description "Newco proprietary payload";
    leaf value-keyword {
        type enumeration {
            enum icmp {
                description "Internet Control Message Protocol";
            }
            enum icmp6 {
                description
                    "Internet Control Message Protocol
                     Version 6";
            }
            enum range {
                description "Range of values";
            }
        }
        description "(null)";
    }
}
}

```

The following figure is the tree diagram of example-newco-acl. In this example, /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace/ietf-acl:matches are augmented with two new choices, protocol-payload-choice and metadata. The protocol-payload-choice uses a grouping with an enumeration of all supported protocol values. Metadata matches apply to fields associated with the packet but not

in the packet header such as overall packet length. In another example, `/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace/ietf-acl:actions` are augmented with a new choice of actions.

```
module: example-newco-acl
  augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
    +--rw (protocol-payload-choice)?
    |   +--:(protocol-payload)
    |   |   +--rw protocol-payload* [value-keyword]
    |   |   +--rw value-keyword      enumeration
    |   +--rw (metadata)?
    |   |   +--:(packet-length)
    |   |   |   +--rw packet-length?      uint16
    |   +--rw (action)?
    |   |   +--:(count)
    |   |   |   +--rw count?                uint32
    |   |   +--:(policer)
    |   |   |   +--rw policer?              string
    |   |   +--:(hierarchical-policer)
    |   |   |   +--rw hierarchitacl-policer? string
    |   +--rw (default-action)?
    |   |   +--rw default-action?          identityref
```

A.2. Linux nftables

As Linux platform is becoming more popular as networking platform, the Linux data model is changing. Previously ACLs in Linux were highly protocol specific and different utilities were used (iptables, ip6tables, arptables, ebtables), so each one had separate data model. Recently, this has changed and a single utility, nftables, has been developed. With a single application, it has a single data model for firewall filters and it follows very similarly to the ietf-access-control list module proposed in this draft. The nftables support input and output ACEs and each ACE can be defined with match and action.

The example in Section 4.3 can be configured using nftable tool as below.

```
nft add table ip filter
nft add chain filter input
nft add rule ip filter input ip protocol tcp ip saddr \
  192.0.2.1/24 drop
```

The configuration entries added in nftable would be.


```
table ip filter {
  chain input {
    ip protocol tcp ip saddr 192.0.2.1/24 drop
  }
}
```

We can see that there are many similarities between Linux nftables and IETF ACL YANG data models and its extension models. It should be fairly easy to do translation between ACL YANG model described in this draft and Linux nftables.

A.3. Ethertypes

The ACL module is dependent on the definition of ethertypes. IEEE owns the allocation of those ethertypes. This model is being included here to enable definition of those types till such time that IEEE takes up the task of publication of the model that defines those ethertypes. At that time, this model can be deprecated.

```
<CODE BEGINS> file "ietf-ethertypes@2018-11-06.yang"

module iETF-ethertypes {
  namespace "urn:ietf:params:xml:ns:yang:ietf-ethertypes";
  prefix ethertypes;

  organization
    "IETF NETMOD (NETCONF Data Modeling Language)";

  contact
    "WG Web: <http://tools.ietf.org/wg/netmod/>
    WG List: <mailto:netmod@ietf.org>

    Editor: Mahesh Jethanandani
            <mjethanandani@gmail.com>";

  description
    "This module contains the common definitions for the
    Ethertype used by different modules. It is a
    placeholder module, till such time that IEEE
    starts a project to define these Ethertypes
    and publishes a standard.

    At that time this module can be deprecated.";

  revision 2018-11-06 {
    description
      "Initial revision.";
```

```
    reference
      "RFC XXXX: IETF Ethertype YANG Data Module.";
  }

typedef ethertype {
  type union {
    type uint16;
    type enumeration {
      enum ipv4 {
        value 2048;
        description
          "Internet Protocol version 4 (IPv4) with a
           hex value of 0x0800.";
        reference
          "RFC 791: Internet Protocol.";
      }
      enum arp {
        value 2054;
        description
          "Address Resolution Protocol (ARP) with a
           hex value of 0x0806.";
        reference
          "RFC 826: An Ethernet Address Resolution Protocol.";
      }
      enum wlan {
        value 2114;
        description
          "Wake-on-LAN. Hex value of 0x0842.";
      }
      enum trill {
        value 8947;
        description
          "Transparent Interconnection of Lots of Links.
           Hex value of 0x22F3.";
        reference
          "RFC 6325: Routing Bridges (RBridges): Base Protocol
           Specification.";
      }
      enum srp {
        value 8938;
        description
          "Stream Reservation Protocol. Hex value of
           0x22EA.";
        reference
          "IEEE 801.1Q-2011.";
      }
      enum decnet {
        value 24579;
      }
    }
  }
}
```

```
        description
            "DECnet Phase IV. Hex value of 0x6003.";
    }
    enum rarp {
        value 32821;
        description
            "Reverse Address Resolution Protocol.
             Hex value 0x8035.";
        reference
            "RFC 903. A Reverse Address Resolution Protocol.";
    }
    enum appletalk {
        value 32923;
        description
            "Appletalk (Ethertalk). Hex value 0x809B.";
    }
    enum aarp {
        value 33011;
        description
            "Appletalk Address Resolution Protocol. Hex value
             of 0x80F3.";
    }
    enum vlan {
        value 33024;
        description
            "VLAN-tagged frame (802.1Q) and Shortest Path
             Bridging IEEE 802.1aq with NNI compatibility.
             Hex value 0x8100.";
        reference
            "802.1Q.";
    }
    enum ipx {
        value 33079;
        description
            "Internetwork Packet Exchange (IPX). Hex value
             of 0x8137.";
    }
    enum qnx {
        value 33284;
        description
            "QNX Qnet. Hex value of 0x8204.";
    }
    enum ipv6 {
        value 34525;
        description
            "Internet Protocol Version 6 (IPv6). Hex value
             of 0x86DD.";
        reference
```

```
        "RFC 8200: Internet Protocol, Version 6 (IPv6)
          Specification
          RFC 8201: Path MTU Discovery for IPv6.";
    }
    enum efc {
        value 34824;
        description
            "Ethernet flow control using pause frames.
             Hex value of 0x8808";
        reference
            "IEEE Std. 802.1Qbb.";
    }
    enum esp {
        value 34825;
        description
            "Ethernet Slow Protocol. Hex value of 0x8809.";
        reference
            "IEEE Std. 802.3-2015";
    }
    enum cobranet {
        value 34841;
        description
            "CobraNet. Hex value of 0x8819";
    }
    enum mpls-unicast {
        value 34887;
        description
            "MultiProtocol Label Switch (MPLS) unicast traffic.
             Hex value of 0x8847.";
        reference
            "RFC 3031: Multiprotocol Label Switching Architecture.";
    }
    enum mpls-multicast {
        value 34888;
        description
            "MultiProtocol Label Switch (MPLS) multicast traffic.
             Hex value of 0x8848.";
        reference
            "RFC 3031: Multiprotocol Label Switching Architecture.";
    }
    enum pppoe-discovery {
        value 34915;
        description
            "Point-to-Point Protocol over Ethernet. Used during
             the discovery process. Hex value of 0x8863.";
        reference
            "RFC 2516: A method for Transmitting PPP over Ethernet
             PPoE.";
```

```
}
enum pppoe-session {
  value 34916;
  description
    "Point-to-Point Protocol over Ethernet. Used during
    session stage. Hex value of 0x8864.";
  reference
    "RFC 2516: A method for Transmitting PPP over Ethernet
    PPPoE.";
}
enum intel-ans {
  value 34925;
  description
    "Intel Advanced Networking Services. Hex value of
    0x886D.";
}
enum jumbo-frames {
  value 34928;
  description
    "Jumbo frames or Ethernet frames with more than
    1500 bytes of payload, upto 9000 bytes.";
}
enum homeplug {
  value 34939;
  description
    "Family name for the various power line
    communications. Hex value of 0x887B.";
}
enum eap {
  value 34958;
  description
    "Ethernet Access Protocol (EAP) over LAN. Hex value
    of 0x888E.";
  reference
    "IEEE 802.1X";
}
enum profinet {
  value 34962;
  description
    "PROcess FIeld Net (PROFINET). Hex value of 0x8892.";
}
enum hyperscsi {
  value 34970;
  description
    "SCSI over Ethernet. Hex value of 0x889A";
}
enum aoe {
  value 34978;
```

```
        description
            "Advanced Technology Advancement (ATA) over Ethernet.
             Hex value of 0x88A2.";
    }
    enum ethercat {
        value 34980;
        description
            "Ethernet for Control Automation Technology (EtherCAT).
             Hex value of 0x88A4.";
    }
    enum provider-bridging {
        value 34984;
        description
            "Provider Bridging (802.1ad) and Shortest Path Bridging
             (801.1aq). Hex value of 0x88A8.";
        reference
            "IEEE 802.1ad, IEEE 802.1aq).";
    }
    enum ethernet-powerlink {
        value 34987;
        description
            "Ethernet Powerlink. Hex value of 0x88AB.";
    }
    enum goose {
        value 35000;
        description
            "Generic Object Oriented Substation Event (GOOSE).
             Hex value of 0x88B8.";
        reference
            "IEC/ISO 8802-2 and 8802-3.";
    }
    enum gse {
        value 35001;
        description
            "Generic Substation Events. Hex value of 88B9.";
        reference
            "IEC 61850.";
    }
    enum sv {
        value 35002;
        description
            "Sampled Value Transmission. Hex value of 0x88BA.";
        reference
            "IEC 61850.";
    }
    enum lldp {
        value 35020;
        description
```

```
        "Link Layer Discovery Protocol (LLDP). Hex value of
          0x88CC.";
      reference
        "IEEE 802.1AB.";
    }
    enum sercos {
      value 35021;
      description
        "Sercos Interface. Hex value of 0x88CD.";
    }
    enum wsmpp {
      value 35036;
      description
        "WAVE Short Message Protocol (WSMP). Hex value of
          0x88DC.";
    }
    enum homeplug-av-mme {
      value 35041;
      description
        "HomePlug AV MME. Hex value of 88E1.";
    }
    enum mrp {
      value 35043;
      description
        "Media Redundancy Protocol (MRP). Hex value of
          0x88E3.";
      reference
        "IEC62439-2.";
    }
    enum macsec {
      value 35045;
      description
        "MAC Security. Hex value of 0x88E5.";
      reference
        "IEEE 802.1AE.";
    }
    enum pbb {
      value 35047;
      description
        "Provider Backbone Bridges (PBB). Hex value of
          0x88E7.";
      reference
        "IEEE 802.1ah.";
    }
    enum cfm {
      value 35074;
      description
        "Connectivity Fault Management (CFM). Hex value of
```

```
        0x8902.";
    reference
        "IEEE 802.1ag.";
}
enum fcoe {
    value 35078;
    description
        "Fiber Channel over Ethernet (FCoE). Hex value of
        0x8906.";
    reference
        "T11 FC-BB-5.";
}
enum fcoe-ip {
    value 35092;
    description
        "FCoE Initialization Protocol. Hex value of 0x8914.";
}
enum roce {
    value 35093;
    description
        "RDMA over Converged Ethernet (RoCE). Hex value of
        0x8915.";
}
enum tte {
    value 35101;
    description
        "TTEthernet Protocol Control Frame (TTE). Hex value
        of 0x891D.";
    reference
        "SAE AS6802.";
}
enum hsr {
    value 35119;
    description
        "High-availability Seamless Redundancy (HSR). Hex
        value of 0x892F.";
    reference
        "IEC 62439-3:2016.";
}
}
}
description
    "The uint16 type placeholder is defined to enable
    users to manage their own ethertypes not
    covered by the module. Otherwise the module contains
    enum definitions for the more commonly used ethertypes.";
}
```


<CODE ENDS>

Authors' Addresses

Mahesh Jethanandani
VMware

Email: mjethanandani@gmail.com

Sonal Agarwal
Cisco Systems, Inc.

Email: sagarwal12@gmail.com

Lisa Huang

Email: huangyi_99@yahoo.com

Dana Blair

Email: dana@blairhome.com

Network Working Group
Internet-Draft
Obsoletes: 6087 (if approved)
Intended status: BCP
Expires: September 14, 2018

A. Bierman
YumaWorks
March 13, 2018

Guidelines for Authors and Reviewers of YANG Data Model Documents
draft-ietf-netmod-rfc6087bis-20

Abstract

This memo provides guidelines for authors and reviewers of specifications containing YANG data model 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 data model modules. This document obsoletes RFC 6087.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 14, 2018.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	5
1.1. Changes Since RFC 6087	5
2. Terminology	8
2.1. Requirements Notation	8
2.2. NETCONF Terms	8
2.3. YANG Terms	8
2.4. NMDA Terms	9
2.5. Terms	9
3. General Documentation Guidelines	10
3.1. Module Copyright	10
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
3.6. Definitions Section	13
3.7. Security Considerations Section	13
3.7.1. Security Considerations Section Template	14
3.8. IANA Considerations Section	15
3.8.1. Documents that Create a New Namespace	15
3.8.2. Documents that Extend an Existing Namespace	16
3.9. Reference Sections	16
3.10. Validation Tools	16
3.11. Module Extraction Tools	17
3.12. Module Usage Examples	17
4. YANG Usage Guidelines	18
4.1. Module Naming Conventions	18
4.2. Prefixes	19
4.3. Identifiers	20
4.3.1. Identifier Naming Conventions	20
4.4. Defaults	21
4.5. Conditional Statements	21
4.6. XPath Usage	22
4.6.1. XPath Evaluation Contexts	22
4.6.2. Function Library	23
4.6.3. Axes	24
4.6.4. Types	24
4.6.5. Wildcards	25
4.6.6. Boolean Expressions	26
4.7. YANG Definition Lifecycle Management	27
4.8. Module Header, Meta, and Revision Statements	28
4.9. Namespace Assignments	29

4.10. Top-Level Data Definitions	31
4.11. Data Types	31
4.11.1. Fixed Value Extensibility	32
4.11.2. Patterns and Ranges	32
4.11.3. Enumerations and Bits	33
4.11.4. Union Types	34
4.11.5. Empty and Boolean	35
4.12. Reusable Type Definitions	36
4.13. Reusable Groupings	37
4.14. Data Definitions	37
4.14.1. Non-Presence Containers	39
4.14.2. Top-Level Data Nodes	40
4.15. Operation Definitions	40
4.16. Notification Definitions	40
4.17. Feature Definitions	41
4.18. YANG Data Node Constraints	42
4.18.1. Controlling Quantity	42
4.18.2. must vs. when	42
4.19. Augment Statements	42
4.19.1. Conditional Augment Statements	43
4.19.2. Conditionally Mandatory Data Definition Statements	43
4.20. Deviation Statements	44
4.21. Extension Statements	45
4.22. Data Correlation	46
4.22.1. Use of Leafref for Key Correlation	47
4.23. Operational State	48
4.23.1. Combining Operational State and Configuration Data	48
4.23.2. Representing Operational Values of Configuration Data	49
4.23.3. NMDA Transition Guidelines	49
4.24. Performance Considerations	53
4.25. Open Systems Considerations	54
4.26. Guidelines for YANG 1.1 Specific Constructs	54
4.26.1. Importing Multiple Revisions	54
4.26.2. Using Feature Logic	54
4.26.3. anyxml vs. anydata	55
4.26.4. action vs. rpc	55
4.27. Updating YANG Modules (Published vs. Unpublished)	56
5. IANA Considerations	57
6. Security Considerations	58
7. Acknowledgments	59
8. References	60
8.1. Normative References	60
8.2. Informative References	61
Appendix A. Change Log	63
A.1. v19 to v20	63
A.2. v18 to v19	63
A.3. v17 to v18	63

A.4.	v16 to v17	63
A.5.	v15 to v16	63
A.6.	v15 to v16	63
A.7.	v14 to v15	63
A.8.	v13 to v14	63
A.9.	v12 to v13	64
A.10.	v11 to v12	64
A.11.	v10 to v11	64
A.12.	v09 to v10	64
A.13.	v08 to v09	64
A.14.	v07 to v08	65
A.15.	v06 to v07	65
A.16.	v05 to v06	65
A.17.	v04 to v05	66
A.18.	v03 ot v04	66
A.19.	v02 to v03	66
A.20.	v01 to v02	67
A.21.	v00 to v01	67
Appendix B.	Module Review Checklist	68
Appendix C.	YANG Module Template	70
Author's Address	72

1. Introduction

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

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

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

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

This document defines usage guidelines related to the NETCONF operations layer and NETCONF content layer, as defined in [RFC6241], and the RESTCONF methods and RESTCONF resources, as defined in [RFC8040],

These guidelines are intended to be used by authors and reviewers to improve the readability and interoperability of published YANG data models.

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

1.1. Changes Since RFC 6087

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

- o Updated NETCONF reference from RFC 4741 to RFC 6241

- o Updated NETCONF over SSH citation from RFC 4742 to RFC 6242
- o Updated YANG Types reference from RFC 6021 to RFC 6991
- o Updated obsolete URLs for IETF resources
- o Changed top-level data node guideline
- o Clarified XPath usage for a literal value representing a YANG identity
- o Clarified XPath usage for a when-stmt
- o Clarified XPath usage for 'proceeding-sibling' and 'following-sibling' axes
- o Added terminology guidelines
- o Added YANG tree diagram guidelines
- o Updated XPath guidelines for type conversions and function library usage.
- o Updated data types section
- o Updated notifications section
- o Clarified conditional key leaf nodes
- o Clarify usage of 'uint64' and 'int64' data types
- o Added text on YANG feature usage
- o Added Identifier Naming Conventions
- o Clarified use of mandatory nodes with conditional augmentations
- o Clarified namespace and domain conventions for example modules
- o Clarified conventions for identifying code components
- o Added YANG 1.1 guidelines
- o Added Data Model Constraints section
- o Added mention of RESTCONF protocol

- o Added guidelines for NMDA Revised Datastores

2. Terminology

2.1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. NETCONF Terms

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

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

2.3. YANG Terms

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

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

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

2.4. NMDA Terms

The following terms are defined in the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores], and are not redefined here:

- o configuration
- o conventional configuration datastore
- o datastore
- o operational state
- o operational state datastore

2.5. Terms

The following terms are used throughout this document:

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

3. General Documentation Guidelines

YANG data model modules under review are likely to be contained in Internet-Drafts. All guidelines for Internet-Draft authors [ID-Guidelines] MUST be followed. The RFC Editor provides guidelines for authors of RFCs, which are first published as Internet-Drafts. These guidelines should be followed and are defined in [RFC7322] and updated in [RFC7841], "RFC Document Style" [RFC-STYLE], and [I-D.flanagan-7322bis].

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

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

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

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

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

3.1. Module Copyright

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

<https://trustee.ietf.org/license-info/>

3.2. Code Components

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

The "<CODE BEGINS>" tag SHOULD be followed by a string identifying the file name specified in Section 5.2 of [RFC7950]. The name string form that includes the revision-date SHOULD be used. The revision date MUST match the date used in the most recent revision of the module.

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

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

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

<CODE ENDS>
```

3.2.1. Example Modules

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

An example module SHOULD be named using the term "example", followed by a hyphen, followed by a descriptive name, e.g., "example-toaster". See Section 4.9 regarding the namespace guidelines for example modules.

3.3. Terminology Section

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

3.4. Tree Diagrams

YANG tree diagrams provide a concise representation of a YANG module, and SHOULD be included to help readers understand YANG module structure. Guidelines on tree diagrams can be found in Section 3 of [I-D.ietf-netmod-yang-tree-diagrams].

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 [I-D.ietf-netmod-rfc8022bis] for an example of such a reference.

3.5. Narrative Sections

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

If the module(s) defined by the specification imports definitions from other modules (except for those defined in the [RFC7950] or [RFC6991] documents), or are always implemented in conjunction with other modules, then those facts MUST be noted in the overview section, as MUST be noted any special interpretations of definitions in other modules. Refer to section 2.3 of [I-D.ietf-netmod-rfc8022bis] for an example of this overview section.

If the documents contains YANG module(s) that are compliant with the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores], then the Introduction section should mention this fact.

Example:

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [I-D.ietf-netmod-revised-datastores].

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

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

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

3.6. Definitions Section

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

A YIN syntax version of the module MAY also be present in the document. There MAY also be other types of modules present in the document, such as 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] apply to YANG description statements in normative modules exactly as they would in any other normative section.

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

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

See Section 4 for guidelines on YANG usage.

3.7. Security Considerations Section

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

This section MUST be patterned after the latest approved template (available at <https://trac.ietf.org/trac/ops/wiki/yang-security-guidelines>). Section 3.7.1 contains the security considerations template dated 2013-05-08 and last updated 2017-12-21. Authors MUST check the WEB page at the URL listed above in case there is a more recent version available.

In particular:

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

3.7.1. Security Considerations Section Template

X. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

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

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

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

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

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

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

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

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

<list RPC operations and state why they are sensitive>

3.8. IANA Considerations Section

In order to comply with IESG policy as set forth in <https://www.ietf.org/id-info/checklist.html>, every Internet-Draft that is submitted to the IESG for publication MUST contain an IANA Considerations section. The requirements for this section vary depending on what actions are required of the IANA. If there are no IANA considerations applicable to the document, then the IANA Considerations section stating that there are no actions might be removed by the RFC Editor before publication. Refer to the guidelines in [RFC8126] for more details.

Each normative YANG module MUST be registered in the XML namespace Registry [RFC3688], and the YANG Module Names Registry [RFC6020]. This applies to new modules and updated modules. Examples of these registrations for the "ietf-template" module can be found in Section 5.

3.8.1. Documents that Create a New Namespace

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

Considerations section that specifies how the namespace is to be administered.

Specifically, if any YANG module namespace statement value contained in the document is not already registered with IANA, then a new YANG Namespace registry entry MUST be requested from the IANA. The [RFC7950] specification includes the procedure for this purpose in its IANA Considerations section.

3.8.2. Documents that Extend an Existing Namespace

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

3.9. Reference 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 Internet Draft. The 'pyang' YANG compiler is freely available from github:

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

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

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

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

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

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

3.11. Module Extraction Tools

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

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

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

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

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

3.12. Module Usage Examples

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

4. YANG Usage Guidelines

Modules in IETF Standards Track specifications MUST comply with all syntactic and semantic requirements of YANG 1.1 [RFC7950]. See the exception for YANG 1.0 in Section 3.6. The guidelines in this section are intended to supplement the YANG specification, which is intended to define a minimum set of conformance requirements.

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

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

4.1. Module Naming Conventions

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

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

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

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

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

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

4.2. Prefixes

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

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

  container foo;
}

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

  typedef foo { type uint32; }
}

module example-one {
  namespace "tag:example.com,2017:example-one";
  prefix one;
  import example-foo { prefix f; }
  import example-bar { prefix b; }

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

YANG defines the following rules for prefix usage:

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

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

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

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

4.3. Identifiers

Identifiers for all YANG identifiers in published modules MUST be between 1 and 64 characters in length. These include any construct specified as an 'identifier-arg-str' token in the ABNF in Section 13 of [RFC7950].

4.3.1. Identifier Naming Conventions

Identifiers SHOULD follow a consistent naming pattern throughout the module. Only lower-case letters, numbers, and dashes SHOULD be used in identifier names. Upper-case characters, 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 the set of sibling nodes defined in the same module namespace.

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

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

4.4. Defaults

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

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

Statement Defaults

4.5. Conditional Statements

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

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

If a data definition is optional, depending on server support for a NETCONF or RESTCONF protocol capability, then a YANG 'feature' statement SHOULD be defined. The defined "feature" statement SHOULD then be used in the conditional "if-feature" statement referencing the optional data definition.

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

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

key leaf that do not also apply to the parent list node.

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

4.6. XPath Usage

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

4.6.1. XPath Evaluation Contexts

YANG defines 5 separate contexts for evaluation of XPath statements:

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

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

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

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

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

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

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

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

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

4.6.2. Function Library

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

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

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

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

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

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

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

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

Example:

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

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

4.6.3. Axes

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

The 'preceding', and 'following' axes SHOULD NOT be used. These constructs rely on XML document order within a NETCONF or RESTCONF server configuration database, which may not be supported consistently or produce reliable results across implementations. Predicate expressions based on static node properties (e.g., element name or value, 'ancestor' or 'descendant' axes) SHOULD be used instead. The 'preceding' and 'following' axes MAY be used if document order is not relevant to the outcome of the expression (e.g., check for global uniqueness of a parameter value).

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

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

4.6.4. Types

Data nodes that use the 'int64' and 'uint64' built-in type SHOULD NOT be used within numeric or boolean expressions. There are boundary

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

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

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

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

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

Example:

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

4.6.5. Wildcards

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

Wildcard expansion is done within a server against all the nodes from all namespaces, so it is possible for a 'must' or 'when' expression that uses the '*' operator will always evaluate to false if processed

within a single YANG module. In such cases, the 'description' statement SHOULD clarify that augmenting objects are expected to match the wildcard expansion.

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

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 fool {  
  type enumeration {  
    enum one;  
    enum two;  
    enum three;  
    enum four;  
  }  
}
```

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

4.7. YANG Definition Lifecycle Management

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

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

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

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

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

If an import statement is for a module from a stable source (e.g., an RFC for an IETF module), then a reference-stmt **SHOULD** be present within an import statement.

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

If submodules are used, then the document containing the main module **MUST** be updated so that the main module revision date is equal 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 chartered to write the document. For other standards organizations, a similar approach is also suggested.

The contact statement MUST be present. If the module is contained in a document intended for Standards Track status, then the working group 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 Working Group chairs.

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

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

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

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

```
revision "2012-02-22" {  
  description  
    "Initial version";  
  reference  
    "RFC 6536: Network Configuration Protocol (NETCONF)  
      Access Control Model";  
}
```

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

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

```
revision "2017-12-11" {  
  description  
    "Added support for YANG 1.1 actions and notifications tied to  
    data nodes. Clarify how NACM extensions can be used by other  
    data models.";  
  reference  
    "RFC XXXX: Network Configuration Protocol (NETCONF)  
      Access Control Model";  
}  
  
revision "2012-02-22" {  
  description  
    "Initial version";  
  reference  
    "RFC 6536: Network Configuration Protocol (NETCONF)  
      Access Control Model";  
}
```

4.9. Namespace Assignments

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

- o the module to compile correctly instead of generating disruptive fatal errors.

- o early implementors to use the modules without picking a random value for the XML namespace.
- o early interoperability testing since independent implementations will use the same XML namespace value.

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

A standard namespace statement value SHOULD have the following form:

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

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

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

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

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

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

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

Note that a different URN prefix string SHOULD be used for non-Standards-Track modules. The string SHOULD be selected according to the guidelines in [RFC7950].

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

Example URIs using URLs per RFC 3986 [RFC3986]:

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

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

Example URIs using tags per RFC 4151 [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 the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores] 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, and therefore few requirements can be specified on that subject.

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

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

4.11.1. Fixed Value Extensibility

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

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

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

```
identity foo-type {  
  description "Base for the extensible type";  
}  
  
identity one {  
  base f:foo-type;  
}  
identity two {  
  base f:foo-type;  
}  
  
leaf foo {  
  type identityref {  
    base f:foo-type;  
  }  
}
```

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

4.11.2. Patterns and Ranges

For string data types, if a machine-readable pattern can be defined for the desired semantics, then one or more pattern statements SHOULD be present. A single quoted string SHOULD be used to specify the pattern, since a double-quoted string can modify the content. If the patterns used in a type definition have known limitations such as false negative or false positive matches, then these limitations SHOULD be documented within the typedef or data definition.

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

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

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

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

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

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

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

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

4.11.3. Enumerations and Bits

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

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

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

4.11.4. Union Types

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

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

Incorrect:

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

Correct:

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

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

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

Incorrect:

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

Correct:

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

4.11.5. Empty and Boolean

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

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

The advantage of using type "empty" instead of type "boolean" is that the default (not present) does not take up any bytes in a representation. The disadvantage is that the client may not be sure if an empty leaf is missing because it was filtered somehow or not

implemented. The client may not have a complete and accurate schema for the data returned by the server, and 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 [RFC6991], then it SHOULD be used instead of defining a new derived type.

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

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

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

The description statement MUST be present.

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

4.13. Reusable Groupings

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

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

- o Clearly identify the purpose of the grouping in the "description" statement.
- o There are 5 different XPath contexts in YANG (rpc/input, rpc/output, notification, config=true data nodes, and all data nodes). Clearly identify which XPath contexts are applicable or excluded for the grouping.
- o Do not reference data outside the grouping in any "path", "must", or "when" statements.
- o Do not include a "default" sub-statement on a leaf or choice unless the value applies on all possible contexts.
- o Do not include a "config" sub-statement on a data node unless the value applies on all possible contexts.
- o Clearly identify any external dependencies in the grouping "description" statement, such as nodes referenced by absolute path from a "path", "must", or "when" statement.

4.14. Data Definitions

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

- o anyxml
- o augment
- o choice

- o container
- o extension
- o feature
- o grouping
- o identity
- o leaf
- o leaf-list
- o list
- o notification
- o rpc
- o typedef

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

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

It has been found that the 'anyxml' statement is not implemented consistently across all servers. It is possible that mixed mode XML will not be supported, or 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, 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., "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 sub-directories in a file system. Although these containers do not have any semantics, they can impact protocol operations for the descendant data nodes within a non-presence container, so use of these containers SHOULD be considered carefully.

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

4.14.2. Top-Level Data Nodes

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

- o top-level siblings are not ordered
- o top-level siblings are not static, and depends on the modules that are loaded
- o for sub-tree filtering, retrieval of a top-level leaf-list will be treated as a content-match node for all top-level-siblings
- o a top-level list with many instances may impact performance

4.15. Operation Definitions

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

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

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

4.16. Notification Definitions

The description statement **MUST** be present.

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

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

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

Note that there are no formal YANG statements to identify any data node resources associated with a notification. The description statement for the notification SHOULD specify if and how the notification identifies any data node resources associated with the specific event.

4.17. Feature Definitions

The YANG "feature" statement is used to define a label for a set of optional functionality within a module. The "if-feature" statement is used in the YANG statements associated with a feature. The description-stmt within a feature-stmt MUST specify any interactions with other features.

The set of YANG features defined in a module should be considered carefully. Very fine granular features increase interoperability complexity and should be avoided. A likely misuse of the feature mechanism is the tagging of individual leafs (e.g., counters) with separate features.

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

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

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

For example, feature2 requires implementation of feature1:

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

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

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 vs. when

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

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

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

4.19. Augment Statements

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

A top-level "augment" statement SHOULD NOT be used if the target data

node is in the same module or submodule as the evaluated "augment" statement. The data definition statements SHOULD be added inline instead.

4.19.1. Conditional Augment Statements

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

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

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

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

4.19.2. Conditionally Mandatory Data Definition Statements

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

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

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

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

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

this external data.

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

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

  import iana-if-type { prefix iana; }
  import ietf-interfaces { prefix if; }

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

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

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

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

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

4.20. Deviation Statements

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

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

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

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

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

Example documenting platform resource limits:

Wrong: (max-elements in the list itself)

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

Correct: (max-elements in a deviation)

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

4.21. Extension Statements

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

The following guidelines apply to the usage of YANG extensions:

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

4.22. Data Correlation

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

- o inline: update the module with new protocol-accessible objects. The naming and data organization of the original objects is used. The new objects are in the original module namespace.
- o augment: create a new module with new protocol-accessible objects that augment the original data structure. The naming and data organization of the original objects is used. The new objects are in the new module namespace.
- o mirror: create new objects in a new module or the original module, except use new a naming scheme and data location. The naming can be coupled in different ways. Tight coupling is achieved with a "leafref" data type, with the "require-instance" sub-statement set to "true". This method SHOULD be used.

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

The relationship between configuration and operational state has been

clarified in NMDA [I-D.ietf-netmod-revised-datastores].

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 YANG XPath definition much more complicated.

Operational state is now modeled using YANG according to the new NMDA [I-D.ietf-netmod-revised-datastores], and is now 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 leaves 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-state" and "oper-state" leaves in [RFC7223]. 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 leaves 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 leaves. 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 assuming they will be used on servers supporting the operational state datastore. With this in mind, YANG modules SHOULD define config "false" 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 values may differ, in which

case a distinct config "false" node SHOULD be defined to hold the operational value for the configured node.

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

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

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

(a) Modules that require immediate support for the NMDA features SHOULD be structured for NMDA. A temporary non-NMDA version of this type of module MAY exist, either an existing model or a model created either by hand or with suitable tools that mirror the current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative appendix. The use of the non-NMDA module will allow temporary bridging of the time period until NMDA implementations are available.

(b) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] has been restructured as an NMDA-compatible model in [I-D.ietf-netmod-rfc7223bis]. The "/interfaces-state" hierarchy has been marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.

(c) For models that augment models which have not been structured with the NMDA, the modeler will have to consider the structure of the base model and the guidelines listed above. Where possible, such models should move to new revisions of the base model that are NMDA-compatible. When that is not possible, augmenting "state" containers SHOULD be avoided, with the expectation that the base model will be re-released with the state containers marked as deprecated. It is RECOMMENDED to augment only the "/foo" hierarchy of the base model. Where this recommendation cannot be followed, then any new "state" elements SHOULD be included in their own module.

4.23.3.1. Temporary non-NMDA Modules

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

A server that needs to support both NMDA and non-NMDA clients can advertise both the new NMDA module and the temporary non-NMDA module. A non-NMDA client can use separate "foo" and "foo-state" subtrees, except the "foo-state" subtree is located in a different (temporary) module. The NMDA module can be used by a non-NMDA client to access the conventional configuration datastores, and the deprecated <get> operation to access nested config=false data nodes.

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

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

4.23.3.2. Example: Create a New NMDA Module

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

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

4.23.3.3. Example: Convert an old Non-NMDA Module

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

Old Module:

```
module example-foo {  
  namespace "urn:example.com:params:xml:ns:yang:example-foo";  
  prefix "foo";  
  
  container foo {  
    // configuration data child nodes  
  }  
  
  container foo-state {  
    config false;  
    // operational state child nodes  
  }  
}
```

Converted NMDA Module:

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

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

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

4.23.3.4. Example: Create a Temporary NMDA Module:

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

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

  // import new or converted module; not used in this example
  import example-foo { prefix foo; }

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

4.24. Performance Considerations

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

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

4.25. Open Systems Considerations

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

4.26. Guidelines for YANG 1.1 Specific Constructs

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

4.26.1. Importing Multiple Revisions

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

4.26.2. Using Feature Logic

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

YANG features SHOULD be used instead of the "when" statement, if possible. Features are advertised by the server and objects conditional by if-feature 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 module is loaded and not changed

at runtime). However every client edit might cause a "when" statement result to change.

4.26.3. anyxml vs. 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 vs. 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) [I-D.ietf-netconf-rfc6536bis] 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 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 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 vs. 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 which is considered to be stable, and unpublished work which is considered to be unstable. For example, in the IETF, the RFC document is used for published work, and the Internet-Draft is used for unpublished work.

5. IANA Considerations

- RFC Ed: These registries need to be updated to reference this RFC instead of RFC 6087 for the ietf-template module, and remove this note.

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

The following registration has been made in [RFC6087] and updated by this document.

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

Registrant Contact: The IESG.

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

The following assignment has been made in [RFC6087] and updated by this document in the YANG Module Names Registry, or the YANG module template in Appendix C.

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

YANG Registry Assignment

6. Security Considerations

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

7. Acknowledgments

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

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

8. References

8.1. Normative References

- [I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", draft-ietf-netmod-revised-datastores-10
(work in progress), January 2018.
- [ID-Guidelines]
Housley, R., Ed., "Guidelines to Authors of Internet-
Drafts", December 2010,
<<https://www.ietf.org/standards/ids/guidelines/>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
January 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, January 2005.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/
RFC5246, August 2008,
<<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide
to the IETF Trust", BCP 78, RFC 5378, November 2008.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the
Network Configuration Protocol (NETCONF)", RFC 6020,
October 2010.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language",
RFC 7950, DOI 10.17487/RFC7950, August 2016,
<<http://www.rfc-editor.org/info/rfc7950>>.
- [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>>.
- [W3C.REC-xpath-19991116]
Clark, J. and S. DeRose, "XML Path Language (XPath)
Version 1.0", World Wide Web Consortium

Recommendation REC-xpath-19991116, November 1999,
<<http://www.w3.org/TR/1999/REC-xpath-19991116>>.

8.2. Informative References

- [I-D.flanagan-7322bis]
Flanagan, H. and R. Editor, "RFC Style Guide",
draft-flanagan-7322bis-02 (work in progress),
September 2017.
- [I-D.ietf-netconf-rfc6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration
Access Control Module", draft-ietf-netconf-rfc6536bis-09
(work in progress), December 2017.
- [I-D.ietf-netmod-rfc7223bis]
Bjorklund, M., "A YANG Data Model for Interface
Management", draft-ietf-netmod-rfc7223bis-03 (work in
progress), January 2018.
- [I-D.ietf-netmod-rfc8022bis]
Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for
Routing Management (NMDA Version)",
draft-ietf-netmod-rfc8022bis-11 (work in progress),
January 2018.
- [I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams",
draft-ietf-netmod-yang-tree-diagrams-06 (work in
progress), February 2018.
- [RFC-STYLE]
Braden, R., Ginoza, S., and A. Hagens, "RFC Document
Style", September 2009,
<<http://www.rfc-editor.org/styleguide/>>.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision
3", BCP 9, RFC 2026, DOI 10.17487/RFC2026, October 1996,
<<http://www.rfc-editor.org/info/rfc2026>>.
- [RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme",
RFC 4151, DOI 10.17487/RFC4151, October 2005,
<<http://www.rfc-editor.org/info/rfc4151>>.
- [RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB
Documents", BCP 111, RFC 4181, September 2005.
- [RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG

Data Model Documents", RFC 6087, January 2011.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.
- [RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <<http://www.rfc-editor.org/info/rfc7322>>.
- [RFC7841] Halpern, J., Ed., Daigle, L., Ed., and O. Kolkman, Ed., "RFC Streams, Headers, and Boilerplates", RFC 7841, DOI 10.17487/RFC7841, May 2016, <<http://www.rfc-editor.org/info/rfc7841>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://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>>.

Appendix A. Change Log

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

A.1. v19 to v20

- o fix examples

A.2. v18 to v19

- o address IESG ballot comments

A.3. v17 to v18

- o address Area Director review comments Part 2
- o clarify preferred list key order

A.4. v16 to v17

- o address Area Director review comments Part 1

A.5. v15 to v16

- o address Area Director review comments posted 2018-01-25

A.6. v15 to v16

- o address document shephard comments posted 2018-01-15
- o add yang-version to template module

A.7. v14 to v15

- o changed Intended status from Informational to BCP
- o update tree diagram guidelines section
- o Change IANA template to list IESG instead of NETMOD WG as the Registrant
- o Update some references

A.8. v13 to v14

- o Replaced sec. 4.23 Operational Data with Operational Data from NMDA text by Lou Berger and Kent Watsen

- o Added NMDA Terms section
- o Changed term operational data to operational state
- o Clarified that reference-stmt SHOULD be present in import-stmt

A.9. v12 to v13

- o Clarify that the revision-date SHOULD be used in a CODE BEGINS YANG file extraction macro.
- o Clarify the IANA requirements section wrt/ XML namespace and YANG module name registries.
- o Clarify YANG Usage section wrt/ XML and/or JSON encoding format.
- o Update Operation Data section to consider revised datastores.
- o Add reference to YANG Tree Diagrams and update 2 sections that use this reference.
- o Add reference to Revised Datastores and guidelines drafts

A.10. v11 to v12

- o fix incorrect location of new Module Usage Examples section

A.11. v10 to v11

- o updated YANG tree diagram syntax to align with pyang 1.7.1
- o added general guideline to include module usage examples

A.12. v09 to v10

- o clarified <CODE BEGINS> is only for normative modules
- o clarified example module namespace URI conventions
- o clarified pyang usage for normative and example modules
- o updated YANG tree diagrams section with text from RFC 8022

A.13. v08 to v09

- o fixed references

- o added mention of RESTCONF to abstract and intro
- o created separate section for code components
- o fixed document status

A.14. v07 to v08

- o changed CODE BEGINS guideline for example modules
- o updated tree diagram guidelines
- o clarified published and unpublished terms
- o added section on Empty and Boolean data types
- o clarified how to update the revision statement
- o updated operational state guidelines
- o added 'YANG fragment' to terminology section

A.15. v06 to v07

- o update contact statement guideline
- o update example modules guidelines
- o add guidelines on top-level data nodes
- o add guideline on use of NP containers
- o added guidelines on union types
- o add guideline on deviations
- o added section on open systems considerations
- o added guideline about definitions reserved for future use

A.16. v05 to v06

- o Changed example 'my-module' to 'example-module'
- o Added section Updating YANG Modules (Published vs. Unpublished)
- o Added Example Modules section

- o Added "<EXAMPLE BEGINS>" convention for full example modules
- o Added section on using action vs. rpc
- o Changed term "operational state" to "operational data"
- o Added section on YANG Data Node Constraints
- o Added guidelines on using must vs. when statements
- o Made ietf-foo module validate for I-D submission

A.17. v04 to v05

- o Clarified that YANG 1.1 SHOULD be used but YANG 1.0 MAY be used if no YANG 1.1 features needed
- o Changed SHOULD follow YANG naming conventions to MUST follow (for standards track documents only)
- o Clarified module naming conventions for normative modules, example modules, and modules from other SDOs.
- o Added prefix value selection guidelines
- o Added new section on guidelines for reusable groupings
- o Made header guidelines less IETF-specific
- o Added new section on guidelines for extension statements
- o Added guidelines for nested "choice" statement within a "case" statement

A.18. v03 to v04

- o Added sections for deviation statements and performance considerations
- o Added YANG 1.1 section
- o Updated YANG reference from 1.0 to 1.1

A.19. v02 to v03

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

A.20. v01 to v02

- o Updated draft based on mailing list comments.

A.21. v00 to v01

All issues from the issue tracker have been addressed.

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

- o Issue 1: Tree Diagrams: Added 'tree-diagrams' section so RFCs with YANG modules can use an Informative reference to this RFC for tree diagrams. Updated guidelines to reference this RFC when tree diagrams are used
- o Issue 2: XPath function restrictions: Added paragraphs in XPath usage section for 'id', 'namespace-uri', 'name', and 'lang' functions
- o Issue 3: XPath function document order issues: Added paragraph in XPath usage section about node-set ordering for 'local-name', 'namespace-uri', 'name', 'string' and 'number' functions. Also any function that implicitly converts a node-set to a string.
- o Issue 4: XPath preceding-sibling and following-sibling: Checked and text in XPath usage section already has proposed text from Lada.
- o Issue 5: XPath 'when-stmt' reference to descendant nodes: Added exception and example in XPath Usage section for augmented nodes.
- o Issue 6: XPath numeric conversions: Changed 'numeric expressions' to 'numeric and boolean expressions'
- o Issue 7: XPath module containment: Added sub-section on XPath wildcards
- o Issue 8: status-stmt usage: Added text to Lifecycle Management section about transitioning from active to deprecated and then to obsolete.
- o Issue 9: resource identification in notifications: Add text to Notifications section about identifying resources and using the leafref data type.
- o Issue 10: single quoted strings: Added text to Data Types section about using a single-quoted string for patterns.

Appendix B. Module Review Checklist

This section is adapted from RFC 4181.

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

- o I-D Boilerplate -- verify that the draft contains the required Internet-Draft boilerplate (see <https://www.ietf.org/id-info/guidelines.html>), including the appropriate statement to permit publication as an RFC, and that I-D boilerplate does not contain references or section numbers.
- o Abstract -- verify that the abstract does not contain references, that it does not have a section number, and that its content follows the guidelines in <https://www.ietf.org/id-info/guidelines.html>.
- o Copyright Notice -- verify that the draft has the appropriate text regarding the rights that document contributors provide to the IETF Trust [RFC5378]. Verify that it contains the full IETF Trust copyright notice at the beginning of the document. The IETF Trust Legal Provisions (TLP) can be found at:

<https://trustee.ietf.org/license-info/>

- o Security Considerations section -- verify that the draft uses the latest approved template from the OPS area website (<https://trac.tools.ietf.org/area/ops/trac/wiki/yang-security-guidelines>) and that the guidelines therein have been followed.
- o IANA Considerations section -- this section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:

XML Namespace Registry: Register the YANG module namespace.

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

- o References -- verify that the references are properly divided between normative and informative references, that RFC 2119 and RFC 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 OK to include an informative reference to a previous version of a specification to help explain a feature included for backward compatibility). Be sure citations for all imported modules are present somewhere in the document text (outside the YANG module). If a YANG module contains reference or description statements that refer to an Internet Draft (I-D), then the I-D is included as an Informative Reference.
- o License -- verify that the draft contains the Simplified BSD License in each YANG module or submodule. Some guidelines related to this requirement are described in Section 3.1. Make sure that the correct year is used in all copyright dates. Use the approved text from the latest Trust Legal Provisions (TLP) document, which can be found at:

<https://trustee.ietf.org/license-info/>

- o Other Issues -- check for any issues mentioned in <https://www.ietf.org/id-info/checklist.html> that are not covered elsewhere.
- o Technical Content -- review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information can be found at:

<https://trac.tools.ietf.org/wg/netconf/trac/wiki>

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

Appendix C. YANG Module Template

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

module ietf-template {

    yang-version 1.1;

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

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

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

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

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

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

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

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

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).
```

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

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

reference "RFC XXXX: <Replace With Document Title>";

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

// replace '2016-03-20' with the module publication date
// The format is (year-month-day)
revision "2016-03-20" {
    description "what changed in this revision";
    reference "document containing this module";
}

// extension statements

// feature statements

// identity statements

// typedef statements

// grouping statements

// data definition statements

// augment statements

// rpc statements

// notification statements

// DO NOT put deviation statements in a published module
}

<CODE ENDS>
```


Author's Address

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems
November 03, 2016

A YANG Data Model for Routing Management
draft-ietf-netmod-routing-cfg-25

Abstract

This document contains a specification of three YANG modules and one submodule. Together they form the core routing data model which serves as a framework for configuring and managing a routing subsystem. It is expected that these modules will be augmented by additional YANG modules defining data models for control plane protocols, route filters and other functions. The core routing data model provides common building blocks for such extensions -- routes, routing information bases (RIB), and controlplane protocols.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 7, 2017.

Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	4
2.1. Glossary of New Terms	5
2.2. Tree Diagrams	5
2.3. Prefixes in Data Node Names	6
3. Objectives	6
4. The Design of the Core Routing Data Model	7
4.1. System-Controlled and User-Controlled List Entries	8
5. Basic Building Blocks	9
5.1. Route	9
5.2. Routing Information Base (RIB)	9
5.3. Control Plane Protocol	10
5.3.1. Routing Pseudo-Protocols	10
5.3.2. Defining New Control Plane Protocols	11
5.4. Parameters of IPv6 Router Advertisements	12
6. Interactions with Other YANG Modules	13
6.1. Module "ietf-interfaces"	13
6.2. Module "ietf-ip"	13
7. Routing Management YANG Module	14
8. IPv4 Unicast Routing Management YANG Module	26
9. IPv6 Unicast Routing Management YANG Module	32
9.1. IPv6 Router Advertisements Submodule	37
10. IANA Considerations	47
11. Security Considerations	49
12. Acknowledgments	49
13. References	49
13.1. Normative References	50
13.2. Informative References	50
Appendix A. The Complete Data Trees	51
A.1. Configuration Data	51
A.2. State Data	53
Appendix B. Minimum Implementation	54
Appendix C. Example: Adding a New Control Plane Protocol	54
Appendix D. Data Tree Example	57
Appendix E. Change Log	65
E.1. Changes Between Versions -24 and -25	65
E.2. Changes Between Versions -23 and -24	65
E.3. Changes Between Versions -22 and -23	65
E.4. Changes Between Versions -21 and -22	66
E.5. Changes Between Versions -20 and -21	66
E.6. Changes Between Versions -19 and -20	66

E.7. Changes Between Versions -18 and -19	66
E.8. Changes Between Versions -17 and -18	66
E.9. Changes Between Versions -16 and -17	67
E.10. Changes Between Versions -15 and -16	67
E.11. Changes Between Versions -14 and -15	68
E.12. Changes Between Versions -13 and -14	68
E.13. Changes Between Versions -12 and -13	68
E.14. Changes Between Versions -11 and -12	69
E.15. Changes Between Versions -10 and -11	69
E.16. Changes Between Versions -09 and -10	70
E.17. Changes Between Versions -08 and -09	70
E.18. Changes Between Versions -07 and -08	70
E.19. Changes Between Versions -06 and -07	70
E.20. Changes Between Versions -05 and -06	71
E.21. Changes Between Versions -04 and -05	71
E.22. Changes Between Versions -03 and -04	72
E.23. Changes Between Versions -02 and -03	72
E.24. Changes Between Versions -01 and -02	73
E.25. Changes Between Versions -00 and -01	73
Authors' Addresses	74

1. Introduction

This document contains a specification of the following YANG modules:

- o Module "ietf-routing" provides generic components of a routing data model.
- o Module "ietf-ipv4-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv4 unicast.
- o Module "ietf-ipv6-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv6 unicast. Its submodule "ietf-ipv6-router-advertisements" also augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with IPv6 router configuration variables required by [RFC4861].

These modules together define the so-called core routing data model, which is intended as a basis for future data model development covering more sophisticated routing systems. While these three modules can be directly used for simple IP devices with static routing (see Appendix B), their main purpose is to provide essential building blocks for more complicated data models involving multiple control plane protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

2. Terminology and Notation

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

The following terms are defined in [RFC6241]:

- o client,
- o message,
- o protocol operation,
- o server.

The following terms are defined in [RFC7950]:

- o action,
- o augment,
- o configuration data,
- o container,
- o container with presence,
- o data model,
- o data node,
- o feature,
- o leaf,
- o list,
- o mandatory node,
- o module,
- o schema tree,
- o state data,
- o RPC operation.

2.1. Glossary of New Terms

core routing data model: YANG data model comprising "ietf-routing", "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing" modules.

direct route: a route to a directly connected network.

routing information base (RIB): An object containing a list of routes together with other information. See Section 5.2 for details.

system-controlled entry: An entry of a list in state data ("config false") that is created by the system independently of what has been explicitly configured. See Section 4.1 for details.

user-controlled entry: An entry of a list in state data ("config false") that is created and deleted as a direct consequence of certain configuration changes. See Section 4.1 for details.

2.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Appendix A, and similar diagrams of its various subtrees appear in the main text.

- o Brackets "[" and "]" enclose list keys.
- o Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- o Abbreviations before data node names: "rw" means configuration (read-write), "ro" state data (read-only), "-x" RPC operations or actions, and "-n" notifications.
- o Symbols after data node names: "?" means an optional node, "!" a container with presence, and "*" denotes a "list" or "leaf-list".
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2.3. Prefixes in Data Node Names

In this document, names of data nodes, actions and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
if	ietf-interfaces	[RFC7223]
ip	ietf-ip	[RFC7277]
rt	ietf-routing	Section 7
v4ur	ietf-ipv4-unicast-routing	Section 8
v6ur	ietf-ipv6-unicast-routing	Section 9
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and corresponding YANG modules

3. Objectives

The initial design of the core routing data model was driven by the following objectives:

- o The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).
- o A simple IP routing system, such as one that uses only static routing, should be configurable in a simple way, ideally without any need to develop additional YANG modules.
- o On the other hand, the core routing framework must allow for complicated implementations involving multiple routing information bases (RIB) and multiple control plane protocols, as well as controlled redistributions of routing information.
- o Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules and one submodule. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. Module "ietf-ipv6-unicast-routing" has a submodule, "ietf-ipv6-router-advertisements", that augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with configuration variables for IPv6 router advertisements as required by [RFC4861]. Figures 1 and 2 show abridged views of the configuration and state data hierarchies. See Appendix A for the complete data trees.

```
+--rw routing
  +--rw router-id?
  +--rw control-plane-protocols
    |   +--rw control-plane-protocol* [type name]
    |   |   +--rw type
    |   |   +--rw name
    |   |   +--rw description?
    |   |   +--rw static-routes
    |   |   |   +--rw v6ur:ipv6
    |   |   |   |   ...
    |   |   |   +--rw v4ur:ipv4
    |   |   |   |   ...
    |   |   |   ...
    |   |   ...
  +--rw ribs
    +--rw rib* [name]
      +--rw name
      +--rw address-family?
      +--rw description?
```

Figure 1: Configuration data hierarchy.

```
+--ro routing-state
  +--ro router-id?
  +--ro interfaces
  |   +--ro interface*
  +--ro control-plane-protocols
  |   +--ro control-plane-protocol* [type name]
  |       +--ro type
  |       +--ro name
  +--ro ribs
  |   +--ro rib* [name]
  |       +--ro name
  |       +--ro address-family
  |       +--ro default-rib?
  |       +--ro routes
  |           +--ro route*
  |           ...
```

Figure 2: State data hierarchy.

As can be seen from Figures 1 and 2, the core routing data model introduces several generic components of a routing framework: routes, RIBs containing lists of routes, and control plane protocols. Section 5 describes these components in more detail.

4.1. System-Controlled and User-Controlled List Entries

The core routing data model defines several lists in the schema tree, such as "rib", that have to be populated with at least one entry in any properly functioning device, and additional entries may be configured by a client.

In such a list, the server creates the required item as a so-called system-controlled entry in state data, i.e., inside the "routing-state" container.

An example can be seen in Appendix D: the "/routing-state/ribs/rib" list has two system-controlled entries named "ipv4-master" and "ipv6-master".

Additional entries may be created in the configuration by a client, e.g., via the NETCONF protocol. These are so-called user-controlled entries. If the server accepts a configured user-controlled entry, then this entry also appears in the state data version of the list.

Corresponding entries in both versions of the list (in state data and configuration) have the same value of the list key.

A client may also provide supplemental configuration of system-controlled entries. To do so, the client creates a new entry in the configuration with the desired contents. In order to bind this entry to the corresponding entry in the state data list, the key of the configuration entry has to be set to the same value as the key of the state entry.

Deleting a user-controlled entry from the configuration list results in the removal of the corresponding entry in the state data list. In contrast, if a system-controlled entry is deleted from the configuration list, only the extra configuration specified in that entry is removed but the corresponding state data entry remains in the list.

5. Basic Building Blocks

This section describes the essential components of the core routing data model.

5.1. Route

Routes are basic elements of information in a routing system. The core routing data model defines only the following minimal set of route attributes:

- o "destination-prefix": address prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.
- o "route-preference": an integer value (also known as administrative distance) that is used for selecting a preferred route among routes with the same destination prefix. A lower value means a more preferred route.
- o "next-hop": determines the outgoing interface and/or next-hop address(es), other operation to be performed with a packet.

Routes are primarily state data that appear as entries of RIBs (Section 5.2) but they may also be found in configuration data, for example as manually configured static routes. In the latter case, configurable route attributes are generally a subset of attributes defined for RIB routes.

5.2. Routing Information Base (RIB)

Every implementation of the core routing data model manages one or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Each RIB contains only routes

of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

In the core routing data model, RIBs are state data represented as entries of the list "/routing-state/ribs/rib". The contents of RIBs are controlled and manipulated by control plane protocol operations which may result in route additions, removals and modifications. This also includes manipulations via the "static" and/or "direct" pseudo-protocols, see Section 5.3.1.

For every supported address family, exactly one RIB MUST be marked as the so-called default RIB. Its role is explained in Section 5.3.

Simple router implementations that do not advertise the feature "multiple-ribs" will typically create one system-controlled RIB per supported address family, and mark it as the default RIB.

More complex router implementations advertising the "multiple-ribs" feature support multiple RIBs per address family that can be used for policy routing and other purposes.

The following action (see Section 7.15 of [RFC7950]) is defined for the "rib" list:

- o active-route -- return the active RIB route for the destination address that is specified as the action's input parameter.

5.3. Control Plane Protocol

The core routing data model provides an open-ended framework for defining multiple control plane protocol instances, e.g., for Layer 3 routing protocols. Each control plane protocol instance MUST be assigned a type, which is an identity derived from the "rt:control-plane-protocol" base identity. The core routing data model defines two identities for the direct and static pseudo-protocols (Section 5.3.1).

Multiple control plane protocol instances of the same type MAY be configured.

5.3.1. Routing Pseudo-Protocols

The core routing data model defines two special routing protocol types -- "direct" and "static". Both are in fact pseudo-protocols, which means that they are confined to the local device and do not exchange any routing information with adjacent routers.

Every implementation of the core routing data model MUST provide exactly one instance of the "direct" pseudo-protocol type. It is the source of direct routes for all configured address families. Direct routes are normally supplied by the operating system kernel, based on the configuration of network interface addresses, see Section 6.2.

A pseudo-protocol of the type "static" allows for specifying routes manually. It MAY be configured in zero or multiple instances, although a typical configuration will have exactly one instance.

5.3.2. Defining New Control Plane Protocols

It is expected that future YANG modules will create data models for additional control plane protocol types. Such a new module has to define the protocol-specific configuration and state data, and it has to integrate it into the core routing framework in the following way:

- o A new identity MUST be defined for the control plane protocol and its base identity MUST be set to "rt:control-plane-protocol", or to an identity derived from "rt:control-plane-protocol".
- o Additional route attributes MAY be defined, preferably in one place by means of defining a YANG grouping. The new attributes have to be inserted by augmenting the definitions of the nodes

```
/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route
```

and

```
/rt:routing-state/rt:ribs/rt:rib/rt:output/rt:route,
```

and possibly other places in the configuration, state data, notifications, and input/output parameters of actions or RPC operations.

- o Configuration parameters and/or state data for the new protocol can be defined by augmenting the "control-plane-protocol" data node under both "/routing" and "/routing-state".

By using a "when" statement, the augmented configuration parameters and state data specific to the new protocol SHOULD be made conditional and valid only if the value of "rt:type" or "rt:source-protocol" is equal to (or derived from) the new protocol's identity.

It is also RECOMMENDED that protocol-specific data nodes be encapsulated in an appropriately named container with presence. Such a container may contain mandatory data nodes that are otherwise forbidden at the top level of an augment.

The above steps are implemented by the example YANG module for the RIP routing protocol in Appendix C.

5.4. Parameters of IPv6 Router Advertisements

YANG module "ietf-ipv6-router-advertisements" (Section 9.1), which is a submodule of the "ietf-ipv6-unicast-routing" module, augments the configuration and state data of IPv6 interfaces with definitions of the following variables as required by [RFC4861], sec. 6.2.1:

- o send-advertisements,
- o max-rtr-adv-interval,
- o min-rtr-adv-interval,
- o managed-flag,
- o other-config-flag,
- o link-mtu,
- o reachable-time,
- o retrans-timer,
- o cur-hop-limit,
- o default-lifetime,
- o prefix-list: a list of prefixes to be advertised.

The following parameters are associated with each prefix in the list:

- * valid-lifetime,
- * on-link-flag,
- * preferred-lifetime,
- * autonomous-flag.

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [RFC7277] (leaf "ip:forwarding").

2. The original specification [RFC4861] allows the implementations to decide whether the "valid-lifetime" and "preferred-lifetime" parameters remain the same in consecutive advertisements, or decrement in real time. However, the latter behavior seems problematic because the values might be reset again to the (higher) configured values after a configuration is reloaded. Moreover, no implementation is known to use the decrementing behavior. The "ietf-ipv6-router-advertisements" submodule therefore stipulates the former behavior with constant values.

6. Interactions with Other YANG Modules

The semantics of the core routing data model also depends on several configuration parameters that are defined in other YANG modules.

6.1. Module "ietf-interfaces"

The following boolean switch is defined in the "ietf-interfaces" YANG module [RFC7223]:

```
/if:interfaces/if:interface/if:enabled
```

If this switch is set to "false" for a network layer interface, then all routing and forwarding functions MUST be disabled on that interface.

6.2. Module "ietf-ip"

The following boolean switches are defined in the "ietf-ip" YANG module [RFC7277]:

```
/if:interfaces/if:interface/ip:ipv4/ip:enabled
```

If this switch is set to "false" for a network layer interface, then all IPv4 routing and forwarding functions MUST be disabled on that interface.

```
/if:interfaces/if:interface/ip:ipv4/ip:forwarding
```

If this switch is set to "false" for a network layer interface, then the forwarding of IPv4 datagrams through this interface MUST be disabled. However, the interface MAY participate in other IPv4 routing functions, such as routing protocols.

```
/if:interfaces/if:interface/ip:ipv6/ip:enabled
```

If this switch is set to "false" for a network layer interface, then all IPv6 routing and forwarding functions MUST be disabled on that interface.

```
/if:interfaces/if:interface/ip:ipv6/ip:forwarding
```

If this switch is set to "false" for a network layer interface, then the forwarding of IPv6 datagrams through this interface MUST be disabled. However, the interface MAY participate in other IPv6 routing functions, such as routing protocols.

In addition, the "ietf-ip" module allows for configuring IPv4 and IPv6 addresses and network prefixes or masks on network layer interfaces. Configuration of these parameters on an enabled interface MUST result in an immediate creation of the corresponding direct route. The destination prefix of this route is set according to the configured IP address and network prefix/mask, and the interface is set as the outgoing interface for that route.

7. Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-routing@2016-11-03.yang"
```

```
module ietf-routing {  
  
    yang-version "1.1";  
  
    namespace "urn:ietf:params:xml:ns:yang:ietf-routing";  
  
    prefix "rt";  
  
    import ietf-yang-types {  
        prefix "yang";  
    }  
  
    import ietf-interfaces {  
        prefix "if";  
    }  
  
    organization  
        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";  
  
    contact  
        "WG Web:    <https://tools.ietf.org/wg/netmod/>
```


WG List: <mailto:netmod@ietf.org>

WG Chair: Lou Berger
<mailto:lberger@labn.net>

WG Chair: Kent Watsen
<mailto:kwatsen@juniper.net>

Editor: Ladislav Lhotka
<mailto:lhotka@nic.cz>

Editor: Acee Lindem
<mailto:acee@cisco.com>";

description

"This YANG module defines essential components for the management of a routing subsystem.

Copyright (c) 2016 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119 (<https://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<https://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices.";

```
revision 2016-11-03 {  
  description  
    "Initial revision."  
  reference  
    "RFC XXXX: A YANG Data Model for Routing Management";  
}
```

```
/* Features */
```

```
feature multiple-ribs {  
  description
```

```
"This feature indicates that the server supports user-defined
RIBs.

Servers that do not advertise this feature SHOULD provide
exactly one system-controlled RIB per supported address family
and make them also the default RIBs. These RIBs then appear as
entries of the list /routing-state/ribs/rib.";
}

feature router-id {
  description
    "This feature indicates that the server supports configuration
    of an explicit 32-bit router ID that is used by some routing
    protocols.

    Servers that do not advertise this feature set a router ID
    algorithmically, usually to one of configured IPv4 addresses.
    However, this algorithm is implementation-specific.";
}

/* Identities */

identity address-family {
  description
    "Base identity from which identities describing address
    families are derived.";
}

identity ipv4 {
  base address-family;
  description
    "This identity represents IPv4 address family.";
}

identity ipv6 {
  base address-family;
  description
    "This identity represents IPv6 address family.";
}

identity control-plane-protocol {
  description
    "Base identity from which control plane protocol identities are
    derived.";
}

identity routing-protocol {
  base control-plane-protocol;
```

```
    description
      "Identity from which Layer 3 routing protocol identities are
       derived.";
  }

  identity direct {
    base routing-protocol;
    description
      "Routing pseudo-protocol that provides routes to directly
       connected networks.";
  }

  identity static {
    base routing-protocol;
    description
      "Static routing pseudo-protocol.";
  }

  /* Type Definitions */

  typedef route-preference {
    type uint32;
    description
      "This type is used for route preferences.";
  }

  /* Groupings */

  grouping address-family {
    description
      "This grouping provides a leaf identifying an address
       family.";
    leaf address-family {
      type identityref {
        base address-family;
      }
      mandatory "true";
      description
        "Address family.";
    }
  }

  grouping router-id {
    description
      "This grouping provides router ID.";
    leaf router-id {
      type yang:dotted-quad;
      description
```

```
        "A 32-bit number in the form of a dotted quad that is used by
        some routing protocols identifying a router.";
    reference
        "RFC 2328: OSPF Version 2.";
    }
}

grouping special-next-hop {
    description
        "This grouping provides a leaf with an enumeration of special
        next-hops.";
    leaf special-next-hop {
        type enumeration {
            enum blackhole {
                description
                    "Silently discard the packet.";
            }
            enum unreachable {
                description
                    "Discard the packet and notify the sender with an error
                    message indicating that the destination host is
                    unreachable.";
            }
            enum prohibit {
                description
                    "Discard the packet and notify the sender with an error
                    message indicating that the communication is
                    administratively prohibited.";
            }
            enum receive {
                description
                    "The packet will be received by the local system.";
            }
        }
    }
    description
        "Special next-hop options.";
}

grouping next-hop-content {
    description
        "Generic parameters of next-hops in static routes.";
    choice next-hop-options {
        mandatory "true";
        description
            "Options for next-hops in static routes."
    }
}
```

It is expected that further cases will be added through

```

    augments from other modules.";
case simple-next-hop {
  description
    "This case represents a simple next hop consisting of the
    next-hop address and/or outgoing interface.

    Modules for address families MUST augment this case with a
    leaf containing a next-hop address of that address
    family.";
  leaf outgoing-interface {
    type if:interface-ref;
    description
      "Name of the outgoing interface.";
  }
}
case special-next-hop {
  uses special-next-hop;
}
case next-hop-list {
  container next-hop-list {
    description
      "Container for multiple next-hops.";
    list next-hop {
      key "index";
      description
        "An entry of a next-hop list.

        Modules for address families MUST augment this list
        with a leaf containing a next-hop address of that
        address family.";
      leaf index {
        type string;
        description
          "An user-specified identifier utilised to uniquely
          reference the next-hop entry in the next-hop list.
          The value of this index has no semantic meaning
          other than for referencing the entry.";
      }
      leaf outgoing-interface {
        type if:interface-ref;
        description
          "Name of the outgoing interface.";
      }
    }
  }
}
}

```

```

grouping next-hop-state-content {
  description
    "Generic parameters of next-hops in state data.";
  choice next-hop-options {
    mandatory "true";
    description
      "Options for next-hops in state data.

      It is expected that further cases will be added through
      augments from other modules, e.g., for recursive
      next-hops.";
    case simple-next-hop {
      description
        "This case represents a simple next hop consisting of the
        next-hop address and/or outgoing interface.

        Modules for address families MUST augment this case with a
        leaf containing a next-hop address of that address
        family.";
      leaf outgoing-interface {
        type if:interface-state-ref;
        description
          "Name of the outgoing interface.";
      }
    }
    case special-next-hop {
      uses special-next-hop;
    }
    case next-hop-list {
      container next-hop-list {
        description
          "Container for multiple next-hops.";
        list next-hop {
          description
            "An entry of a next-hop list.

            Modules for address families MUST augment this list
            with a leaf containing a next-hop address of that
            address family.";
          leaf outgoing-interface {
            type if:interface-state-ref;
            description
              "Name of the outgoing interface.";
          }
        }
      }
    }
  }
}

```

```
    }

    grouping route-metadata {
      description
        "Common route metadata.";
      leaf source-protocol {
        type identityref {
          base routing-protocol;
        }
        mandatory "true";
        description
          "Type of the routing protocol from which the route
           originated.";
      }
      leaf active {
        type empty;
        description
          "Presence of this leaf indicates that the route is preferred
           among all routes in the same RIB that have the same
           destination prefix.";
      }
      leaf last-updated {
        type yang:date-and-time;
        description
          "Time stamp of the last modification of the route. If the
           route was never modified, it is the time when the route was
           inserted into the RIB.";
      }
    }
  }
}

/* State data */

container routing-state {
  config "false";
  description
    "State data of the routing subsystem.";
  uses router-id {
    description
      "Global router ID.

       It may be either configured or assigned algorithmically by
       the implementation.";
  }
  container interfaces {
    description
      "Network layer interfaces used for routing.";
    leaf-list interface {
      type if:interface-state-ref;
    }
  }
}
```

```
        description
            "Each entry is a reference to the name of a configured
            network layer interface.";
    }
}
container control-plane-protocols {
    description
        "Container for the list of routing protocol instances.";
    list control-plane-protocol {
        key "type name";
        description
            "State data of a control plane protocol instance.

            An implementation MUST provide exactly one
            system-controlled instance of the 'direct'
            pseudo-protocol. Instances of other control plane
            protocols MAY be created by configuration.";
        leaf type {
            type identityref {
                base control-plane-protocol;
            }
            description
                "Type of the control plane protocol.";
        }
        leaf name {
            type string;
            description
                "The name of the control plane protocol instance.

                For system-controlled instances this name is persistent,
                i.e., it SHOULD NOT change across reboots.";
        }
    }
}
container ribs {
    description
        "Container for RIBs.";
    list rib {
        key "name";
        min-elements "1";
        description
            "Each entry represents a RIB identified by the 'name' key.
            All routes in a RIB MUST belong to the same address
            family.

            An implementation SHOULD provide one system-controlled
            default RIB for each supported address family.";
        leaf name {
```



```
    type string;
    description
        "The name of the RIB.";
}
uses address-family;
leaf default-rib {
    if-feature "multiple-ribs";
    type boolean;
    default "true";
    description
        "This flag has the value of 'true' if and only if the RIB
        is the default RIB for the given address family.

        By default, control plane protocols place their routes
        in the default RIBs.";
}
container routes {
    description
        "Current content of the RIB.";
    list route {
        description
            "A RIB route entry. This data node MUST be augmented
            with information specific for routes of each address
            family.";
        leaf route-preference {
            type route-preference;
            description
                "This route attribute, also known as administrative
                distance, allows for selecting the preferred route
                among routes with the same destination prefix. A
                smaller value means a more preferred route.";
        }
        container next-hop {
            description
                "Route's next-hop attribute.";
            uses next-hop-state-content;
        }
        uses route-metadata;
    }
}
action active-route {
    description
        "Return the active RIB route that is used for the
        destination address.

        Address family specific modules MUST augment input
        parameters with a leaf named 'destination-address'.";
    output {
```

```

    container route {
      description
        "The active RIB route for the specified destination.

        If no route exists in the RIB for the destination
        address, no output is returned.

        Address family specific modules MUST augment this
        container with appropriate route contents.";
      container next-hop {
        description
          "Route's next-hop attribute.";
        uses next-hop-state-content;
      }
      uses route-metadata;
    }
  }
}

/* Configuration Data */

container routing {
  description
    "Configuration parameters for the routing subsystem.";
  uses router-id {
    if-feature "router-id";
    description
      "Configuration of the global router ID. Routing protocols
      that use router ID can use this parameter or override it
      with another value.";
  }
  container control-plane-protocols {
    description
      "Configuration of control plane protocol instances.";
    list control-plane-protocol {
      key "type name";
      description
        "Each entry contains configuration of a control plane
        protocol instance.";
      leaf type {
        type identityref {
          base control-plane-protocol;
        }
      }
      description
        "Type of the control plane protocol - an identity derived

```

```
        from the 'control-plane-protocol' base identity.";
    }
    leaf name {
        type string;
        description
            "An arbitrary name of the control plane protocol
            instance.";
    }
    leaf description {
        type string;
        description
            "Textual description of the control plane protocol
            instance.";
    }
    container static-routes {
        when "derived-from-or-self(..../type, 'rt:static')" {
            description
                "This container is only valid for the 'static' routing
                protocol.";
        }
        description
            "Configuration of the 'static' pseudo-protocol.

            Address-family-specific modules augment this node with
            their lists of routes.";
    }
}
}
container ribs {
    description
        "Configuration of RIBs.";
    list rib {
        key "name";
        description
            "Each entry contains configuration for a RIB identified by
            the 'name' key.

            Entries having the same key as a system-controlled entry
            of the list /routing-state/ribs/rib are used for
            configuring parameters of that entry. Other entries define
            additional user-controlled RIBs.";
        leaf name {
            type string;
            description
                "The name of the RIB.

                For system-controlled entries, the value of this leaf
                must be the same as the name of the corresponding entry
```

```

in state data.

For user-controlled entries, an arbitrary name can be
used.";
}
uses address-family {
    description
        "Address family of the RIB.

        It is mandatory for user-controlled RIBs. For
        system-controlled RIBs it can be omitted, otherwise it
        must match the address family of the corresponding state
        entry.";
    refine "address-family" {
        mandatory "false";
    }
}
leaf description {
    type string;
    description
        "Textual description of the RIB.";
}
}
}
}
}
<CODE ENDS>
```

8. IPv4 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-ipv4-unicast-routing@2016-11-03.yang"

module ietf-ipv4-unicast-routing {

    yang-version "1.1";

    namespace "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing";

    prefix "v4ur";

    import ietf-routing {
        prefix "rt";
    }
}
```

```
import ietf-inet-types {
  prefix "inet";
}

organization
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

contact
  "WG Web:    <https://tools.ietf.org/wg/netmod/>
  WG List:    <mailto:netmod@ietf.org>

  WG Chair:   Lou Berger
              <mailto:lberger@labn.net>

  WG Chair:   Kent Watsen
              <mailto:kwatsen@juniper.net>

  Editor:     Ladislav Lhotka
              <mailto:lhotka@nic.cz>

  Editor:     Acee Lindem
              <mailto:acee@cisco.com>";

description
  "This YANG module augments the 'ietf-routing' module with basic
  configuration and state data for IPv4 unicast routing.

  Copyright (c) 2016 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject to
  the license terms contained in, the Simplified BSD License set
  forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
  NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
  'OPTIONAL' in the module text are to be interpreted as described
  in RFC 2119 (https://tools.ietf.org/html/rfc2119).

  This version of this YANG module is part of RFC XXXX
  (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
  full legal notices.";

revision 2016-11-03 {
  description
```

```
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Identities */

identity ipv4-unicast {
  base rt:ipv4;
  description
    "This identity represents the IPv4 unicast address family.";
}

/* State data */

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v4ur:ipv4-unicast')" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "This leaf augments an IPv4 unicast route.";
  leaf destination-prefix {
    type inet:ipv4-prefix;
    description
      "IPv4 destination prefix.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
  + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v4ur:ipv4-unicast')" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "Augment 'simple-next-hop' case in IPv4 unicast routes.";
  leaf next-hop-address {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
  + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
```

```
    + "rt:next-hop-list/rt:next-hop" {
when "derived-from-or-self(..../rt:address-family, "
+ "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast.";
}
description
  "This leaf augments the 'next-hop-list' case of IPv4 unicast
  routes.";
leaf address {
  type inet:ipv4-address;
  description
    "IPv4 address of the next-hop.";
}
}

augment
  "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
when "derived-from-or-self(..../rt:address-family, "
+ "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast RIBs.";
}
description
  "This augment adds the input parameter of the 'active-route'
  action.";
leaf destination-address {
  type inet:ipv4-address;
  description
    "IPv4 destination address.";
}
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route" {
when "derived-from-or-self(..../rt:address-family, "
+ "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast.";
}
description
  "This augment adds the destination prefix to the reply of the
  'active-route' action.";
leaf destination-prefix {
  type inet:ipv4-prefix;
  description
    "IPv4 destination prefix.";
}
}
```

```
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:simple-next-hop" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v4ur:ipv4-unicast')" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "Augment 'simple-next-hop' case in the reply to the
    'active-route' action.";
  leaf next-hop-address {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v4ur:ipv4-unicast')" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "Augment 'next-hop-list' case in the reply to the
    'active-route' action.";
  leaf next-hop-address {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

/* Configuration data */

augment "/rt:routing/rt:control-plane-protocols/"
  + "rt:control-plane-protocol/rt:static-routes" {
  description
    "This augment defines the configuration of the 'static'
    pseudo-protocol with data specific to IPv4 unicast.";
  container ipv4 {
    description
      "Configuration of a 'static' pseudo-protocol instance
```



```

    consists of a list of routes.";
list route {
  key "destination-prefix";
  description
    "A list of static routes.";
  leaf destination-prefix {
    type inet:ipv4-prefix;
    mandatory "true";
    description
      "IPv4 destination prefix.";
  }
  leaf description {
    type string;
    description
      "Textual description of the route.";
  }
  container next-hop {
    description
      "Configuration of next-hop.";
    uses rt:next-hop-content {
      augment "next-hop-options/simple-next-hop" {
        description
          "Augment 'simple-next-hop' case in IPv4 static
          routes.";
        leaf next-hop-address {
          type inet:ipv4-address;
          description
            "IPv4 address of the next-hop.";
        }
      }
      augment "next-hop-options/next-hop-list/next-hop-list/"
        + "next-hop" {
        description
          "Augment 'next-hop-list' case in IPv4 static
          routes.";
        leaf next-hop-address {
          type inet:ipv4-address;
          description
            "IPv4 address of the next-hop.";
        }
      }
    }
  }
}

```

<CODE ENDS>

9. IPv6 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-ipv6-unicast-routing@2016-11-03.yang"

```
module ietf-ipv6-unicast-routing {  
  
    yang-version "1.1";  
  
    namespace "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";  
  
    prefix "v6ur";  
  
    import ietf-routing {  
        prefix "rt";  
    }  
  
    import ietf-inet-types {  
        prefix "inet";  
    }  
  
    include ietf-ipv6-router-advertisements {  
        revision-date 2016-11-03;  
    }  
  
    organization  
        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";  
  
    contact  
        "WG Web:    <https://tools.ietf.org/wg/netmod/>  
        WG List:    <mailto:netmod@ietf.org>  
  
        WG Chair: Lou Berger  
                    <mailto:lberger@labn.net>  
  
        WG Chair: Kent Watsen  
                    <mailto:kwatsen@juniper.net>  
  
        Editor:    Ladislav Lhotka  
                    <mailto:lhotka@nic.cz>  
  
        Editor:    Acee Lindem  
                    <mailto:acee@cisco.com>";
```

`description`

"This YANG module augments the 'ietf-routing' module with basic configuration and state data for IPv6 unicast routing.

Copyright (c) 2016 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119 (<https://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<https://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices."

```
revision 2016-11-03 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Identities */

identity ipv6-unicast {
  base rt:ipv6;
  description
    "This identity represents the IPv6 unicast address family.";
}

/* State data */

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v6ur:ipv6-unicast')";
  description
    "This augment is valid only for IPv6 unicast.";
}
description
  "This leaf augments an IPv6 unicast route.";
```

```
    leaf destination-prefix {
      type inet:ipv6-prefix;
      description
        "IPv6 destination prefix.";
    }
  }

  augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
    + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
    when "derived-from-or-self(..../rt:address-family, "
      + "'v6ur:ipv6-unicast')" {
      description
        "This augment is valid only for IPv6 unicast.";
    }
    description
      "Augment 'simple-next-hop' case in IPv6 unicast routes.";
    leaf next-hop-address {
      type inet:ipv6-address;
      description
        "IPv6 address of the next-hop.";
    }
  }

  augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
    + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
    + "rt:next-hop-list/rt:next-hop" {
    when "derived-from-or-self(..../rt:address-family, "
      + "'v6ur:ipv6-unicast')" {
      description
        "This augment is valid only for IPv6 unicast.";
    }
    description
      "This leaf augments the 'next-hop-list' case of IPv6 unicast
        routes.";
    leaf address {
      type inet:ipv6-address;
      description
        "IPv6 address of the next-hop.";
    }
  }

  augment
    "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
    when "derived-from-or-self(../rt:address-family, "
      + "'v6ur:ipv6-unicast')" {
      description
        "This augment is valid only for IPv6 unicast RIBs.";
    }
  }
```

```
    description
      "This augment adds the input parameter of the 'active-route'
      action.";
    leaf destination-address {
      type inet:ipv6-address;
      description
        "IPv6 destination address.";
    }
  }

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v6ur:ipv6-unicast')" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "This augment adds the destination prefix to the reply of the
    'active-route' action.";
  leaf destination-prefix {
    type inet:ipv6-prefix;
    description
      "IPv6 destination prefix.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:simple-next-hop" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v6ur:ipv6-unicast')" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "Augment 'simple-next-hop' case in the reply to the
    'active-route' action.";
  leaf next-hop-address {
    type inet:ipv6-address;
    description
      "IPv6 address of the next-hop.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
```

```
when "derived-from-or-self(.../.../.../rt:address-family, "
  + "'v6ur:ipv6-unicast')" {
  description
    "This augment is valid only for IPv6 unicast.";
}
description
  "Augment 'next-hop-list' case in the reply to the
  'active-route' action.";
leaf next-hop-address {
  type inet:ipv6-address;
  description
    "IPv6 address of the next-hop.";
}
}

/* Configuration data */

augment "/rt:routing/rt:control-plane-protocols/"
  + "rt:control-plane-protocol/rt:static-routes" {
  description
    "This augment defines the configuration of the 'static'
    pseudo-protocol with data specific to IPv6 unicast.";
  container ipv6 {
    description
      "Configuration of a 'static' pseudo-protocol instance
      consists of a list of routes.";
    list route {
      key "destination-prefix";
      description
        "A list of static routes.";
      leaf destination-prefix {
        type inet:ipv6-prefix;
        mandatory "true";
        description
          "IPv6 destination prefix.";
      }
      leaf description {
        type string;
        description
          "Textual description of the route.";
      }
    }
    container next-hop {
      description
        "Configuration of next-hop.";
      uses rt:next-hop-content {
        augment "next-hop-options/simple-next-hop" {
          description
            "Augment 'simple-next-hop' case in IPv6 static
```

```
        routes.";
    leaf next-hop-address {
        type inet:ipv6-address;
        description
            "IPv6 address of the next-hop.";
    }
}
augment "next-hop-options/next-hop-list/next-hop-list/"
+ "next-hop" {
    description
        "Augment 'next-hop-list' case in IPv6 static
         routes.";
    leaf next-hop-address {
        type inet:ipv6-address;
        description
            "IPv6 address of the next-hop.";
    }
}
}
```

<CODE ENDS>

9.1. IPv6 Router Advertisements Submodule

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-ipv6-router-advertisements@2016-11-03.yang"
```

```
submodule ietf-ipv6-router-advertisements {
    yang-version "1.1";

    belongs-to ietf-ipv6-unicast-routing {
        prefix "v6ur";
    }

    import ietf-inet-types {
        prefix "inet";
    }

    import ietf-interfaces {
```

```
    prefix "if";
  }

  import ietf-ip {
    prefix "ip";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:    <https://tools.ietf.org/wg/netmod/>
     WG List:   <mailto:netmod@ietf.org>

     WG Chair:  Lou Berger
                <mailto:lberger@labn.net>

     WG Chair:  Kent Watsen
                <mailto:kwatsen@juniper.net>

     Editor:    Ladislav Lhotka
                <mailto:lhotka@nic.cz>

     Editor:    Acee Lindem
                <mailto:acee@cisco.com>";

  description
    "This YANG module augments the 'ietf-ip' module with
    configuration and state data of IPv6 router advertisements.

    Copyright (c) 2016 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject to
    the license terms contained in, the Simplified BSD License set
    forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
    NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
    'OPTIONAL' in the module text are to be interpreted as described
    in RFC 2119 (https://tools.ietf.org/html/rfc2119).

    This version of this YANG module is part of RFC XXXX
    (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
    full legal notices.";
```



```
reference
  "RFC 4861: Neighbor Discovery for IP version 6 (IPv6).";

revision 2016-11-03 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* State data */

augment "/if:interfaces-state/if:interface/ip:ipv6" {
  description
    "Augment interface state data with parameters of IPv6 router
    advertisements.";
  container ipv6-router-advertisements {
    description
      "Parameters of IPv6 Router Advertisements.";
    leaf send-advertisements {
      type boolean;
      description
        "A flag indicating whether or not the router sends periodic
        Router Advertisements and responds to Router
        Solicitations.";
    }
    leaf max-rtr-adv-interval {
      type uint16 {
        range "4..1800";
      }
      units "seconds";
      description
        "The maximum time allowed between sending unsolicited
        multicast Router Advertisements from the interface.";
    }
    leaf min-rtr-adv-interval {
      type uint16 {
        range "3..1350";
      }
      units "seconds";
      description
        "The minimum time allowed between sending unsolicited
        multicast Router Advertisements from the interface.";
    }
    leaf managed-flag {
      type boolean;
      description
        "The value that is placed in the 'Managed address
```

```
        configuration' flag field in the Router Advertisement.";
    }
    leaf other-config-flag {
        type boolean;
        description
            "The value that is placed in the 'Other configuration' flag
            field in the Router Advertisement.";
    }
    leaf link-mtu {
        type uint32;
        description
            "The value that is placed in MTU options sent by the
            router. A value of zero indicates that no MTU options are
            sent.";
    }
    leaf reachable-time {
        type uint32 {
            range "0..3600000";
        }
        units "milliseconds";
        description
            "The value that is placed in the Reachable Time field in
            the Router Advertisement messages sent by the router. A
            value of zero means unspecified (by this router).";
    }
    leaf retrans-timer {
        type uint32;
        units "milliseconds";
        description
            "The value that is placed in the Retrans Timer field in the
            Router Advertisement messages sent by the router. A value
            of zero means unspecified (by this router).";
    }
    leaf cur-hop-limit {
        type uint8;
        description
            "The value that is placed in the Cur Hop Limit field in the
            Router Advertisement messages sent by the router. A value
            of zero means unspecified (by this router).";
    }
    leaf default-lifetime {
        type uint16 {
            range "0..9000";
        }
        units "seconds";
        description
            "The value that is placed in the Router Lifetime field of
            Router Advertisements sent from the interface, in seconds.
```

```
        A value of zero indicates that the router is not to be
        used as a default router.";
    }
    container prefix-list {
        description
            "A list of prefixes that are placed in Prefix Information
            options in Router Advertisement messages sent from the
            interface.

            By default, these are all prefixes that the router
            advertises via routing protocols as being on-link for the
            interface from which the advertisement is sent.";
        list prefix {
            key "prefix-spec";
            description
                "Advertised prefix entry and its parameters.";
            leaf prefix-spec {
                type inet:ipv6-prefix;
                description
                    "IPv6 address prefix.";
            }
            leaf valid-lifetime {
                type uint32;
                units "seconds";
                description
                    "The value that is placed in the Valid Lifetime in the
                    Prefix Information option. The designated value of all
                    1's (0xffffffff) represents infinity.

                    An implementation SHOULD keep this value constant in
                    consecutive advertisements except when it is
                    explicitly changed in configuration.";
            }
            leaf on-link-flag {
                type boolean;
                description
                    "The value that is placed in the on-link flag ('L-bit')
                    field in the Prefix Information option.";
            }
            leaf preferred-lifetime {
                type uint32;
                units "seconds";
                description
                    "The value that is placed in the Preferred Lifetime in
                    the Prefix Information option, in seconds. The
                    designated value of all 1's (0xffffffff) represents
                    infinity.
```

```
        An implementation SHOULD keep this value constant in
        consecutive advertisements except when it is
        explicitly changed in configuration.";
    }
    leaf autonomous-flag {
        type boolean;
        description
            "The value that is placed in the Autonomous Flag field
            in the Prefix Information option.";
    }
}
}
}

/* Configuration data */

augment "/if:interfaces/if:interface/ip:ipv6" {
    description
        "Augment interface configuration with parameters of IPv6 router
        advertisements.";
    container ipv6-router-advertisements {
        description
            "Configuration of IPv6 Router Advertisements.";
        leaf send-advertisements {
            type boolean;
            default "false";
            description
                "A flag indicating whether or not the router sends periodic
                Router Advertisements and responds to Router
                Solicitations.";
            reference
                "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
                AdvSendAdvertisements.";
        }
        leaf max-rtr-adv-interval {
            type uint16 {
                range "4..1800";
            }
            units "seconds";
            default "600";
            description
                "The maximum time allowed between sending unsolicited
                multicast Router Advertisements from the interface.";
            reference
                "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
                MaxRtrAdvInterval.";
        }
    }
}
```

```
leaf min-rtr-adv-interval {
  type uint16 {
    range "3..1350";
  }
  units "seconds";
  must ". <= 0.75 * ../max-rtr-adv-interval" {
    description
      "The value MUST NOT be greater than 75 % of
       'max-rtr-adv-interval'.";
  }
  description
    "The minimum time allowed between sending unsolicited
     multicast Router Advertisements from the interface.

     The default value to be used operationally if this leaf is
     not configured is determined as follows:

     - if max-rtr-adv-interval >= 9 seconds, the default value
       is 0.33 * max-rtr-adv-interval;

     - otherwise it is 0.75 * max-rtr-adv-interval.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     MinRtrAdvInterval.";
}
leaf managed-flag {
  type boolean;
  default "false";
  description
    "The value to be placed in the 'Managed address
     configuration' flag field in the Router Advertisement.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvManagedFlag.";
}
leaf other-config-flag {
  type boolean;
  default "false";
  description
    "The value to be placed in the 'Other configuration' flag
     field in the Router Advertisement.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvOtherConfigFlag.";
}
leaf link-mtu {
  type uint32;
  default "0";
```

```
description
  "The value to be placed in MTU options sent by the router.
   A value of zero indicates that no MTU options are sent.";
reference
  "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
   AdvLinkMTU.";
}
leaf reachable-time {
  type uint32 {
    range "0..3600000";
  }
  units "milliseconds";
  default "0";
  description
    "The value to be placed in the Reachable Time field in the
     Router Advertisement messages sent by the router. A value
     of zero means unspecified (by this router).";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvReachableTime.";
}
leaf retrans-timer {
  type uint32;
  units "milliseconds";
  default "0";
  description
    "The value to be placed in the Retrans Timer field in the
     Router Advertisement messages sent by the router. A value
     of zero means unspecified (by this router).";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvRetransTimer.";
}
leaf cur-hop-limit {
  type uint8;
  description
    "The value to be placed in the Cur Hop Limit field in the
     Router Advertisement messages sent by the router. A value
     of zero means unspecified (by this router).

    If this parameter is not configured, the device SHOULD use
    the value specified in IANA Assigned Numbers that was in
    effect at the time of implementation.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvCurHopLimit.

    IANA: IP Parameters,
```

```
    http://www.iana.org/assignments/ip-parameters";
}
leaf default-lifetime {
    type uint16 {
        range "0..9000";
    }
    units "seconds";
    description
        "The value to be placed in the Router Lifetime field of
        Router Advertisements sent from the interface, in seconds.
        It MUST be either zero or between max-rtr-adv-interval and
        9000 seconds. A value of zero indicates that the router is
        not to be used as a default router. These limits may be
        overridden by specific documents that describe how IPv6
        operates over different link layers.

        If this parameter is not configured, the device SHOULD use
        a value of 3 * max-rtr-adv-interval.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvDefaultLifeTime.";
}
container prefix-list {
    description
        "Configuration of prefixes to be placed in Prefix
        Information options in Router Advertisement messages sent
        from the interface.

        Prefixes that are advertised by default but do not have
        their entries in the child 'prefix' list are advertised
        with the default values of all parameters.

        The link-local prefix SHOULD NOT be included in the list
        of advertised prefixes.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvPrefixList.";
    list prefix {
        key "prefix-spec";
        description
            "Configuration of an advertised prefix entry.";
        leaf prefix-spec {
            type inet:ipv6-prefix;
            description
                "IPv6 address prefix.";
        }
        choice control-adv-prefixes {
            default "advertise";
        }
    }
}
```

```
description
  "The prefix either may be explicitly removed from the
   set of advertised prefixes, or parameters with which
   it is advertised may be specified (default case).";
leaf no-advertise {
  type empty;
  description
    "The prefix will not be advertised.

    This can be used for removing the prefix from the
    default set of advertised prefixes.";
}
case advertise {
  leaf valid-lifetime {
    type uint32;
    units "seconds";
    default "2592000";
    description
      "The value to be placed in the Valid Lifetime in
       the Prefix Information option. The designated
       value of all 1's (0xffffffff) represents
       infinity.";
    reference
      "RFC 4861: Neighbor Discovery for IP version 6
       (IPv6) - AdvValidLifetime.";
  }
  leaf on-link-flag {
    type boolean;
    default "true";
    description
      "The value to be placed in the on-link flag
       ('L-bit') field in the Prefix Information
       option.";
    reference
      "RFC 4861: Neighbor Discovery for IP version 6
       (IPv6) - AdvOnLinkFlag.";
  }
  leaf preferred-lifetime {
    type uint32;
    units "seconds";
    must ". <= ../valid-lifetime" {
      description
        "This value MUST NOT be greater than
         valid-lifetime.";
    }
    default "604800";
    description
      "The value to be placed in the Preferred Lifetime
```



```
        in the Prefix Information option. The designated
        value of all 1's (0xffffffff) represents
        infinity.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6
        (IPv6) - AdvPreferredLifetime.";
    }
    leaf autonomous-flag {
        type boolean;
        default "true";
        description
            "The value to be placed in the Autonomous Flag
            field in the Prefix Information option.";
        reference
            "RFC 4861: Neighbor Discovery for IP version 6
            (IPv6) - AdvAutonomousFlag.";
    }
    }
    }
    }
    }
    }
    }
```

<CODE ENDS>

10. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URIs in the IETF XML registry [RFC3688]:

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

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing

Registrant Contact: The IESG.

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

This document registers the following YANG modules in the YANG Module Names registry [RFC6020]:

name: ietf-routing
namespace: urn:ietf:params:xml:ns:yang:ietf-routing
prefix: rt
reference: RFC XXXX

name: ietf-ipv4-unicast-routing
namespace: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
prefix: v4ur
reference: RFC XXXX

name: ietf-ipv6-unicast-routing
namespace: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
prefix: v6ur
reference: RFC XXXX

This document registers the following YANG submodule in the YANG Module Names registry [RFC6020]:

name: ietf-ipv6-router-advertisements
parent: ietf-ipv6-unicast-routing
reference: RFC XXXX

11. Security Considerations

Configuration and state data conforming to the core routing data model (defined in this document) are designed to be accessed via a management protocol with secure transport layer, such as NETCONF [RFC6241]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

A number of configuration data nodes defined in the YANG modules belonging to the core routing data model are writable/creatable/deletable (i.e., "config true" in YANG terms, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations to these data nodes, such as "edit-config" in NETCONF, can have negative effects on the network if the protocol operations are not properly protected.

The vulnerable "config true" parameters and subtrees are the following:

/routing/control-plane-protocols/control-plane-protocol: This list specifies the control plane protocols configured on a device.

/routing/ribs/rib: This list specifies the RIBs configured for the device.

Unauthorised access to any of these lists can adversely affect the routing subsystem of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations and other problems.

12. Acknowledgments

The authors wish to thank Nitin Bahadur, Martin Bjorklund, Dean Bogdanovic, Jeff Haas, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Stephane Litkowski, Thomas Morin, Tom Petch, Yingzhen Qu, Bruno Rijsman, Juergen Schoenwaelder, Phil Shafer, Dave Thaler, Yi Yang, Derek Man-Kit Yeung and Jeffrey Zhang for their helpful comments and suggestions.

13. References

13.1. 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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <<http://www.rfc-editor.org/info/rfc4861>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://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, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<http://www.rfc-editor.org/info/rfc7277>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

13.2. Informative References

- [RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <<http://www.rfc-editor.org/info/rfc6087>>.

- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <<http://www.rfc-editor.org/info/rfc7895>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.

Appendix A. The Complete Data Trees

This appendix presents the complete configuration and state data trees of the core routing data model. See Section 2.2 for an explanation of the symbols used. Data type of every leaf node is shown near the right end of the corresponding line.

A.1. Configuration Data

```

+--rw routing
  +--rw router-id?                yang:dotted-quad
  +--rw control-plane-protocols
    +--rw control-plane-protocol* [type name]
      +--rw type                  identityref
      +--rw name                  string
      +--rw description?          string
      +--rw static-routes
        +--rw v6ur:ipv6
          +--rw v6ur:route* [destination-prefix]
            +--rw v6ur:destination-prefix    inet:ipv6-prefix
            +--rw v6ur:description?          string
            +--rw v6ur:next-hop
              +--rw (v6ur:next-hop-options)
                +--:(v6ur:simple-next-hop)
                | +--rw v6ur:outgoing-interface?
                | +--rw v6ur:next-hop-address?
                +--:(v6ur:special-next-hop)
                | +--rw v6ur:special-next-hop?  enumeration
                +--:(v6ur:next-hop-list)
                +--rw v6ur:next-hop-list
                  +--rw v6ur:next-hop* [index]
                    +--rw v6ur:index            string
                    +--rw v6ur:outgoing-interface?
                    +--rw v6ur:next-hop-address?
          +--rw v4ur:ipv4
            +--rw v4ur:route* [destination-prefix]
              +--rw v4ur:destination-prefix    inet:ipv4-prefix
              +--rw v4ur:description?          string
              +--rw v4ur:next-hop
                +--rw (v4ur:next-hop-options)
                  +--:(v4ur:simple-next-hop)
                  | +--rw v4ur:outgoing-interface?
                  | +--rw v4ur:next-hop-address?
                  +--:(v4ur:special-next-hop)
                  | +--rw v4ur:special-next-hop?  enumeration
                  +--:(v4ur:next-hop-list)
                  +--rw v4ur:next-hop-list
                    +--rw v4ur:next-hop* [index]
                      +--rw v4ur:index            string
                      +--rw v4ur:outgoing-interface?
                      +--rw v4ur:next-hop-address?
        +--rw ribs
          +--rw rib* [name]
            +--rw name                string
            +--rw address-family?     identityref
            +--rw description?        string

```

A.2. State Data

```

+--ro routing-state
|   +--ro router-id?                yang:dotted-quad
|   +--ro interfaces
|   |   +--ro interface*    if:interface-state-ref
|   +--ro control-plane-protocols
|   |   +--ro control-plane-protocol* [type name]
|   |   |   +--ro type        identityref
|   |   |   +--ro name        string
|   +--ro ribs
|   |   +--ro rib* [name]
|   |   |   +--ro name                string
|   |   |   +--ro address-family      identityref
|   |   |   +--ro default-rib?        boolean {multiple-ribs}?
|   |   +--ro routes
|   |   |   +--ro route*
|   |   |   |   +--ro route-preference?        route-preference
|   |   |   |   +--ro next-hop
|   |   |   |   |   +--ro (next-hop-options)
|   |   |   |   |   |   +--:(simple-next-hop)
|   |   |   |   |   |   |   +--ro outgoing-interface?
|   |   |   |   |   |   |   +--ro v6ur:next-hop-address?
|   |   |   |   |   |   |   +--ro v4ur:next-hop-address?
|   |   |   |   |   |   +--:(special-next-hop)
|   |   |   |   |   |   |   +--ro special-next-hop?        enumeration
|   |   |   |   |   |   +--:(next-hop-list)
|   |   |   |   |   |   |   +--ro next-hop-list
|   |   |   |   |   |   |   |   +--ro next-hop*
|   |   |   |   |   |   |   |   |   +--ro outgoing-interface?
|   |   |   |   |   |   |   |   |   +--ro v6ur:address?
|   |   |   |   |   |   |   |   |   +--ro v4ur:address?
|   |   |   |   +--ro source-protocol        identityref
|   |   |   +--ro active?                    empty
|   |   |   +--ro last-updated?              yang:date-and-time
|   |   |   +--ro v6ur:destination-prefix?   inet:ipv6-prefix
|   |   |   +--ro v4ur:destination-prefix?   inet:ipv4-prefix
|   +--x active-route
|   |   +--w input
|   |   |   +--w v6ur:destination-address?   inet:ipv6-address
|   |   |   +--w v4ur:destination-address?   inet:ipv4-address
|   +--ro output
|   |   +--ro route
|   |   |   +--ro next-hop
|   |   |   |   +--ro (next-hop-options)
|   |   |   |   |   +--:(simple-next-hop)
|   |   |   |   |   |   +--ro outgoing-interface?
|   |   |   |   |   |   +--ro v6ur:next-hop-address?

```

```
| | +--ro v4ur:next-hop-address?
| | +--:(special-next-hop)
| | | +--ro special-next-hop? enumeration
| | +--:(next-hop-list)
| | | +--ro next-hop-list
| | | | +--ro next-hop*
| | | | | +--ro outgoing-interface?
| | | | | +--ro v6ur:next-hop-address?
| | | | | +--ro v4ur:next-hop-address?
+--ro source-protocol identityref
+--ro active? empty
+--ro last-updated? yang:date-and-time
+--ro v6ur:destination-prefix? inet:ipv6-prefix
+--ro v4ur:destination-prefix? inet:ipv4-prefix
```

Appendix B. Minimum Implementation

Some parts and options of the core routing model, such as user-defined RIBs, are intended only for advanced routers. This appendix gives basic non-normative guidelines for implementing a bare minimum of available functions. Such an implementation may be used for hosts or very simple routers.

A minimum implementation does not support the feature "multiple-ribs". This means that a single system-controlled RIB is available for each supported address family - IPv4, IPv6 or both. These RIBs are also the default RIBs. No user-controlled RIBs are allowed.

In addition to the mandatory instance of the "direct" pseudo-protocol, a minimum implementation should support configuring instance(s) of the "static" pseudo-protocol.

For hosts that are never intended to act as routers, the ability to turn on sending IPv6 router advertisements (Section 5.4) should be removed.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" control plane protocol instances.

Appendix C. Example: Adding a New Control Plane Protocol

This appendix demonstrates how the core routing data model can be extended to support a new control plane protocol. The YANG module "example-rip" shown below is intended as an illustration rather than a real definition of a data model for the RIP routing protocol. For the sake of brevity, this module does not obey all the guidelines specified in [RFC6087]. See also Section 5.3.2.


```
module example-rip {  
    yang-version "1.1";  
    namespace "http://example.com/rip";  
    prefix "rip";  
    import ietf-interfaces {  
        prefix "if";  
    }  
    import ietf-routing {  
        prefix "rt";  
    }  
    identity rip {  
        base rt:routing-protocol;  
        description  
            "Identity for the RIP routing protocol.";  
    }  
    typedef rip-metric {  
        type uint8 {  
            range "0..16";  
        }  
    }  
    grouping route-content {  
        description  
            "This grouping defines RIP-specific route attributes.";  
        leaf metric {  
            type rip-metric;  
        }  
        leaf tag {  
            type uint16;  
            default "0";  
            description  
                "This leaf may be used to carry additional info, e.g. AS  
                number.";  
        }  
    }  
    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {  
        when "derived-from-or-self(rt:source-protocol, 'rip:rip')" {  
            description  
                "This augment is only valid for a routes whose source  
                protocol is RIP.";  
        }  
    }  
}
```

```
    }
    description
      "RIP-specific route attributes.";
    uses route-content;
  }

  augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
    + "rt:output/rt:route" {
    description
      "RIP-specific route attributes in the output of 'active-route'
      RPC.";
    uses route-content;
  }

  augment "/rt:routing/rt:control-plane-protocols/"
    + "rt:control-plane-protocol" {
    when "derived-from-or-self(rt:type,'rip:rip')" {
      description
        "This augment is only valid for a routing protocol instance
        of type 'rip'.";
    }
    container rip {
      presence "RIP configuration";
      description
        "RIP instance configuration.";
      container interfaces {
        description
          "Per-interface RIP configuration.";
        list interface {
          key "name";
          description
            "RIP is enabled on interfaces that have an entry in this
            list, unless 'enabled' is set to 'false' for that
            entry.";
          leaf name {
            type if:interface-ref;
          }
          leaf enabled {
            type boolean;
            default "true";
          }
          leaf metric {
            type rip-metric;
            default "1";
          }
        }
      }
    }
    leaf update-interval {
```

```
        type uint8 {
            range "10..60";
        }
        units "seconds";
        default "30";
        description
            "Time interval between periodic updates.";
    }
}
}
```

Appendix D. Data Tree Example

This section contains an example instance data tree in the JSON encoding [RFC7951], containing both configuration and state data. The data conforms to a data model that is defined by the following YANG library specification [RFC7895]:

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "c2elf54169aa7f36e1a6e8d0865d441d3600f9c4",
    "module": [
      {
        "name": "ietf-routing",
        "revision": "2016-11-03",
        "feature": [
          "multiple-ribs",
          "router-id"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv4-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv6-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
        "conformance-type": "implement"
      }
    ]
  }
}
```

```
    "name": "ietf-interfaces",
    "revision": "2014-05-08",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-inet-types",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
    "revision": "2013-07-15",
    "conformance-type": "import"
  },
  {
    "name": "ietf-yang-types",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
    "revision": "2013-07-15",
    "conformance-type": "import"
  },
  {
    "name": "iana-if-type",
    "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
    "revision": "",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-ip",
    "revision": "2014-06-16",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
    "conformance-type": "implement"
  }
]
}
```

A simple network set-up as shown in Figure 3 is assumed: router "A" uses static default routes with the "ISP" router as the next-hop. IPv6 router advertisements are configured only on the "eth1" interface and disabled on the upstream "eth0" interface.

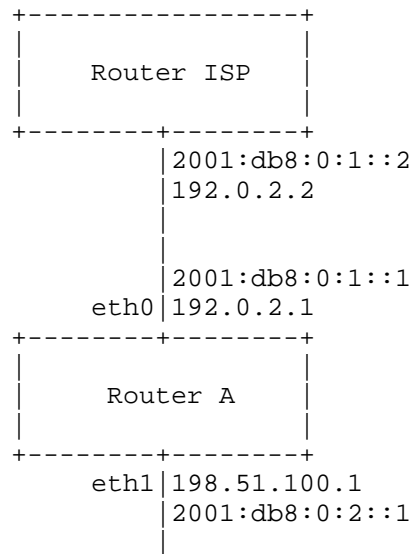


Figure 3: Example network configuration

The instance data tree could then be as follows:

```

{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.1",
              "prefix-length": 24
            }
          ]
        },
        "forwarding": true
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "198.51.100.1",
              "prefix-length": 24
            }
          ]
        },
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "2001:db8:0:2::1",
              "prefix-length": 64
            }
          ]
        },
        "forwarding": true
      }
    ]
  }
}
  
```

```
        "autoconf": {
          "create-global-addresses": false
        }
      },
    {
      "name": "eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "description": "Interface to the internal network.",
      "ietf-ip:ipv4": {
        "address": [
          {
            "ip": "198.51.100.1",
            "prefix-length": 24
          }
        ],
        "forwarding": true
      },
      "ietf-ip:ipv6": {
        "address": [
          {
            "ip": "2001:0db8:0:2::1",
            "prefix-length": 64
          }
        ],
        "forwarding": true,
        "autoconf": {
          "create-global-addresses": false
        },
        "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
          "send-advertisements": true
        }
      }
    }
  ]
},
"ietf-interfaces:interfaces-state": {
  "interface": [
    {
      "name": "eth0",
      "type": "iana-if-type:ethernetCsmacd",
      "phys-address": "00:0C:42:E5:B1:E9",
      "oper-status": "up",
      "statistics": {
        "discontinuity-time": "2015-10-24T17:11:27+02:00"
      },
      "ietf-ip:ipv4": {
        "forwarding": true,
```

```
        "mtu": 1500,
        "address": [
            {
                "ip": "192.0.2.1",
                "prefix-length": 24
            }
        ]
    },
    "ietf-ip:ipv6": {
        "forwarding": true,
        "mtu": 1500,
        "address": [
            {
                "ip": "2001:0db8:0:1::1",
                "prefix-length": 64
            }
        ],
        "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
            "send-advertisements": true,
            "prefix-list": {
                "prefix": [
                    {
                        "prefix-spec": "2001:db8:0:2::/64"
                    }
                ]
            }
        }
    }
},
{
    "name": "eth1",
    "type": "iana-if-type:ethernetCsmacd",
    "phys-address": "00:0C:42:E5:B1:EA",
    "oper-status": "up",
    "statistics": {
        "discontinuity-time": "2015-10-24T17:11:29+02:00"
    },
    "ietf-ip:ipv4": {
        "forwarding": true,
        "mtu": 1500,
        "address": [
            {
                "ip": "198.51.100.1",
                "prefix-length": 24
            }
        ]
    },
    "ietf-ip:ipv6": {
```

```
    "forwarding": true,
    "mtu": 1500,
    "address": [
      {
        "ip": "2001:0db8:0:2::1",
        "prefix-length": 64
      }
    ],
    "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
      "send-advertisements": true,
      "prefix-list": {
        "prefix": [
          {
            "prefix-spec": "2001:db8:0:2::/64"
          }
        ]
      }
    }
  }
}

],
"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:static",
        "name": "st0",
        "description":
          "Static routing is used for the internal network.",
        "static-routes": {
          "ietf-ipv4-unicast-routing:ipv4": {
            "route": [
              {
                "destination-prefix": "0.0.0.0/0",
                "next-hop": {
                  "next-hop-address": "192.0.2.2"
                }
              }
            ]
          }
        },
        "ietf-ipv6-unicast-routing:ipv6": {
          "route": [
            {
              "destination-prefix": "::/0",
              "next-hop": {
                "next-hop-address": "2001:db8:0:1::2"
              }
            }
          ]
        }
      }
    ]
  }
}
```



```

    }
  }
]
}
},
"ietf-routing:routing-state": {
  "interfaces": {
    "interface": [
      "eth0",
      "eth1"
    ]
  },
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:static",
        "name": "st0"
      }
    ]
  },
  "ribs": {
    "rib": [
      {
        "name": "ipv4-master",
        "address-family":
          "ietf-ipv4-unicast-routing:ipv4-unicast",
        "default-rib": true,
        "routes": {
          "route": [
            {
              "ietf-ipv4-unicast-routing:destination-prefix":
                "192.0.2.1/24",
              "next-hop": {
                "outgoing-interface": "eth0"
              },
              "route-preference": 0,
              "source-protocol": "ietf-routing:direct",
              "last-updated": "2015-10-24T17:11:27+02:00"
            },
            {
              "ietf-ipv4-unicast-routing:destination-prefix":
                "198.51.100.0/24",
              "next-hop": {
                "outgoing-interface": "eth1"
              }
            }
          ]
        }
      }
    ]
  }
}

```

```
    },
    "source-protocol": "ietf-routing:direct",
    "route-preference": 0,
    "last-updated": "2015-10-24T17:11:27+02:00"
  },
  {
    "ietf-ipv4-unicast-routing:destination-prefix":
      "0.0.0.0/0",
    "source-protocol": "ietf-routing:static",
    "route-preference": 5,
    "next-hop": {
      "ietf-ipv4-unicast-routing:next-hop-address":
        "192.0.2.2"
    },
    "last-updated": "2015-10-24T18:02:45+02:00"
  }
]
}
},
{
  "name": "ipv6-master",
  "address-family":
    "ietf-ipv6-unicast-routing:ipv6-unicast",
  "default-rib": true,
  "routes": {
    "route": [
      {
        "ietf-ipv6-unicast-routing:destination-prefix":
          "2001:db8:0:1::/64",
        "next-hop": {
          "outgoing-interface": "eth0"
        },
        "source-protocol": "ietf-routing:direct",
        "route-preference": 0,
        "last-updated": "2015-10-24T17:11:27+02:00"
      },
      {
        "ietf-ipv6-unicast-routing:destination-prefix":
          "2001:db8:0:2::/64",
        "next-hop": {
          "outgoing-interface": "eth1"
        },
        "source-protocol": "ietf-routing:direct",
        "route-preference": 0,
        "last-updated": "2015-10-24T17:11:27+02:00"
      },
      {
        "ietf-ipv6-unicast-routing:destination-prefix":
```

```
        "::/0",
        "next-hop": {
            "ietf-ipv6-unicast-routing:next-hop-address":
                "2001:db8:0:1::2"
        },
        "source-protocol": "ietf-routing:static",
        "route-preference": 5,
        "last-updated": "2015-10-24T18:02:45+02:00"
    }
}
]
}
}
}
```

Appendix E. Change Log

RFC Editor: Remove this section upon publication as an RFC.

E.1. Changes Between Versions -24 and -25

- o Minor edits based on IETF Last Call reviews.

E.2. Changes Between Versions -23 and -24

- o Fix paths in "when" expressions due to errata 4749 of RFC 7950.

E.3. Changes Between Versions -22 and -23

- o Removed "route-tag" feature.
- o Removed next-hop classifiers.
- o Fixed invalid when expressions in augments.
- o In simple-next-hop, an address, outgoing interface or both can be specified.
- o RPC "fib-route" changed into RIB action "active-route".
- o The requirement that direct routes be always placed in default RIBs.

E.4. Changes Between Versions -21 and -22

- o Added "next-hop-list" as a new case of the "next-hop-options" choice.
- o Renamed "routing protocol" to "control plane protocol" in both the YANG modules and I-D text.

E.5. Changes Between Versions -20 and -21

- o Routing instances were removed.
- o IPv6 RA parameters were moved to the "ietf-ipv6-router-advertisements".

E.6. Changes Between Versions -19 and -20

- o Assignment of L3 interfaces to routing instances is now part of interface configuration.
- o Next-hop options in configuration were aligned with state data.
- o It is recommended to enclose protocol-specific configuration in a presence container.

E.7. Changes Between Versions -18 and -19

- o The leaf "route-preference" was removed from the "routing-protocol" container in both "routing" and "routing-state".
- o The "vrf-routing-instance" identity was added in support of a common routing-instance type in addition to the "default-routing-instance".
- o Removed "enabled" switch from "routing-protocol".

E.8. Changes Between Versions -17 and -18

- o The container "ribs" was moved under "routing-instance" (in both "routing" and "routing-state").
- o Typedefs "rib-ref" and "rib-state-ref" were removed.
- o Removed "recipient-ribs" (both state and configuration).
- o Removed "connected-ribs" from "routing-protocol" (both state and configuration).

- o Configuration and state data for IPv6 RA were moved under "if:interface" and "if:interface-state".
- o Assignment of interfaces to routing instances now use leaf-list rather than list (both config and state). The opposite reference from "if:interface" to "rt:routing-instance" was changed to a single leaf (an interface cannot belong to multiple routing instances).
- o Specification of a default RIB is now a simple flag under "rib" (both config and state).
- o Default RIBs are marked by a flag in state data.

E.9. Changes Between Versions -16 and -17

- o Added Acee as a co-author.
- o Removed all traces of route filters.
- o Removed numeric IDs of list entries in state data.
- o Removed all next-hop cases except "simple-next-hop" and "special-next-hop".
- o Removed feature "multipath-routes".
- o Augmented "ietf-interfaces" module with a leaf-list of leafrefs pointing from state data of an interface entry to the routing instance(s) to which the interface is assigned.

E.10. Changes Between Versions -15 and -16

- o Added 'type' as the second key component of 'routing-protocol', both in configuration and state data.
- o The restriction of no more than one connected RIB per address family was removed.
- o Removed the 'id' key of routes in RIBs. This list has no keys anymore.
- o Remove the 'id' key from static routes and make 'destination-prefix' the only key.
- o Added 'route-preference' as a new attribute of routes in RIB.
- o Added 'active' as a new attribute of routes in RIBs.

- o Renamed RPC operation 'active-route' to 'fib-route'.
- o Added 'route-preference' as a new parameter of routing protocol instances, both in configuration and state data.
- o Renamed identity 'rt:standard-routing-instance' to 'rt:default-routing-instance'.
- o Added next-hop lists to state data.
- o Added two cases for specifying next-hops indirectly - via a new RIB or a recursive list of next-hops.
- o Reorganized next-hop in static routes.
- o Removed all 'if-feature' statements from state data.

E.11. Changes Between Versions -14 and -15

- o Removed all defaults from state data.
- o Removed default from 'cur-hop-limit' in config.

E.12. Changes Between Versions -13 and -14

- o Removed dependency of 'connected-ribs' on the 'multiple-ribs' feature.
- o Removed default value of 'cur-hop-limit' in state data.
- o Moved parts of descriptions and all references on IPv6 RA parameters from state data to configuration.
- o Added reference to RFC 6536 in the Security section.

E.13. Changes Between Versions -12 and -13

- o Wrote appendix about minimum implementation.
- o Remove "when" statement for IPv6 router interface state data - it was dependent on a config value that may not be present.
- o Extra container for the next-hop list.
- o Names rather than numeric ids are used for referring to list entries in state data.

- o Numeric ids are always declared as mandatory and unique. Their description states that they are ephemeral.
- o Descriptions of "name" keys in state data lists are required to be persistent.
- o
- o Removed "if-feature multiple-ribs;" from connected-ribs.
- o "rib-name" instead of "name" is used as the name of leafref nodes.
- o "next-hop" instead of "nexthop" or "gateway" used throughout, both in node names and text.

E.14. Changes Between Versions -11 and -12

- o Removed feature "advanced-router" and introduced two features instead: "multiple-ribs" and "multipath-routes".
- o Unified the keys of config and state versions of "routing-instance" and "rib" lists.
- o Numerical identifiers of state list entries are not keys anymore, but they are constrained using the "unique" statement.
- o Updated acknowledgements.

E.15. Changes Between Versions -10 and -11

- o Migrated address families from IANA enumerations to identities.
- o Terminology and node names aligned with the I2RS RIB model: router -> routing instance, routing table -> RIB.
- o Introduced uint64 keys for state lists: routing-instance, rib, route, nexthop.
- o Described the relationship between system-controlled and user-controlled list entries.
- o Feature "user-defined-routing-tables" changed into "advanced-router".
- o Made nexthop into a choice in order to allow for nexthop-list (I2RS requirement).

- o Added nexthop-list with entries having priorities (backup) and weights (load balancing).
- o Updated bibliography references.

E.16. Changes Between Versions -09 and -10

- o Added subtree for state data ("/routing-state").
- o Terms "system-controlled entry" and "user-controlled entry" defined and used.
- o New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.
- o Added grouping "router-id".
- o In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".
- o Renamed "main-routing-table" to "default-routing-table".

E.17. Changes Between Versions -08 and -09

- o Fixed "must" expression for "connected-routing-table".
- o Simplified "must" expression for "main-routing-table".
- o Moved per-interface configuration of a new routing protocol under 'routing-protocol'. This also affects the 'example-rip' module.

E.18. Changes Between Versions -07 and -08

- o Changed reference from RFC6021 to RFC6021bis.

E.19. Changes Between Versions -06 and -07

- o The contents of <get-reply> in Appendix D was updated: "eth[01]" is used as the value of "location", and "forwarding" is on for both interfaces and both IPv4 and IPv6.
- o The "must" expression for "main-routing-table" was modified to avoid redundant error messages reporting address family mismatch when "name" points to a non-existent routing table.
- o The default behavior for IPv6 RA prefix advertisements was clarified.

- o Changed type of "rt:router-id" to "ip:dotted-quad".
- o Type of "rt:router-id" changed to "yang:dotted-quad".
- o Fixed missing prefixes in XPath expressions.

E.20. Changes Between Versions -05 and -06

- o Document title changed: "Configuration" was replaced by "Management".
- o New typedefs "routing-table-ref" and "route-filter-ref".
- o Double slashes "//" were removed from XPath expressions and replaced with the single "/".
- o Removed uniqueness requirement for "router-id".
- o Complete data tree is now in Appendix A.
- o Changed type of "source-protocol" from "leafref" to "string".
- o Clarified the relationship between routing protocol instances and connected routing tables.
- o Added a must constraint saying that a routing table connected to the direct pseudo-protocol must not be a main routing table.

E.21. Changes Between Versions -04 and -05

- o Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".
- o "must" statement for "static-routes" changed to "when".
- o Added "main-routing-tables" containing references to main routing tables for each address family.
- o Removed the defaults for "address-family" and "safi" and made them mandatory.
- o Removed the default for route-filter/type and made this leaf mandatory.
- o If there is no active route for a given destination, the "active-route" RPC returns no output.
- o Added "enabled" switch under "routing-protocol".

- o Added "router-type" identity and "type" leaf under "router".
- o Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".
- o The "direct" pseudo-protocol is always connected to main routing tables.
- o Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".
- o Added "must" constraint saying that a routing table must not be its own recipient.

E.22. Changes Between Versions -03 and -04

- o Changed "error-tag" for both RPC operations from "missing element" to "data-missing".
- o Removed the decrementing behavior for advertised IPv6 prefix parameters "valid-lifetime" and "preferred-lifetime".
- o Changed the key of the static route lists from "seqno" to "id" because the routes needn't be sorted.
- o Added 'must' constraint saying that "preferred-lifetime" must not be greater than "valid-lifetime".

E.23. Changes Between Versions -02 and -03

- o Module "iana-afn-safi" moved to I-D "iana-if-type".
- o Removed forwarding table.
- o RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).
- o New RPC "route-count".
- o For both RPCs, specification of negative responses was added.
- o Relaxed separation of router instances.
- o Assignment of interfaces to router instances needn't be disjoint.
- o Route filters are now global.
- o Added "allow-all-route-filter" for symmetry.

- o Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- o Added "router-id" leaf.
- o Specified the names for IPv4/IPv6 unicast main routing tables.
- o Route parameter "last-modified" changed to "age".
- o Added container "recipient-routing-tables".

E.24. Changes Between Versions -01 and -02

- o Added module "ietf-ipv6-unicast-routing".
- o The example in Appendix D now uses IP addresses from blocks reserved for documentation.
- o Direct routes appear by default in the forwarding table.
- o Network layer interfaces must be assigned to a router instance. Additional interface configuration may be present.
- o The "when" statement is only used with "augment", "must" is used elsewhere.
- o Additional "must" statements were added.
- o The "route-content" grouping for IPv4 and IPv6 unicast now includes the material from the "ietf-routing" version via "uses rt:route-content".
- o Explanation of symbols in the tree representation of data model hierarchy.

E.25. Changes Between Versions -00 and -01

- o AFN/SAFI-independent stuff was moved to the "ietf-routing" module.
- o Typedefs for AFN and SAFI were placed in a separate "iana-afn-safi" module.
- o Names of some data nodes were changed, in particular "routing-process" is now "router".
- o The restriction of a single AFN/SAFI per router was lifted.
- o RPC operation "delete-route" was removed.

- o Illegal XPath references from "get-route" to the datastore were fixed.
- o Section "Security Considerations" was written.

Authors' Addresses

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Acee Lindem
Cisco Systems

Email: acee@cisco.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: 7 October 2022

J. Clarke, Ed.
Cisco
M. Jethanandani, Ed.
Kloud Services
C. Wildes, Ed.
Cisco Systems Inc.
K. Koushik, Ed.
Verizon Wireless
5 April 2022

A YANG Data Model for Syslog Configuration
draft-ietf-netmod-syslog-model-27

Abstract

This document defines a YANG data model for the configuration of a syslog process. It is intended this model be used by vendors who implement syslog in their systems.

Status of This Memo

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

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

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

This Internet-Draft will expire on 7 October 2022.

Copyright Notice

Copyright (c) 2022 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	2
1.1. Requirements Language	3
2. Terminology	3
3. NDMA Compliance	3
4. Editorial Note (To be removed by RFC Editor)	4
5. Design of the Syslog Model	4
5.1. Syslog Module	6
6. Syslog YANG Module	14
6.1. The ietf-syslog Module	14
7. Usage Examples	32
7.1. Syslog Configuration for Severity Critical	32
7.2. Remote Syslog Configuration	33
8. Acknowledgements	34
9. IANA Considerations	34
9.1. The IETF XML Registry	34
9.2. The YANG Module Names Registry	35
10. Security Considerations	35
11. References	36
11.1. Normative References	36
11.2. Informative References	37
Appendix A. Implementer Guidelines	38
A.1. Extending Facilities	38
A.2. Syslog Terminal Output	39
A.3. Syslog File Naming Convention	40
Authors' Addresses	40

1. Introduction

This document defines a YANG [RFC7950] configuration data model that may be used to configure the syslog feature running on a system. YANG models can be used with network management protocols such as NETCONF [RFC6241] to install, manipulate, and delete the configuration of network devices.

The data model makes use of the YANG "feature" construct which allows implementations to support only those syslog features that lie within their capabilities.

This module can be used to configure the syslog application conceptual layers as implemented on the target system.

Essentially, a syslog process receives messages (from the kernel, processes, applications or other syslog processes) and processes them. The processing may involve logging to a local file, and/or displaying on console, and/or relaying to syslog processes on other machines. The processing is determined by the "facility" that originated the message and the "severity" assigned to the message by the facility.

Such definitions of syslog protocol are defined in [RFC5424], and are used in this RFC.

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

1.1. Requirements Language

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. Terminology

The term "originator" is defined in [RFC5424]: an "originator" generates syslog content to be carried in a message.

The term "relay" is defined in [RFC5424]: a "relay" forwards messages, accepting messages from originators or other relays and sending them to collectors or other relays

The term "collectors" is defined in [RFC5424]: a "collector" gathers syslog content for further analysis.

The term "action" refers to the processing that takes place for each syslog message received.

3. NDMA Compliance

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

4. Editorial Note (To be removed by RFC Editor)

This document contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- * I-D.ietf-netconf-crypto-types --> the assigned RFC value for draft-ietf-netconf-crypto-types
- * I-D.ietf-netconf-tls-client-server --> the assigned RFC value for draft-ietf-netconf-tls-client-server
- * zzzz --> the assigned RFC value for this draft

5. Design of the Syslog Model

The syslog model was designed by comparing various syslog features implemented by various vendors' in different implementations.

This document addresses the common leafs between implementations and creates a common model, which can be augmented with proprietary features, if necessary. This model is designed to be very simple for maximum flexibility.

Some optional features are defined in this document to specify functionality that is present in specific vendor configurations.

Syslog consists of originators and collectors. The following diagram shows syslog messages flowing from originators, to collectors where filtering can take place.

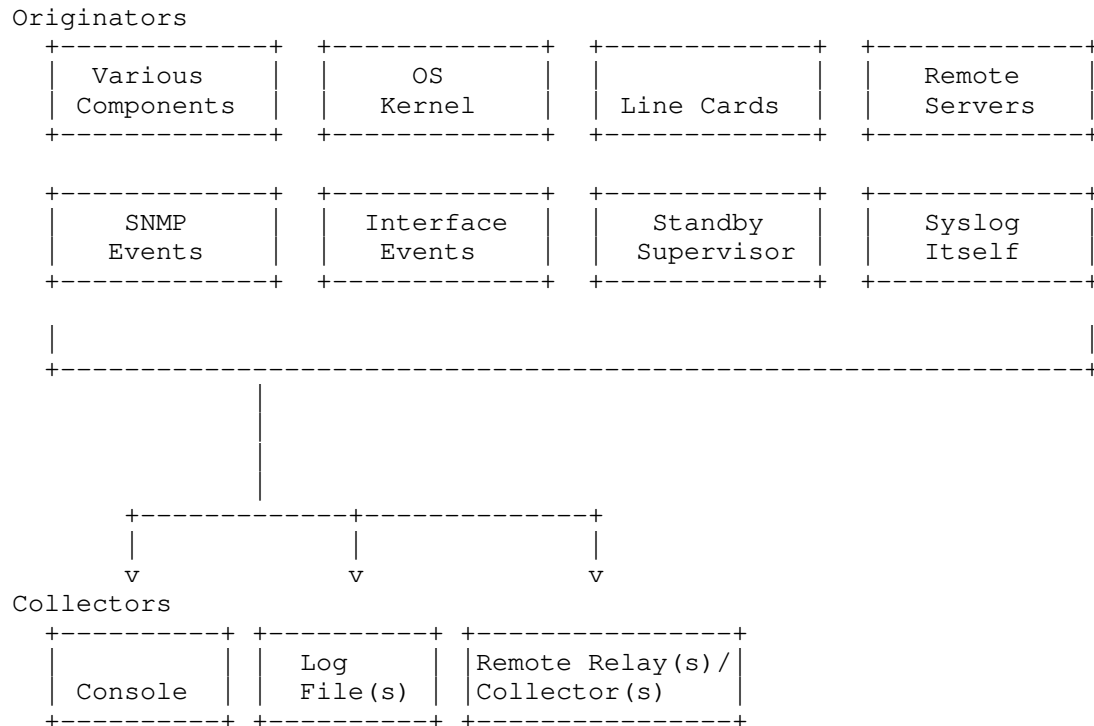


Figure 1. Syslog Processing Flow

Collectors are configured using the leaves in the syslog model "actions" container which correspond to each message collector:

```
console
```

```
log file(s)
```

```
remote relay(s)/collector(s)
```

Within each action, a selector is used to filter syslog messages. A selector consists of a list of one or more filters specified by facility-severity pairs, and, if supported via the select-match feature, an optional regular expression pattern match that is performed on the [RFC5424] field.

A syslog message is processed if:

There is an element of facility-list (F, S) where
 the message facility matches F
 and the message severity matches S
 and/or the message text matches the regex pattern (if it
 is present)

The facility is one of a specific syslog-facility, or all facilities.

The severity is one of type syslog-severity, all severities, or none. None is a special case that can be used to disable a filter. When filtering severity, the default comparison is that messages of the specified severity and higher are selected to be logged. This is shown in the model as "default equals-or-higher". This behavior can be altered if the select-adv-compare feature is enabled to specify a compare operation and an action. Compare operations are: "equals" to select messages with this single severity, or "equals-or-higher" to select messages of the specified severity and higher. Actions are used to log the message or block the message from being logged.

Many vendors extend the list of facilities available for logging in their implementation. An example is included in Extending Facilities (Appendix A.1).

5.1. Syslog Module

A simplified graphical representation of the data model is used in this document. Please see [RFC8340] for tree diagram notation.

```

module: ietf-syslog
+--rw syslog!
  +--rw actions
    +--rw console! {console-action}?
      +--rw facility-filter
        +--rw facility-list* [facility severity]
          +--rw facility          union
          +--rw severity          union
          +--rw advanced-compare {select-adv-compare}?
            +--rw compare?      enumeration
            +--rw action?       enumeration
        +--rw pattern-match?    string {select-match}?
    +--rw file {file-action}?
      +--rw log-file* [name]
        +--rw name              inet:uri
      +--rw facility-filter
        +--rw facility-list* [facility severity]
          +--rw facility          union
          +--rw severity          union
          +--rw advanced-compare {select-adv-compare}?

```

```

+--rw compare?          enumeration
+--rw action?           enumeration
+--rw pattern-match?    string {select-match}?
+--rw structured-data?  boolean {structured-data}?
+--rw file-rotation
+--rw number-of-files?  uint32 {file-limit-size}?
+--rw max-file-size?    uint32 {file-limit-size}?
+--rw rollover?         uint32
+--rw retention?        uint32
+--rw retention?        {file-limit-duration}?
+--rw retention?        {file-limit-duration}?
+--rw remote {remote-action}?
+--rw destination* [name]
+--rw name              string
+--rw (transport)
+--: (udp)
+--rw udp
+--rw address?         inet:host
+--rw port?            inet:port-number
+--: (tls)
+--rw tls
+--rw address?         inet:host
+--rw port?            inet:port-number
+--rw client-identity!
+--rw (auth-type)
+--: (certificate)
+--rw certificate
+--rw (local-or-keystore)
+--: (local)
+--rw local-definition
+--rw public-key-format
+--rw public-key
+--rw private-key-format?
+--rw (private-key-type)
+--rw cleartext-private-key
+--rw cleartext-private-key
+--rw (hidden-private-key)
+--rw (hidden-keys)?

```

key?							+++rw hidden-private-
							empty
ey)							+++:(encrypted-private-k
							{private-key-en
ryption)?							
te-key							+++rw encrypted-priva
							+++rw encrypted-by
lue-format							+++rw encrypted-va
f							identityre
lue							+++rw encrypted-va
							binary
							+++rw cert-data?
							end-entity-cert-cms
n							++++n certificate-expiratio
							{certificate-expira
tion-notification)?							
							+++ expiration-date
me							yang:date-and-ti
signing-request							++++x generate-certificate-
							{certificate-signin
g-request-generation)?							
							++++w input
							++++w csr-info
							ct:csr-info
							+++ro output
ning-request							+++ro certificate-sig
							ct:csr
							+++:(keystore)
ted,asymmetric-keys)?							{central-keystore-suppor
							+++rw keystore-reference
							+++rw asymmetric-key?
ef							ks:asymmetric-key-r
							{central-keystore-s
upported,asymmetric-keys)?							
fref							+++rw certificate? lea

				<pre> +---:(raw-public-key) {client-ident-raw-public-key}? +---rw raw-private-key +---rw (local-or-keystore) +---:(local) {local-definitions-suppo </pre>
rted, asymmetric-keys)?				<pre> +---rw local-definition +---rw public-key-format identityref +---rw public-key binary +---rw private-key-format? identityref +---rw (private-key-type) +---:(cleartext-private-k </pre>
ey)				
te-key?				<pre> +---rw cleartext-priv </pre>
				<pre> binary +---:(hidden-private-key) {hidden-keys}? +---rw hidden-private- </pre>
key?				
				<pre> empty +---:(encrypted-private-k </pre>
ey)				
ryption)?				<pre> {private-key-en </pre>
te-key				<pre> +---rw encrypted-priv </pre>
				<pre> +---rw encrypted-by +---rw encrypted-va </pre>
lue-format				<pre> identityre </pre>
f				<pre> +---rw encrypted-va </pre>
lue				<pre> binary +---:(keystore) {central-keystore-suppor </pre>
ted, asymmetric-keys)?				<pre> +---rw keystore-reference? ks:asymmetric-key-ref +---:(tls12-psk) {client-ident-tls12-psk}? +---rw tls12-psk +---rw (local-or-keystore) </pre>

					<pre> +---:(local) {local-definitions-suppo </pre>
rted, symmetric-keys}?					
					<pre> +---rw local-definition +---rw key-format? identityref +---rw (key-type) +---:(cleartext-key) +---rw cleartext-key? binary +---:(hidden-key) {hidden-keys}? +---rw hidden-key? empty +---:(encrypted-key) {symmetric-key- </pre>
encryption}?					<pre> +---rw encrypted-key +---rw encrypted-by +---rw encrypted-va </pre>
lue-format					
f					<pre> identityre </pre>
lue					<pre> +---rw encrypted-va </pre>
					<pre> binary +---:(keystore) {central-keystore-suppor </pre>
ted, symmetric-keys}?					<pre> +---rw keystore-reference? ks:symmetric-key-ref +---rw id? string +---:(tls13-epsk) {client-ident-tls13-epsk}? +---rw tls13-epsk +---rw (local-or-keystore) +---:(local) {local-definitions-suppo </pre>
rted, symmetric-keys}?					<pre> +---rw local-definition +---rw key-format? identityref +---rw (key-type) +---:(cleartext-key) +---rw cleartext-key? binary +---:(hidden-key) </pre>

					{hidden-keys}? +--rw hidden-key? empty +--:(encrypted-key) {symmetric-key-
encryption)?					+--rw encrypted-key +--rw encrypted-by +--rw encrypted-va
lue-format					identityre
f					+--rw encrypted-va
lue					binary +--:(keystore) {central-keystore-suppore
ted,symmetric-keys)?					+--rw keystore-reference? ks:symmetric-key-ref +--rw external-identity string +--rw hash tlscmn:epsk-supported-hash +--rw context? string +--rw target-protocol? uint16 +--rw target-kdf? uint16 +--rw server-authentication +--rw ca-certs! {server-auth-x509-cert}? +--rw (local-or-truststore) +--:(local) {local-definitions-supported}? +--rw local-definition +--rw certificate* [name] +--rw name string +--rw cert-data trust-anchor-cert-cms +---n certificate-expiration {certificate-expiratio
n-notification)?					+-- expiration-date yang:date-and-time +--:(truststore) {central-truststore-supported

certificates}?				<pre> +---rw truststore-reference? ts:certificate-bag-ref +---rw ee-certs! {server-auth-x509-cert}? +---rw (local-or-truststore) +---:(local) {local-definitions-supported}? +---rw local-definition +---rw certificate* [name] +---rw name string +---rw cert-data trust-anchor-cert-cms +---n certificate-expiration {certificate-expiratio </pre>
n-notification}?				<pre> +--- expiration-date yang:date-and-time +---:(truststore) {central-truststore-supported, </pre>
certificates}?				<pre> +---rw truststore-reference? ts:certificate-bag-ref +---rw raw-public-keys! {server-auth-raw-public-key}? +---rw (local-or-truststore) +---:(local) {local-definitions-supported}? +---rw local-definition +---rw public-key* [name] +---rw name string +---rw public-key-format identityref +---rw public-key binary +---:(truststore) {central-truststore-supported, </pre>
public-keys}?				<pre> +---rw truststore-reference? ts:public-key-bag-ref +---rw tls12-psks? {server-auth-tls12-psk}? +---rw tls13-epsks? {server-auth-tls13-epsk}? +---rw hello-params {tlscmn:hello-params}? +---rw tls-versions +---rw tls-version* identityref </pre>


```

    |         +---rw cipher-suites
    |         |         +---rw cipher-suite*  identityref
    +---rw keepalives {tls-client-keepalives}?
    |         +---rw peer-allowed-to-send?  empty
    |         +---rw test-peer-aliveness!
    |         |         +---rw max-wait?      uint16
    |         |         +---rw max-attempts?  uint8
+---rw facility-filter
|   +---rw facility-list* [facility severity]
|   |   +---rw facility      union
|   |   +---rw severity     union
|   +---rw advanced-compare {select-adv-compare}?
|   |   +---rw compare?     enumeration
|   |   +---rw action?      enumeration
+---rw pattern-match?      string {select-match}?
+---rw structured-data?    boolean {structured-data}?
+---rw facility-override?  identityref
+---rw source-interface?   if:interface-ref
|   {remote-source-interface}?
+---rw signing! {signed-messages}?
+---rw cert-signers
|   +---rw cert-signer* [name]
|   |   +---rw name          string
|   |   +---rw cert
|   |   |   +---rw public-key-format
|   |   |   |   identityref
|   |   |   +---rw public-key
|   |   |   |   binary
|   |   |   +---rw private-key-format?
|   |   |   |   identityref
|   |   |   +---rw (private-key-type)
|   |   |   |   +---:(cleartext-private-key)
|   |   |   |   |   +---rw cleartext-private-key?
|   |   |   |   |   |   binary
|   |   |   |   +---:(hidden-private-key) {hidden-keys}?
|   |   |   |   |   +---rw hidden-private-key?
|   |   |   |   |   |   empty
|   |   |   |   +---:(encrypted-private-key)
|   |   |   |   |   {private-key-encryption}?
|   |   |   |   |   +---rw encrypted-private-key
|   |   |   |   |   |   +---rw encrypted-by
|   |   |   |   |   |   |   +---rw encrypted-value-format
|   |   |   |   |   |   |   |   identityref
|   |   |   |   |   |   |   +---rw encrypted-value
|   |   |   |   |   |   |   |   binary
|   |   |   +---rw certificates
|   |   |   |   +---rw certificate* [name]
|   |   |   |   |   +---rw name

```



Figure 1: Tree Diagram for Syslog Model

6. Syslog YANG Module

6.1. The ietf-syslog Module

This module imports typedefs from [RFC6991], [RFC8343], groupings from [I-D.ietf-netconf-crypto-types], and [I-D.ietf-netconf-tls-client-server], and it references [RFC5424], [RFC5425], [RFC5426], and [RFC5848], [RFC8089], [RFC8174], and [Std-1003.1-2008].

```

<CODE BEGINS> file "ietf-syslog@2022-04-05.yang"
module ietf-syslog {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-syslog";
  prefix syslog;

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

```

```
import ietf-interfaces {
  prefix if;
  reference
    "RFC 8343: A YANG Data Model for Interface Management";
}
import ietf-tls-client {
  prefix tlsc;
  reference
    "I-D.ietf-netconf-tls-client-server:
    YANG Groupings for TLS Clients and TLS Servers";
}
import ietf-crypto-types {
  prefix ct;
  reference
    "I-D.ietf-netconf-crypto-types: YANG Data Types for
    Cryptography";
}

organization
  "IETF NETMOD (Network Modeling) Working Group";
contact
  "WG Web:  <https://datatracker.ietf.org/wg/netmod/>
  WG List:  <mailto:netmod@ietf.org>

  Editor:    Mahesh Jethanandani
             <mailto:mjethanandani@gmail.com>

  Editor:    Joe Clarke
             <mailto:jclarke@cisco.com>

  Editor:    Kiran Agrahara Sreenivasa
             <mailto:kirankoushik.agraharasreenivasa@
             verizonwireless.com>

  Editor:    Clyde Wildes
             <mailto:cwildes@cisco.com>";
description
  "This module contains a collection of YANG definitions
  for syslog configuration.

  Copyright (c) 2022 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>).

This version of this YANG module is part of RFC zzzz (<https://www.rfc-editor.org/info/rfczzzz>); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision 2022-04-05 {
  description
    "Initial Revision";
  reference
    "RFC zzzz: Syslog YANG Model";
}

feature console-action {
  description
    "This feature indicates that the local console action is
    supported.";
}

feature file-action {
  description
    "This feature indicates that the local file action is
    supported.";
}

feature file-limit-size {
  description
    "This feature indicates that file logging resources
    are managed using size and number limits.";
}

feature file-limit-duration {
  description
    "This feature indicates that file logging resources
    are managed using time based limits.";
}

feature remote-action {
  description
    "This feature indicates that the remote server action is
    supported.";
}
```

```
feature remote-source-interface {
  description
    "This feature indicates that source-interface is supported
    supported for the remote-action.";
}

feature select-adv-compare {
  description
    "This feature represents the ability to select messages
    using the additional comparison operators when comparing
    the syslog message severity.";
}

feature select-match {
  description
    "This feature represents the ability to select messages
    based on a Posix 1003.2 regular expression pattern match.";
}

feature structured-data {
  description
    "This feature represents the ability to log messages
    in structured-data format.";
  reference
    "RFC 5424: The Syslog Protocol";
}

feature signed-messages {
  description
    "This feature represents the ability to configure signed
    syslog messages.";
  reference
    "RFC 5848: Signed Syslog Messages";
}

typedef syslog-severity {
  type enumeration {
    enum emergency {
      value 0;
      description
        "The severity level 'Emergency' indicating that the
        system is unusable.";
    }
    enum alert {
      value 1;
      description
        "The severity level 'Alert' indicating that an action
        must be taken immediately.";
    }
  }
}
```

```
    }
    enum critical {
        value 2;
        description
            "The severity level 'Critical' indicating a critical
            condition.";
    }
    enum error {
        value 3;
        description
            "The severity level 'Error' indicating an error
            condition.";
    }
    enum warning {
        value 4;
        description
            "The severity level 'Warning' indicating a warning
            condition.";
    }
    enum notice {
        value 5;
        description
            "The severity level 'Notice' indicating a normal but
            significant condition.";
    }
    enum info {
        value 6;
        description
            "The severity level 'Info' indicating an informational
            message.";
    }
    enum debug {
        value 7;
        description
            "The severity level 'Debug' indicating a debug-level
            message.";
    }
    }
    description
        "The definitions for Syslog message severity.
        Note that a lower value is a higher severity. Comparisons of
        equal-or-higher severity mean equal or lower numeric value";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity syslog-facility {
    description
```

```
        "This identity is used as a base for all syslog facilities.";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity kern {
    base syslog-facility;
    description
        "The facility for kernel messages (0).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity user {
    base syslog-facility;
    description
        "The facility for user-level messages (1).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity mail {
    base syslog-facility;
    description
        "The facility for the mail system (2).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity daemon {
    base syslog-facility;
    description
        "The facility for the system daemons (3).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity auth {
    base syslog-facility;
    description
        "The facility for security/authorization messages (4).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity syslog {
    base syslog-facility;
    description
```

```
        "The facility for messages generated internally by syslogd
        facility (5).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity lpr {
    base syslog-facility;
    description
        "The facility for the line printer subsystem (6).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity news {
    base syslog-facility;
    description
        "The facility for the network news subsystem (7).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity uucp {
    base syslog-facility;
    description
        "The facility for the UUCP subsystem (8).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity cron {
    base syslog-facility;
    description
        "The facility for the clock daemon (9).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity authpriv {
    base syslog-facility;
    description
        "The facility for privileged security/authorization messages
        (10).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity ftp {
```



```
    base syslog-facility;
    description
        "The facility for the FTP daemon (11).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity ntp {
    base syslog-facility;
    description
        "The facility for the NTP subsystem (12).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity audit {
    base syslog-facility;
    description
        "The facility for log audit messages (13).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity console {
    base syslog-facility;
    description
        "The facility for log alert messages (14).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity cron2 {
    base syslog-facility;
    description
        "The facility for the second clock daemon (15).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local0 {
    base syslog-facility;
    description
        "The facility for local use 0 messages (16).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local1 {
```

```
    base syslog-facility;
    description
        "The facility for local use 1 messages (17).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local2 {
    base syslog-facility;
    description
        "The facility for local use 2 messages (18).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local3 {
    base syslog-facility;
    description
        "The facility for local use 3 messages (19).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local4 {
    base syslog-facility;
    description
        "The facility for local use 4 messages (20).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local5 {
    base syslog-facility;
    description
        "The facility for local use 5 messages (21).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local6 {
    base syslog-facility;
    description
        "The facility for local use 6 messages (22).";
    reference
        "RFC 5424: The Syslog Protocol";
}

identity local7 {
```

```
base syslog-facility;
description
  "The facility for local use 7 messages (23).";
reference
  "RFC 5424: The Syslog Protocol";
}

grouping severity-filter {
  description
    "This grouping defines the processing used to select
    log messages by comparing syslog message severity using
    the following processing rules:
    - if 'none', do not match.
    - if 'all', match.
    - else compare message severity with the specified severity
    according to the default compare rule (all messages of the
    specified severity and greater match) or if the
    select-adv-compare feature is present, use the
    advance-compare rule.";
  leaf severity {
    type union {
      type syslog-severity;
      type enumeration {
        enum none {
          value 2147483647;
          description
            "This enum describes the case where no severities
            are selected.";
        }
        enum all {
          value -2147483648;
          description
            "This enum describes the case where all severities
            are selected.";
        }
      }
    }
    mandatory true;
    description
      "This leaf specifies the syslog message severity.";
  }
  container advanced-compare {
    when "../severity != \"all\" and
    ../severity != \"none\"" {
      description
        "The advanced compare container is not applicable for
        severity 'all' or severity 'none'";
    }
  }
}
```

```
if-feature "select-adv-compare";
leaf compare {
  type enumeration {
    enum equals {
      description
        "This enum specifies that the severity comparison
        operation will be equals.";
    }
    enum equals-or-higher {
      description
        "This enum specifies that the severity comparison
        operation will be equals or higher.";
    }
  }
  default "equals-or-higher";
  description
    "The compare can be used to specify the comparison
    operator that should be used to compare the syslog message
    severity with the specified severity.";
}
leaf action {
  type enumeration {
    enum log {
      description
        "This enum specifies that if the compare operation is
        true the message will be logged.";
    }
    enum block {
      description
        "This enum specifies that if the compare operation is
        true the message will not be logged.";
    }
  }
  default "log";
  description
    "The action can be used to specify if the message should
    be logged or blocked based on the outcome of the compare
    operation.";
}
description
  "This container describes additional severity compare
  operations that can be used in place of the default
  severity comparison. The compare leaf specifies the type of
  the compare that is done and the action leaf specifies the
  intended result.
  Example: compare->equals and action->block means
  messages that have a severity that are equal to the
  specified severity will not be logged.";
```

```
    }  
  }  
  
  grouping selector {  
    description  
      "This grouping defines a syslog selector which is used to  
      select log messages for the log-actions (console, file,  
      remote, etc.). Choose one or both of the following:  
        facility [<facility> <severity>...]  
        pattern-match regular-expression-match-string  
      If both facility and pattern-match are specified, both must  
      match in order for a log message to be selected.";  
    container facility-filter {  
      description  
        "This container describes the syslog filter parameters.";  
      list facility-list {  
        key "facility severity";  
        ordered-by user;  
        description  
          "This list describes a collection of syslog  
          facilities and severities.";  
        leaf facility {  
          type union {  
            type identityref {  
              base syslog-facility;  
            }  
            type enumeration {  
              enum all {  
                description  
                  "This enum describes the case where all  
                  facilities are requested.";  
              }  
            }  
          }  
          description  
            "The leaf uniquely identifies a syslog facility.";  
        }  
        uses severity-filter;  
      }  
    }  
    leaf pattern-match {  
      if-feature "select-match";  
      type string;  
      description  
        "This leaf describes a Posix 1003.2 regular expression  
        string that can be used to select a syslog message for  
        logging. The match is performed on the SYSLOG-MSG field.";  
      reference
```

```
        "RFC 5424: The Syslog Protocol
        Std-1003.1-2008 Regular Expressions";
    }
}

grouping structured-data {
    description
        "This grouping defines the syslog structured data option
        which is used to select the format used to write log
        messages.";
    leaf structured-data {
        if-feature "structured-data";
        type boolean;
        default "false";
        description
            "This leaf describes how log messages are written.
            If true, messages will be written with one or more
            STRUCTURED-DATA elements; if false, messages will be
            written with STRUCTURED-DATA = NILVALUE.";
        reference
            "RFC 5424: The Syslog Protocol";
    }
}

container syslog {
    presence "Enables logging.";
    description
        "This container describes the configuration parameters for
        syslog.";
    container actions {
        description
            "This container describes the log-action parameters
            for syslog.";
        container console {
            if-feature "console-action";
            presence "Enables logging to the console";
            description
                "This container describes the configuration parameters
                for console logging.";
            uses selector;
        }
        container file {
            if-feature "file-action";
            description
                "This container describes the configuration parameters for
                file logging. If file-archive limits are not supplied, it
                is assumed that the local implementation defined limits
                will be used.";
        }
    }
}
```

```
list log-file {
  key "name";
  description
    "This list describes a collection of local logging
    files.";
  leaf name {
    type inet:uri {
      pattern 'file:.*';
    }
    description
      "This leaf specifies the name of the log file which
      MUST use the uri scheme file:.";
    reference
      "RFC 8089: The file URI Scheme";
  }
  uses selector;
  uses structured-data;
  container file-rotation {
    description
      "This container describes the configuration
      parameters for log file rotation.";
    leaf number-of-files {
      if-feature "file-limit-size";
      type uint32;
      default "1";
      description
        "This leaf specifies the maximum number of log
        files retained. Specify 1 for implementations
        that only support one log file.";
    }
    leaf max-file-size {
      if-feature "file-limit-size";
      type uint32;
      units "megabytes";
      description
        "This leaf specifies the maximum log file size.";
    }
    leaf rollover {
      if-feature "file-limit-duration";
      type uint32;
      units "minutes";
      description
        "This leaf specifies the length of time that log
        events should be written to a specific log file.
        Log events that arrive after the rollover period
        cause the current log file to be closed and a new
        log file to be opened.";
    }
  }
}
```

```
    leaf retention {
      if-feature "file-limit-duration";
      type uint32;
      units "minutes";
      description
        "This leaf specifies the length of time that
        completed/closed log event files should be stored
        in the file system before they are removed.";
    }
  }
}
container remote {
  if-feature "remote-action";
  description
    "This container describes the configuration parameters
    for forwarding syslog messages to remote relays or
    collectors.";
  list destination {
    key "name";
    description
      "This list describes a collection of remote logging
      destinations.";
    leaf name {
      type string;
      description
        "An arbitrary name for the endpoint to connect to.";
    }
    choice transport {
      mandatory true;
      description
        "This choice describes the transport option.";
      case udp {
        container udp {
          description
            "This container describes the UDP transport
            options.";
          reference
            "RFC 5426: Transmission of Syslog Messages over
            UDP";
          leaf address {
            type inet:host;
            description
              "The leaf uniquely specifies the address of
              the remote host. One of the following must be
              specified: an ipv4 address, an ipv6 address,
              or a host name.";
          }
        }
      }
    }
  }
}
```



```
        leaf port {
            type inet:port-number;
            default "514";
            description
                "This leaf specifies the port number used to
                deliver messages to the remote server.";
        }
    }
}
case tls {
    container tls {
        description
            "This container describes the TLS transport
            options.";
        reference
            "RFC 5425: Transport Layer Security (TLS)
            Transport Mapping for Syslog ";
        leaf address {
            type inet:host;
            description
                "The leaf uniquely specifies the address of
                the remote host. One of the following must be
                specified: an ipv4 address, an ipv6 address,
                or a host name.";
        }
        leaf port {
            type inet:port-number;
            default "6514";
            description
                "TCP port 6514 has been allocated as the default
                port for syslog over TLS.";
        }
        uses tlsc:tls-client-grouping;
    }
}
}
uses selector;
uses structured-data;
leaf facility-override {
    type identityref {
        base syslog-facility;
    }
    description
        "If specified, this leaf specifies the facility used
        to override the facility in messages delivered to
        the remote server.";
}
leaf source-interface {
```

```
if-feature "remote-source-interface";
type if:interface-ref;
description
  "This leaf sets the source interface to be used to
  send messages to the remote syslog server. If not
  set, messages can be sent on any interface.";
}
container signing {
  if-feature "signed-messages";
  presence "If present, syslog-signing options is activated.";
  description
    "This container describes the configuration
    parameters for signed syslog messages.";
  reference
    "RFC 5848: Signed Syslog Messages";
  container cert-signers {
    description
      "This container describes the signing certificate
      configuration for Signature Group 0 which covers
      the case for administrators who want all Signature
      Blocks to be sent to a single destination.";
    list cert-signer {
      key "name";
      description
        "This list describes a collection of syslog
        message signers.";
      leaf name {
        type string;
        description
          "This leaf specifies the name of the syslog
          message signer.";
      }
      container cert {
        uses ct:asymmetric-key-pair-with-certs-grouping;
        description
          "This is the certificate that is periodically
          sent to the remote receiver. The certificate
          is inherently associated with its private
          and public keys.";
      }
      leaf hash-algorithm {
        type enumeration {
          enum SHA1 {
            value 1;
            description
              "This enum describes the SHA1 algorithm.";
          }
          enum SHA256 {
```

```
        value 2;
        description
            "This enum describes the SHA256 algorithm.";
    }
}
description
    "This leaf describes the syslog signer hash
    algorithm used.";
}
}
leaf cert-initial-repeat {
    type uint32;
    default "3";
    description
        "This leaf specifies the number of times each
        Certificate Block should be sent before the first
        message is sent.";
}
leaf cert-resend-delay {
    type uint32;
    units "seconds";
    default "3600";
    description
        "This leaf specifies the maximum time delay in
        seconds until resending the Certificate Block.";
}
leaf cert-resend-count {
    type uint32;
    default "0";
    description
        "This leaf specifies the maximum number of other
        syslog messages to send until resending the
        Certificate Block.";
}
}
leaf sig-max-delay {
    type uint32;
    units "seconds";
    default "60";
    description
        "This leaf specifies when to generate a new
        Signature Block. If this many seconds have
        elapsed since the message with the first message
        number of the Signature Block was sent, a new
        Signature Block should be generated.";
}
}
leaf sig-number-resends {
    type uint32;
    default "0";
```

```

        description
            "This leaf specifies the number of times a
             Signature Block is resent. (It is recommended to
             select a value of greater than 0 in particular
             when the UDP transport RFC 5426 is used).";
    }
    leaf sig-resend-delay {
        type uint32;
        units "seconds";
        default "5";
        description
            "This leaf specifies when to send the next
             Signature Block transmission based on time. If
             this many seconds have elapsed since the previous
             sending of this Signature Block, resend it.";
    }
    leaf sig-resend-count {
        type uint32;
        default "0";
        description
            "This leaf specifies when to send the next
             Signature Block transmission based on a count.
             If this many other syslog messages have been
             sent since the previous sending of this
             Signature Block, resend it. A value of 0 means
             that you don't resend based on the number of
             messages.";
    }
}
}
}
}
}
}
}
<CODE ENDS>
```

Figure 2: Sylog YANG Model

7. Usage Examples

7.1. Syslog Configuration for Severity Critical

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

<!--
  Enable console logging of syslogs of severity critical
-->

<?xml version="1.0" encoding="UTF-8"?>
<syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog">
  <actions>
    <console>
      <facility-filter>
        <facility-list>
          <facility>all</facility>
          <severity>critical</severity>
        </facility-list>
      </facility-filter>
    </console>
  </actions>
</syslog>
```

Figure 3: Syslog Configuration for Severity Critical

7.2. Remote Syslog Configuration

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

<!--
    Enable remote logging of syslogs to udp destination
    foo.example.com for facility auth, severity error
-->
<?xml version="1.0" encoding="UTF-8"?>
<syslog xmlns="urn:ietf:params:xml:ns:yang:ietf-syslog">
  <actions>
    <remote>
      <destination>
        <name>remote1</name>
        <udp>
          <address>foo.example.com</address>
        </udp>
        <facility-filter>
          <facility-list>
            <facility>auth</facility>
            <severity>error</severity>
          </facility-list>
        </facility-filter>
      </destination>
    </remote>
  </actions>
</syslog>
```

Figure 4: Remote Syslog Configuration

8. Acknowledgements

The authors wish to thank the following who commented on this proposal:

Andy Bierman, Martin Bjorklund, Alex Campbell, Alex Clemm, Francis Dupont, Jim Gibson, Jeffrey Haas, Bob Harold, John Heasley, Giles Heron, Lisa Huang, Mahesh Jethanandani, Warren Kumari, Jeffrey K Lange, Jan Lindblad, Chris Lonvick, Alexey Melnikov, Kathleen Moriarty, Tom Petch, Adam Roach, Juergen Schoenwaelder, Phil Shafer, Yaron Sheffer, Jason Sterne, Peter Van Horne, Kent Watsen, Bert Wijnen, Dale R Worley, and Aleksandr Zhdankin.

9. IANA Considerations

9.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688] . Following the format in [RFC3688], the following registration is requested:

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

9.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC7895]. Following the format in [RFC7950], the following registration is requested:

name: ietf-syslog
namespace: urn:ietf:params:xml:ns:yang:ietf-syslog
prefix: ietf-syslog
reference: RFC zzzz

10. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes should be considered sensitive or vulnerable in all network environments. Logging in particular is used to assess the state of systems and can be used to indicate a network compromise. If logging were to be disabled through malicious means, attacks may not be readily detectable. Therefore write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations and on network security.

In addition there are data nodes that require careful analysis and review. These are the subtrees and data nodes and their sensitivity/vulnerability:

facility-filter/pattern-match: When writing this node, implementations MUST ensure that the regular expression pattern match is not constructed to cause a regular expression denial of service attack due to a pattern that causes the regular expression implementation to work very slowly (exponentially related to input size).

remote/destination/signing/cert-signer: When writing this subtree, implementations MUST NOT specify a private key that is used for any other purpose.

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

remote/destination/transport: This subtree contains information about other hosts in the network, and the TLS transport certificate properties if TLS is selected as the transport protocol.

remote/destination/signing: This subtree contains information about the syslog message signing properties including signing certificate information.

There are no RPC operations defined in this YANG module.

11. References

11.1. Normative References

[I-D.ietf-netconf-crypto-types]

Watsen, K., "YANG Data Types and Groupings for Cryptography", Work in Progress, Internet-Draft, draft-ietf-netconf-crypto-types-22, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-netconf-crypto-types-22.txt>>.

[I-D.ietf-netconf-tls-client-server]

Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-27, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-netconf-tls-client-server-27.txt>>.

[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>>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <<https://www.rfc-editor.org/info/rfc5424>>.

- [RFC5425] Miao, F., Ed., Ma, Y., Ed., and J. Salowey, Ed., "Transport Layer Security (TLS) Transport Mapping for Syslog", RFC 5425, DOI 10.17487/RFC5425, March 2009, <<https://www.rfc-editor.org/info/rfc5425>>.
- [RFC5426] Okmianski, A., "Transmission of Syslog Messages over UDP", RFC 5426, DOI 10.17487/RFC5426, March 2009, <<https://www.rfc-editor.org/info/rfc5426>>.
- [RFC5848] Kelsey, J., Callas, J., and A. Clemm, "Signed Syslog Messages", RFC 5848, DOI 10.17487/RFC5848, May 2010, <<https://www.rfc-editor.org/info/rfc5848>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <<https://www.rfc-editor.org/info/rfc7895>>.
- [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>>.
- [RFC8089] Kerwin, M., "The "file" URI Scheme", RFC 8089, DOI 10.17487/RFC8089, February 2017, <<https://www.rfc-editor.org/info/rfc8089>>.
- [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>>.
- [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>>.
- [Std-1003.1-2008] Group, I. A. T. O., "'Chapter 9: Regular Expressions". The Open Group Base Specifications Issue 6, IEEE Std 1003.1-2008, 2016 Edition.", September 2016, <<http://pubs.opengroup.org/onlinepubs/9699919799/>>.

11.2. Informative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.

- [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>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [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>>.
- [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>>.
- [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>>.

Appendix A. Implementer Guidelines

A.1. Extending Facilities

Many vendors extend the list of facilities available for logging in their implementation. Additional facilities may not work with the syslog protocol as defined in [RFC5424] and hence such facilities apply for local syslog-like logging functionality.

The following is an example that shows how additional facilities could be added to the list of available facilities (in this example two facilities are added):

```
module example-vendor-syslog-types {
  namespace "http://example.com/ns/vendor-syslog-types";
  prefix vendor-syslogtypes;

  import ietf-syslog {
    prefix syslogtypes;
  }

  organization "Example, Inc.";
  contact
    "Example, Inc.
     Customer Service

     E-mail: syslog-yang@example.com";

  description
    "This module contains a collection of vendor-specific YANG type
     definitions for SYSLOG.";

  revision 2017-08-11 {
    description
      "Version 1.0";
    reference
      "Vendor SYSLOG Types: SYSLOG YANG Model";
  }

  identity vendor_specific_type_1 {
    base syslogtypes:syslog-facility;
    description
      "Adding vendor specific type 1 to syslog-facility";
  }

  identity vendor_specific_type_2 {
    base syslogtypes:syslog-facility;
    description
      "Adding vendor specific type 2 to syslog-facility";
  }
}
```

A.2. Syslog Terminal Output

Terminal output with requirements more complex than the console subtree currently provides, are expected to be supported via vendor extensions rather than handled via the file subtree.

A.3. Syslog File Naming Convention

The `syslog/file/log-file/file-rotation` container contains configuration parameters for syslog file rotation. This section describes how these fields might be used by an implementer to name syslog files in a rotation process. This information is offered as an informative guide only.

When an active syslog file with a name specified by `log-file/name`, reaches `log-file/max-file-size` and/or syslog events arrive after the period specified by `log-file/rollover`, the logging system can close the file, can compress it, and can name the archive file `<log-file/name>.0.gz`. The logging system can then open a new active syslog file `<log-file/name>`.

When the new syslog file reaches either of the size limits referenced above, `<log-file/name>.0.gz` can be renamed `<log-file/name>.1.gz` and the new syslog file can be closed, compressed and renamed `<log-file/name>.0.gz`. Each time that a new syslog file is closed, each of the prior syslog archive files named `<log-file/name>.<n>.gz` can be renamed to `<log-file/name>.<n + 1>.gz`.

Removal of archive log files could occur when either or both:

- `log-file/number-of-files` specified - the logging system can create up to `log-file/number-of-files` syslog archive files after which, the contents of the oldest archived file could be overwritten.

- `log-file/retention` specified - the logging system can remove those syslog archive files whose file expiration time (file creation time plus the specified `log-file/retention` time) is prior to the current time.

Authors' Addresses

Joe Clarke (editor)
Cisco
United States of America
Email: jclarke@cisco.com

Mahesh Jethanandani (editor)
Kloud Services
United States of America
Email: mjethanandani@gmail.com

Clyde Wildes (editor)
Cisco Systems Inc.
170 West Tasman Drive
San Jose, CA 95134
United States of America
Phone: +1 408 527-2672
Email: cwildes@cisco.com

Kiran Koushik (editor)
Verizon Wireless
500 W Dove Rd.
Southlake, TX 76092
United States of America
Phone: +1 512 650-0210
Email: kirankoushik.agraharasreenivasa@verizonwireless.com

NETMOD
Internet-Draft
Intended status: Informational
Expires: December 15, 2017

D. Bogdanovic
Volta Networks, Inc.
B. Claise
C. Moberg
Cisco Systems, Inc.
June 13, 2017

YANG Module Classification
draft-ietf-netmod-yang-model-classification-08

Abstract

The YANG data modeling language is currently being considered for a wide variety of applications throughout the networking industry at large. Many standards development organizations (SDOs), open source software projects, vendors and users are using YANG to develop and publish YANG modules for a wide variety of applications. At the same time, there is currently no well-known terminology to categorize various types of YANG modules.

A consistent terminology would help with the categorization of YANG modules, assist in the analysis of the YANG data modeling efforts in the IETF and other organizations, and bring clarity to the YANG-related discussions between the different groups.

This document describes a set of concepts and associated terms to support consistent classification of YANG modules.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 15, 2017.

Copyright Notice

Copyright (c) 2017 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. First Dimension: YANG Module Abstraction Layers	4
2.1. Network Service YANG Modules	6
2.2. Network Element YANG Modules	7
3. Second Dimension: Module Origin Types	7
3.1. Standard YANG Modules	8
3.2. Vendor-specific YANG Modules and Extensions	8
3.3. User-specific YANG Modules and Extensions	9
4. Security Considerations	9
5. IANA Considerations	9
6. Acknowledgements	9
7. Change log [RFC Editor: Please remove]	9
8. References	10
8.1. Normative References	10
8.2. Informative References	10
Authors' Addresses	11

1. Introduction

The Internet Engineering Steering Group (IESG) has been actively encouraging IETF working groups to use the YANG data modeling language [RFC7950], [RFC7950] and NETCONF protocol [RFC6241] for configuration management purposes, especially in new working group charters [Writable-MIB-Module-IESG-Statement].

YANG is also gaining wide acceptance as the de-facto standard data modeling language in the broader industry. This extends beyond the IETF, including many standards development organizations, industry consortia, ad hoc groups, open source projects, vendors, and end-users.

There are currently no clear guidelines on how to classify the layering of YANG modules according to abstraction, or how to classify modules along the continuum spanning formal standards publications, vendor-specific modules and modules provided by end-users.

This document presents a set of concepts and terms to form a useful taxonomy for consistent classification of YANG modules in two dimensions:

- o The layering of modules based on their abstraction levels
- o The module origin type based on the nature and intent of the content

The intent of this document is to provide a taxonomy to simplify human communication around YANG modules. While the classification boundaries are at times blurry, this document should provide a robust starting point as the YANG community gains further experience with designing and deploying modules. To be more explicit, it is expected that the classification criteria will change over time.

A number of modules have created substantial discussion during the development of this document: for examples, modules concerned with topologies. Topology modules are useful both on the Network Element level (e.g. link-state database content) as well as on the Network Service level (e.g. network-wide, configured topologies). In the end, it is the module developer that classifies the module according to the initial intent of the module content.

This document should provide benefits to multiple audiences:

- o First, a common taxonomy helps with the different standards development organizations and industry consortia discussions, whose goals are determined in their respective areas of work.
- o Second, operators might look at the YANG module abstraction layers to understand which Network Service YANG modules and Network Element YANG modules are available for their service composition. It is difficult to determine the module type without inspecting the YANG module itself. The YANG module name might provide some useful information but is not a definite answer. For example, an L2VPN YANG module might be a Network Service YANG module, ready to be used as a service model by a network operator. Alternatively, it might be a Network Element YANG module that contains the L2VPN data definitions required to be configured on a single device.
- o And thirdly, this taxonomy would help equipment vendors (whether physical or virtual), controller vendors, orchestrator vendors to

explain to their customers the relationship between the different YANG modules they support in their products.

1.1. Terminology

[RFC7950] specifies:

- o data model: A data model describes how data is represented and accessed.
- o module: A YANG module defines hierarchies of schema nodes. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and "compilable".

2. First Dimension: YANG Module Abstraction Layers

Module developers have taken two approaches to developing YANG modules: top-down and bottom-up. The top-down approach starts with high level abstractions modeling business or customer requirements and maps them to specific networking technologies. The bottom-up approach starts with fundamental networking technologies and maps them into more abstract constructs.

There are currently no specific requirements on, or well-defined best practices around the development of YANG modules. This document considers both bottom-up and top-down approaches as they are both used and they each provide benefits that appeal to different groups.

For layering purposes, this document suggests the classification of YANG modules into two distinct abstraction layers:

- o Network Element YANG Modules describe the configuration, state data, operations and notifications of specific device-centric technologies or features
- o Network Service YANG Modules describe the configuration, state data, operations and notifications of abstract representations of services implemented on one or multiple network elements

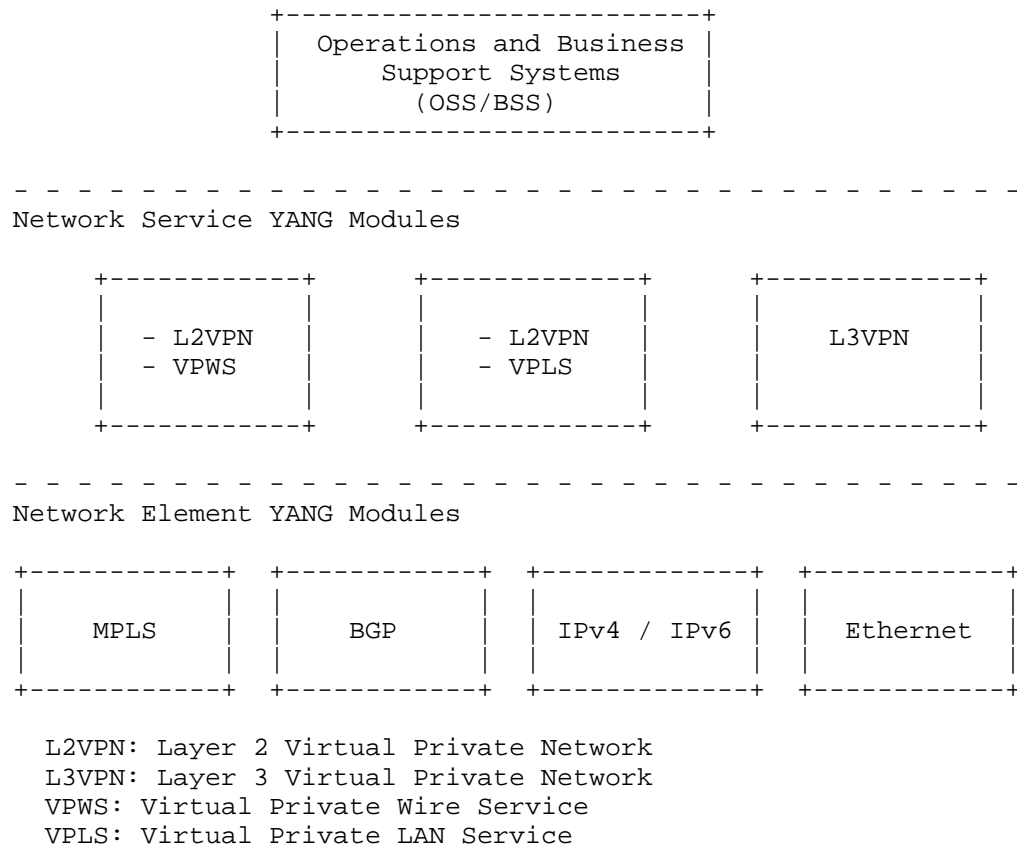


Figure 1: YANG Module Abstraction Layers

Figure 1 illustrates the application of YANG modules at different layers of abstraction. Layering of modules allows for reusability of existing lower layer modules by higher level modules while limiting duplication of features across layers.

For module developers, per-layer modeling allows for separation of concern across editing teams focusing on specific areas.

As an example, experience from the IETF shows that creating useful Network Element YANG modules for e.g. routing or switching protocols requires teams that include developers with experience of implementing those protocols.

On the other hand, Network Service YANG modules are best developed by network operators experienced in defining network services for

consumption by programmers developing e.g. flow-through provisioning systems or self-service portals.

2.1. Network Service YANG Modules

Network Service YANG Modules describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service module does not expose the detailed configuration parameters of all participating network elements and features, but describes an abstract model that allows instances of the service to be decomposed into instance data according to the Network Element YANG Modules of the participating network elements. The service-to-element decomposition is a separate process with details depending on how the network operator chooses to realize the service. For the purpose of this document, the term "orchestrator" is used to describe to describe a system implementing such a process.

Network Service YANG Modules define service models to be consumed by external systems. External systems can be provisioning systems, service orchestrators, Operations Support Systems, Business Support Systems and applications exposed to network service consumers, being either internal network operations people or external customers. These modules are commonly designed, developed and deployed by network infrastructure teams.

YANG allows for different design patterns to describe network services, ranging from monolithic to component-based approaches.

The monolithic approach captures the entire service in a single module and does not put focus on reusability of internal data definitions and groupings. The monolithic approach has the advantages of single-purpose development including development speed at the expense of reusability.

The component-based approach captures device-centric features (e.g. the definition of a VPN Routing and Forwarding (VRF), routing protocols, or packet filtering) in a vendor-independent manner. The components are designed for reuse across many service modules. The set of components required for a specific service is then composed into the higher-level service. The component-based approach has the advantages of modular development including a higher degree of reusability at the expense of initial development speed.

As an example, an L2VPN service can be built on many different types of transport network technologies, including e.g. MPLS or carrier ethernet. A component-based approach would allow for reuse of e.g. User-Network Interface (UNI) definitions independent of the underlying transport network (e.g. MEF UNI interface or MPLS

interface). The monolithic approach would assume a specific set of transport technologies and interface definitions.

An example of a Network Service YANG module is in [RFC8049]. It provides an abstract model for Layer 3 IP VPN service configuration. This module includes e.g. the concept of a 'site-network-access' to represent bearer and connection parameters. An orchestrator receives operations on service instances according to the service module and decomposes the data into configuration data according to specific Network Element YANG Modules to configure the participating network elements to the service. In the case of the L3VPN module, this would include translating the 'site-network-access' parameters to the appropriate parameters in the Network Element YANG Module implemented on the constituent elements.

2.2. Network Element YANG Modules

Network Element YANG Modules describe the characteristics of a network device as defined by the vendor of that device. The modules are commonly structured around features of the device, e.g. interface configuration [RFC7223], OSPF configuration [I-D.ietf-ospf-yang], and firewall rules definitions [I-D.ietf-netmod-acl-model].

The module provides a coherent data model representation of the software environment consisting of the operating system and applications running on the device. The decomposition, ordering, and execution of changes to the operating system and application configuration is the task of the agent that implements the module.

3. Second Dimension: Module Origin Types

This document suggests classifying YANG module origin types as standard YANG modules, vendor-specific YANG modules and extensions, or user-specific YANG modules and extensions

The suggested classification applies to both Network Element YANG Modules and Network Service YANG Modules.

It is to be expected that real-world implementations of both Network Service YANG Modules and Network Element YANG Modules will include a mix of all three module origin types.

Figure 2 illustrates the relationship between the three types of modules.

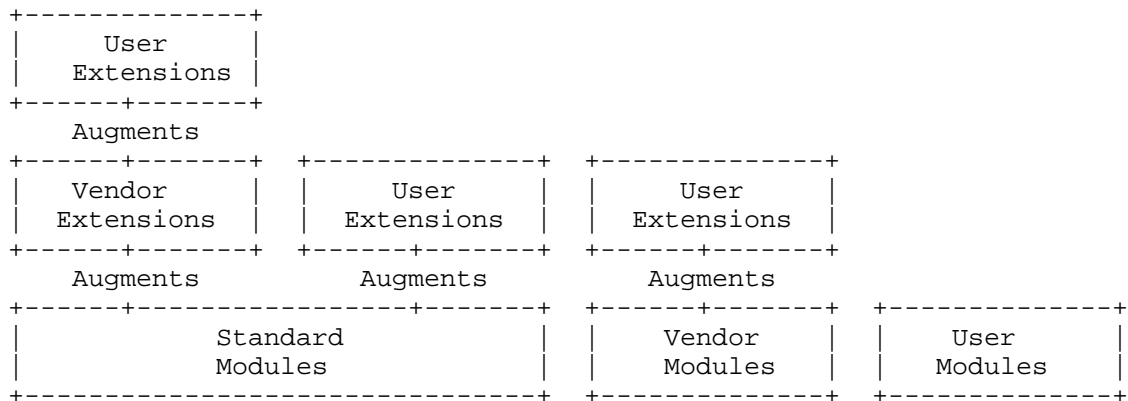


Figure 2: YANG Module Origin Types

3.1. Standard YANG Modules

Standard YANG Modules are published by standards development organizations (SDOs). Most SDOs create specifications according to a formal process in order to produce a standard that is useful for their constituencies.

The lifecycle of these modules is driven by the editing cycle of the specification and not tied to a specific implementation.

Examples of SDOs in the networking industry are the IETF and the IEEE.

3.2. Vendor-specific YANG Modules and Extensions

Vendor-specific YANG Modules are developed by organizations with the intent to support a specific set of implementations under control of that organization. For example vendors of virtual or physical equipment, industry consortia, and opensource projects. The intent of these modules range from providing openly published YANG modules that may eventually be contributed back to, or adopted by, an SDO, to strictly internal YANG modules not intended for external consumption.

The lifecycle of these modules are generally aligned with the release cycle of the product or open source software project deliverables.

It is worth noting that there is an increasing amount of interaction between open source projects and SDOs in the networking industry. This includes open source projects implementing published standards as well as open source projects contributing content to SDO processes.

Vendors also develop Vendor-specific Extensions to standard modules using YANG constructs for extending data definitions of previously published modules. This is done using the 'augment' statement that allows locally defined data trees to be added into locations in externally defined data trees.

Vendors use this to extend standard modules to cover the full scope of features in implementations, which commonly is broader than that covered by the standard module.

3.3. User-specific YANG Modules and Extensions

User-specific YANG Modules are developed by organizations that operate YANG-based infrastructure including devices and orchestrators. For example, network administrators in enterprises, or at service providers. The intent of these modules is to express the specific needs for a certain implementation, above and beyond what is provided by vendors.

This module type obviously requires the infrastructure to support the introduction of user-provided modules and extensions. This would include the ability to describe the service-to-network decomposition in orchestrators and the module to configuration decomposition in devices.

The lifecycles of these modules are generally aligned with the change cadence of the infrastructure.

4. Security Considerations

This document doesn't have any Security Considerations.

5. IANA Considerations

This document has no IANA actions.

6. Acknowledgements

Thanks to David Ball and Jonathen Hansford for feedback and suggestions.

7. Change log [RFC Editor: Please remove]

version 00: Renamed and small fixes based on WG feedback.

version 01: Language fixes, collapsing of vendor data models and extensions, and the introduction of user data models and extensions.

version 02: Updated the YANG Module Catalog section, terminology alignment (YANG data model versus YANG module), explain better the distinction between the Network Element and Service YANG data models even if sometimes there are grey areas, editorial pass. Changed the use of the term 'model' to 'module' to be better aligned with RFC6020.

version 06: updates based on comments from Adrian Farrel about YANG Data Model for L3VPN Service Delivery.

version 07: updates based on comments from Pete Resnick

8. References

8.1. Normative References

- [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, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <<http://www.rfc-editor.org/info/rfc8049>>.

8.2. Informative References

- [I-D.ietf-netmod-acl-model]
Bogdanovic, D., Koushik, K., Huang, L., and D. Blair,
"Network Access Control List (ACL) YANG Data Model",
draft-ietf-netmod-acl-model-10 (work in progress), March 2017.
- [I-D.ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem,
"Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-07 (work in progress), March 2017.

```
[Writable-MIB-Module-IESG-Statement]
    "Writable MIB Module IESG Statement",
    <https://www.ietf.org/iesg/statement/writable-mib-
module.html>.
```

Authors' Addresses

Dean Bogdanovic
Volta Networks, Inc.

Email: dean@voltanet.io

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Carl Moberg
Cisco Systems, Inc.

Email: camoberg@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 22, 2016

E. Lear
Cisco Systems
January 19, 2016

Using DNS Names in the IETF ACL Model
draft-lear-ietf-netmod-acl-dnsname-00

Abstract

End points are commonly referenced by higher level functions through the DNS. This is especially the case in cloud-based functions, which might have hundreds of IP addresses for the same name. This brief memo extends the IETF-ACL model to allow access-control via domain names.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 22, 2016.

Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Element Definitions	3
2.1. source-dnsname	3
2.2. destination-dnsname	3
3. The ietf-acl-dnsname Model	3
4. Example	4
5. Security Considerations	5
6. IANA Considerations	5
7. Acknowledgments	6
8. Normative References	6
Author's Address	6

1. Introduction

The IETF-ACL model [I-D.ietf-netmod-acl-model] specifies a schema for access lists. That model is intentionally kept constrained to the small number of packet-passing functions that are considered ubiquitous. While that is a necessary step, there are a number of circumstances in which it will not be sufficient. In a world where load balancing and shifting commonly takes place, it may not be practical to maintain the complete list of IP addresses in all instances. Furthermore, even in more static environments, occasionally the name to address mapping needs to change. Lastly, there are resources that may not be tied to packet processing at all that may yet be well described by this augmentation. Allowing domain names in ACLs reduces the number of points within a network that need to be reconfigured when such changes take place.

This memo specifies an extension to IETF-ACL model such that domain names may be referenced by augmenting the "matches" element. Different implementations may deploy differing methods to maintain the mapping between IP address and domain name, if indeed any are needed. However, the intent is that resources that are referred to using a name should be authorized (or not) within an access list.

The structure of the change is as follows:

```
augment
/acl:access-lists/acl:acl/acl:access-list-entries
  /acl:ace/acl:matches/acl:ace-type/acl:ace-ip:
    +--rw source-dnsname?      inet:host
    +--rw destination-dnsname? inet:host
```

The choice of this particular point in the access-list model is based on the assumption that we are in some way referring to IP-related

resources, as that is what the DNS returns. A domain name in our context is defined in [RFC6991].

2. Element Definitions

The following elements are defined.

2.1. source-dnsname

The argument corresponds to a domain name of a source as specified by `inet:host`. Depending on how the model is used, it may or may not be resolved, as required by the implementation and circumstances.

2.2. destination-dnsname

The argument corresponds to a domain name of a destination as specified by `inet:host`. Depending on how the model is used, it may or may not be resolved, as required by the implementation and circumstances.

3. The ietf-acl-dnsname Model

```
<CODE BEGINS>file "ietf-acl-dnsname.yang";

module ietf-acl-dnsname {
  yang-version 1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-acl-dnsname";
  prefix "ietf-acl-dnsname";

  import ietf-access-control-list {
    prefix "acl";
  }

  import ietf-inet-types
  {
    prefix "inet";
  }

  organization
    "Cisco Systems, Inc.";

  contact
    "Eliot Lear
     lear@cisco.com
    ";

  description
```

```
"This YANG module defines a component that augments the
IETF description of an access list to allow dns names
as matching criteria.";

revision "2016-01-14" {
  description "Initial revision";
  reference "This document?";
}

augment "/acl:access-lists/acl:acl/" +
  "acl:access-list-entries/acl:ace/" +
  "acl:matches/acl:ace-type/acl:ace-ip" {
  description "adding domain names to matching";

  leaf source-dnsname {
    type inet:host;
    description "domain name to be matched against";
  }
  leaf destination-dnsname {
    type inet:host;
    description "domain name to be matched against";
  }
}

}

<CODE ENDS>
```

4. Example

The following example is taken from [I-D.ietf-netmod-acl-model] (the optional and irrelevant components have been removed). It allows traffic from `www.cloud.example.com`.

```
<?xml version='1.0' encoding='UTF-8'?>
  <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <access-lists
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      xmlns:ietf-acl-dnsname="urn:ietf:params:xml:ns:yang:ietf-acl-dnsname">
      <acl>
        <acl-oper-data />
        <access-list-entries>
          <ace>
            <matches>
              <source-dnsname>
                www.cloud.example.com
              </destination-dnsname>
            </matches>
            <actions>
              <permit />
            </actions>
            <rule-name>rule1<rule-name/>
          </ace>
        </access-list-entries>
        <acl-name>sample-dns-acl<acl-name/>
        <acl-type>ipv4-acl<acl-type/>
      </acl>
    </access-lists>
  </data>
```

5. Security Considerations

If the mapping between a domain name and the underlying resource to which it refers becomes stale, the access list will be incorrect. It is therefore important that implementations employ some means for maintaining the mapping, if it is required. In those circumstances, when other systems are in play, those other systems would be required to indicate what domains they are attempting to connect to. Under the current circumstances, this is readily observable. However, in future such information sharing may raise privacy concerns, and the name and mapping may not be available to the system employing the ACL model.

6. IANA Considerations

The IANA is not requested to make any changes. The RFC Editor is requested to remove this section prior to publication.

7. Acknowledgments

The author wishes to acknowledge Kiran Koushik and Einar Nilsen-Nygaard for their review and contributions to this work.

8. Normative References

- [I-D.ietf-netmod-acl-model]
Bogdanovic, D., Koushik, K., Huang, L., and D. Blair,
"Network Access Control List (ACL) YANG Data Model",
draft-ietf-netmod-acl-model-06 (work in progress),
December 2015.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC
6991, DOI 10.17487/RFC6991, July 2013,
<<http://www.rfc-editor.org/info/rfc6991>>.

Author's Address

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 2, 2018

X. Liu
Jabil
I. Bryskin
Huawei Technologies
V. Beeram
Juniper Networks
T. Saad
Cisco Systems Inc
H. Shah
Ciena
O. Gonzalez de Dios
Telefonica
March 1, 2018

A YANG Data Model for Configuration Scheduling
draft-liu-netmod-yang-schedule-05

Abstract

This document describes a data model for configuration scheduling.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 2, 2018.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Motivation	3
3. Configuration Scheduling YANG Data Model Overview	3
4. Usage Example	4
5. Relations to Datastores	6
5.1. Validation	6
5.2. Schedules Expansion and Operational States	7
5.3. Server Executions at Scheduled Moments	7
5.4. Interactions with Locks	7
5.5. Interactions with Authorization Mechanism	7
6. Synchronization Aspects	7
7. Configuration Scheduling YANG Module	8
8. IANA Considerations	13
9. Security Considerations	14
10. Contributors	14
11. References	14
11.1. Normative References	14
11.2. Informative References	16
Authors' Addresses	16

1. Introduction

This document introduces a YANG [RFC6020] [RFC7950] data model for configuration scheduling. This model can be used together with other YANG data models to specify a schedule applied on a configuration data node, so that the configuration data can take effect according to the schedule. Such a configuration schedule can be one-time or recurring, with its properties persistently saved in the datastores of the management system server.

The mechanism described in this document is designed to complement the one described in [RFC7758], which defines a capability extension to NETCONF to allow time-triggered RPCs. Such RPCs can be executed at a future time moment, but cannot be repeated and is not saved in the persistent datastores.

1.1. Terminology

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

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

- o augment
- o data model
- o data node

2. Motivation

Some applications benefit from resource scheduling to allow operators to plan ahead of time. Traffic engineering is one of such examples [RFC7399]. When configuration and state models are designed for such applications, it has been considered that certain data objects need to be configured according to predefined schedules. In other situations, operators need to deconfigure certain data objects at predefined schedules for the purposes such as maintenance. These data objects are interpreted and implemented by the applicable applications.

Delay/Disruption Tolerant Networking (DTN) is another example for which the scheduled configuration can be used, where a long-lived, reliable, low-latency sequenced data delivery session is unsustainable. Section 4.3 of [I-D.birrane-dtn-ama] describes the Autonomous Parameterized Control. Time-based event is one of the two types of triggers in such a system.

3. Configuration Scheduling YANG Data Model Overview

This document defines a YANG data model that specifies configuration schedules for other YANG data models. For each targeted configuration data object or a group of configuration data objects, an entry is specified along with requested schedules using this configuration schedule model. The application implementing the targeted schema nodes implements the configuration schedules, configuring or deconfiguring the specified objects according to the specified schedules. The model schema of the targeted application does not need changes, so the data model described in this document can be used for any data model. The configuration scheduling YANG data model has the following structure:

```

module: ietf-schedule
  +--rw configuration-schedules
    +--rw target* [object]
      +--rw object          yang:xpath1.0
      +--rw schedules
        +--rw schedule* [schedule-id]
          +--rw schedule-id          uint32
          +--rw inclusive-exclusive? enumeration
          +--rw start?               yang:date-and-time
          +--rw schedule-duration?   string
          +--rw repeat-interval?     string
          +--rw operation?           operation
          +--rw data-value?          <anydata>
        +--ro state
          +--ro future-executions
            +--ro execution* [start]
              +--ro start            yang:date-and-time
              +--ro duration?        string
              +--ro operation?       operation
        +---n execution
          +---- operation            operation
          +---- datetime?           yang:date-and-time
          +---- results?            <anydata>

```

4. Usage Example

The following model defines a list of TE (Traffic Engineering) links which can be configured with specified schedules:

```

module: example
  +--rw te-links
    +--rw te-link* [id]
      +--rw id          string
      +--rw enabled?    boolean

```

The following configuration requests that

- o link-1 is configured weekly for five one-day periods, starting from 2016-09-12T23:20:50.52Z.
- o link-2 is deconfigured for two hours, starting from 2016-09-15T01:00:00.00Z.

```
<configuration-schedules>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links</object>
    <schedules>
      <schedule>
        <schedule-id>11</schedule-id>
        <start>2016-09-12T23:20:50.52Z</start>
        <schedule-duration>P1D</schedule-duration>
        <repeat-interval>R5/P1W</repeat-interval>
        <operation>configure</operation>
        <data-value>
          <te-link>
            <id>link-1</id>
            <enabled>true</enabled>
          </te-link>
        </data-value>
      </schedule>
    </schedules>
  </target>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links</object>
    <schedules>
      <schedule>
        <schedule-id>12</schedule-id>
        <inclusive-exclusive>exclusive</inclusive-exclusive>
        <start>2016-09-15T01:00:00.00Z</start>
        <schedule-duration>P2H</schedule-duration>
        <operation>configure</operation>
        <data-value>
          <te-link>
            <id>link-2</id>
            <enabled>true</enabled>
          </te-link>
        </data-value>
      </schedule>
    </schedules>
  </target>
</configuration-schedules>
```

The following configuration requests that

- o link-1 is enabled weekly for five one-day periods, starting from 2016-09-12T23:20:50.52Z.
- o link-2 is not enabled for two hours, starting from 2016-09-15T01:00:00.00Z.

```
<configuration-schedules>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links/ex:te-link[ex:link-id='link-1']/ex:enabled
    </object>
    <schedules>
      <schedule>
        <schedule-id>11</schedule-id>
        <start>2016-09-12T23:20:50.52Z</start>
        <schedule-duration>P1D</schedule-duration>
        <repeat-interval>R5/P1W</repeat-interval>
        <operation>set</operation>
        <data-value>true</data-value>
      </schedule>
    </schedules>
  </target>
  <target xmlns:ex="urn:example">
    <object>/ex:te-links/ex:te-link[ex:link-id='link-2']/ex:enabled
    </object>
    <schedules>
      <schedule>
        <schedule-id>12</schedule-id>
        <inclusive-exclusive>exclusive</inclusive-exclusive>
        <start>2016-09-15T01:00:00.00Z</start>
        <schedule-duration>P2H</schedule-duration>
        <operation>set</operation>
        <data-value>true</data-value>
      </schedule>
    </schedules>
  </target>
</configuration-schedules>
```

5. Relations to Datastores

NETCONF defines configuration datastores and operations that can be used to access these datastores. The configuration data encoded according to this data model is persistently saved in the proper datastores in the same way as other data model, such as ietf-interfaces.

5.1. Validation

When configuration data based on this model is received, the server MUST perform syntax validations on the received data nodes, and examine the requested schedules. The server does not validate whether requested target configuration data can be applied to the

target configuration objects, until the actual scheduled time arrives.

At each scheduled time moment, the server applies the requested target configuration data to the target configuration objects. The server **MUST** perform the validations on the target configuration data along with the current target configuration objects in the proper datastore.

5.2. Schedules Expansion and Operational States

The server **SHOULD** expand these schedules and expose them to the client as operational states.

5.3. Server Executions at Scheduled Moments

At each scheduled time moment, the server applies the requested target configuration data to the target configuration objects, as if an RPC request is newly received. Whether such a time-triggered configuration is successfully applied depends on the configuration data of the target object and requested configuration data. The results of such executions are sent to the client through notifications. The notification management mechanism described in [I-D.ietf-netconf-yang-push] and [I-D.ietf-netconf-subscribed-notifications] can be used to enable, disable, subscribe, filter, and replay the notifications.

5.4. Interactions with Locks

The rules of datastore lock specified by NETCONF [RFC6241] are checked when the schedule configuration data is received and when the target configuration data is applied.

5.5. Interactions with Authorization Mechanism

If the server implements any authorization mechanism, the authorization rules **MUST** be checked against this data model schema when the schedule configuration data is received. At each scheduled time moment, the authorization rules **MUST** be checked against the target objects by using the target configuration data. To check the authorization rules, the server uses the same client credential learned when the initial configuration data was received.

6. Synchronization Aspects

The scheduling mechanisms described in this document assume that servers have access to the wall-clock time. Thus, servers are required to acquire the time-of-day from an external time source, for

example using the Network Time Protocol [RFC5905], or the Precision Time Protocol [IEEE1588].

It is assumed that the client and servers rely on a common time source, so as to guarantee that schedules are defined with respect to a common reference. In order to avoid the potential ambiguity of different time zones and daylight saving time, it is recommended to define all schedules in the UTC time zone, using the suffix 'Z'. For example, the time 2016-09-12T23:20:50.52Z, is specified with respect to the UTC time zone.

7. Configuration Scheduling YANG Module

```
<CODE BEGINS> file "ietf-schedule@2018-02-26.yang"
module ietf-schedule {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schedule";

  prefix "sch";

  import ietf-yang-types {
    prefix "yang";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact
    "WG Web:    <http://tools.ietf.org/wg/netmod/>
    WG List:    <mailto:netmod@ietf.org>

    Editor:     Xufeng Liu
                <mailto:Xufeng_Liu@jabil.com>

    Editor:     Igor Bryskin
                <mailto:Igor.Bryskin@huawei.com>

    Editor:     Vishnu Pavan Beeram
                <mailto:vbeeram@juniper.net>

    Editor:     Tarek Saad
                <mailto:tsaad@cisco.com>

    Editor:     Himanshu Shah
                <mailto:hshah@ciena.com>

    Editor:     Oscar Gonzalez De Dios
                <mailto:oscar.gonzalezdedios@telefonica.com>";
```

```
description
  "The model allows time scheduling parameters to be specified.";

revision 2018-02-26 {
  description "Initial revision";
  reference "TBD";
}

/*
 * Typedefs
 */
typedef operation {
  type enumeration {
    enum configure {
      description
        "Create the configuration data.";
    }
    enum deconfigure {
      description
        "Remove the configuration data.";
    }
    enum set {
      description
        "Set the specified configuration data.";
    }
    enum reset {
      description
        "Revert the specified configuration data back to the
         original value.";
    }
  }
  description "Operation type.";
}

/*
 * Groupings
 */

grouping schedule-config-attributes {
  description
    "A group of attributes for a schedule.";

  leaf inclusive-exclusive {
    type enumeration {
      enum inclusive {
        description
          "The schedule element is inclusive, i.e., the schedule
           specifies the time at which the element is enabled.";
      }
    }
  }
}
```

```

    }
    enum exclusive {
        description
            "The schedule element is exclusive. i.e., the schedule
            specifies the time at which the element is disabled.";
    }
}
default "inclusive";
description
    "Whether the list item is inclusive or exclusive.";
}
leaf start {
    type yang:date-and-time;
    description "Start time.";
}
leaf schedule-duration {
    type string {
        pattern
            'P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?(\d+S)?';
    }
    description "Schedule duration in ISO 8601 format.";
}
leaf repeat-interval {
    type string {
        pattern
            'R\d*/P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?'
            + '(\d+S)?';
    }
    description "Repeat interval in ISO 8601 format.";
}
leaf operation {
    type operation;
    default "configure";
    description
        "Operation type.";
}
anydata data-value {
    description
        "The data value applied to the leaf data node
        specified by data-objects.
        The format of the data value depends on the value of the
        leaf operation defined above:
        configure: data-value is the sub-tree added to the
                    target object;
        deconfigure: data-value is the child to be deleted from
                     the target object;
        set:         the target object MUST be a leaf, and
                     data-value is the new value to be set to

```



```
        the target object;
    reset:      data-value is ignored.";
}
} // schedule-config-attributes

grouping schedule-config-notification {
    description
        "A group of attributes for a schedule notification.";

    notification execution {
        description
            "Notification event for an execution performed on a target
            object.";
        leaf operation {
            type operation;
            mandatory true;
            description "Operation type.";
        }
        leaf datetime {
            type yang:date-and-time;
            description
                "The date and time when the execution was performed.";
        }
        anydata results {
            description
                "This chunk of data contains the results of the execution
                performed on the target object. The results are the same
                or equivalent to the contents of a <rpc-reply> message,
                Because of the nature of such a target execution, a
                <rpc-reply> message is not used to return the execution
                results. Instead, this notification is used to serve
                the same purpose.";
        }
    }
} // schedule-config-notification

grouping schedule-state-attributes {
    description
        "State attributes for a schedule.";
    container future-executions {
        description
            "The state information of the next scheduled event.";
        list execution {
            key "start";
            description
                "List of scheduled future executions.";
            leaf start {
                type yang:date-and-time;
            }
        }
    }
}
```

```

        description "Start time.";
    }
    leaf duration {
        type string {
            pattern
                'P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?(\d+M)?(\d+S)?';
        }
        description "Schedule duration in ISO 8601 format.";
    }
    leaf operation {
        type operation;
        description "Operation type.";
    }
} // event
} // future-events
} // schedule-state-attributes

grouping schedules {
    description
        "A list of schedules defining when a particular
        configuration takes effect.";
    container schedules {
        description
            "Container of a schedule list defining when a particular
            configuration takes effect.";
        list schedule {
            key "schedule-id";
            description "A list of schedule elements.";
            leaf schedule-id {
                type uint32;
                description "Identifies the schedule element.";
            }
            uses schedule-config-attributes;
        }
    }
} // schedules

/*
 * Configuration data and operational state nodes
 */
container configuration-schedules {
    description
        "Serves as top-level container for a list of configuration
        schedules.";
    list target {
        key "object";
        description
            "A list of targets that configuration schedules are

```

```
        applied.";
    leaf object {
        type yang:xpath1.0;
        description
            "Xpath defining the data items of interest.";
    }
    uses schedules;
    container state {
        config false;
        description
            "Operational state data.";
        uses schedule-state-attributes;
    } // state

    uses schedule-config-notification;
} // target
} // configuration-schedules
}
<CODE ENDS>
```

8. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

This document registers the following namespace URI in the IETF XML registry [RFC3688]:

```
-----
URI: urn:ietf:params:xml:ns:yang:ietf-schedule
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
-----
```

This document registers the following YANG module in the YANG Module Names registry [RFC6020]:

```
-----
name:          ietf-schedule
namespace:     urn:ietf:params:xml:ns:yang:ietf-schedule
prefix:        l3te
reference:     RFC XXXX
-----
```

9. Security Considerations

The configuration, state, action and notification data defined in this document are designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and contents.

The functionality defined in this memo can potentially allow network reconnaissance; by gathering information about schedules an attacker can learn about the network policy, its temporal behavior, and future events.

The schedule YANG model defines schedules that are writable, creatable, and deletable. Therefore, this model may be considered sensitive or vulnerable in some network environments. An attacker may maliciously configure a schedule in a way that disrupts the normal behavior of the network. Furthermore, an attacker may attempt to maliciously set a schedule or a set of schedules in a way that amplifies an attack, or schedules an attack to a particularly sensitive time instant.

The use of configuration scheduling implicitly assumes that there is an underlying synchronization or time distribution mechanism. Therefore, an attack on the synchronization mechanism may compromise the configuration scheduling. The security considerations of time protocols are discussed further in [RFC7384].

10. Contributors

Tal Mizrahi

Email: talmi@marvell.com

11. References

11.1. Normative References

[IEEE1588]

IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Version 2", IEEE Standard 1588.

- [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>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [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>>.
- [RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <<https://www.rfc-editor.org/info/rfc6021>>.
- [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>>.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <<https://www.rfc-editor.org/info/rfc7384>>.
- [RFC7399] Farrel, A. and D. King, "Unanswered Questions in the Path Computation Element Architecture", RFC 7399, DOI 10.17487/RFC7399, October 2014, <<https://www.rfc-editor.org/info/rfc7399>>.
- [RFC7758] Mizrahi, T. and Y. Moses, "Time Capability in NETCONF", RFC 7758, DOI 10.17487/RFC7758, February 2016, <<https://www.rfc-editor.org/info/rfc7758>>.
- [I-D.birrane-dtn-ama]
Birrane, E., "Asynchronous Management Architecture", draft-birrane-dtn-ama-06 (work in progress), October 2017.

[I-D.ietf-netconf-subscribed-notifications]

Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and
A. Tripathy, "Custom Subscription to Event Streams",
draft-ietf-netconf-subscribed-notifications-09 (work in
progress), January 2018.

[I-D.ietf-netconf-yang-push]

Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-
Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore
Subscription", draft-ietf-netconf-yang-push-14 (work in
progress), February 2018.

11.2. Informative References

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG
Data Model Documents", RFC 6087, DOI 10.17487/RFC6087,
January 2011, <<https://www.rfc-editor.org/info/rfc6087>>.

Authors' Addresses

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA 22102
USA

EMail: Xufeng_Liu@jabil.com

Igor Bryskin
Huawei Technologies

EMail: Igor.Bryskin@huawei.com

Vishnu Pavan Beeram
Juniper Networks

EMail: vbeeram@juniper.net

Tarek Saad
Cisco Systems Inc

EMail: tsaad@cisco.com

Himanshu Shah
Ciena

EMail: hshah@ciena.com

Oscar Gonzalez de Dios
Telefonica

EMail: oscar.gonzalezdedios@telefonica.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 27, 2017

A. Lindem, Ed.
Cisco Systems
L. Berger, Ed.
LabN Consulting, L.L.C.
D. Bogdanovic

C. Hopps
Deutsche Telekom
July 26, 2016

Network Device YANG Organizational Models
draft-rtgyangdt-rtgwg-device-model-05

Abstract

This document presents an approach for organizing YANG models in a comprehensive structure that may be used to configure and operate network devices. The structure is itself represented as a YANG model, with all of the related component models logically organized in a way that is operationally intuitive, but this model is not expected to be implemented. The identified component modules are expected to be defined and implemented on common network devices.

This document is derived from work submitted to the IETF by members of the informal OpenConfig working group of network operators and is a product of the Routing Area YANG Architecture design team.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 27, 2017.

Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Status of Work and Open Issues	4
2. Module Overview	5
2.1. Interface Model Components	7
2.2. System Management	9
2.3. Network Services	10
2.4. OAM Protocols	11
2.5. Routing	11
2.6. MPLS	12
3. Security Considerations	12
4. IANA Considerations	13
5. Network Device Model Structure	13
6. References	19
6.1. Normative References	19
6.2. Informative References	20
Appendix A. Acknowledgments	21
Authors' Addresses	22

1. Introduction

"Operational Structure and Organization of YANG Models"
[I-D.openconfig-netmod-model-structure], highlights the value of organizing individual, self-standing YANG [RFC6020] models into a more comprehensive structure. This document builds on that work and presents a derivative structure for use in representing the networking infrastructure aspects of physical and virtual devices. [I-D.openconfig-netmod-model-structure] and earlier versions of this document presented a single device-centric model root, this document no longer contains this element. Such an element would have translated to a single device management model that would be the root

of all other models and was judged to be overly restrictive in terms of definition, implementation, and operation.

The document presents a notional network device YANG organizational structure that provides a conceptual framework for the models that may be used to configure and operate network devices. The structure is itself presented as a YANG module, with all of the related component modules logically organized in a way that is operationally intuitive. This network device model is not expected to be implemented, but rather provide as context for the identified representative component modules with are expected to be defined, and supported on typical network devices.

This document refers to two new modules that are expected to be implemented. These models are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Two forms of resource partitioning are referenced:

The first form provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined in [LNE-MODEL]. The module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices; each accessed independently. Optionally, and when supported by the implementation, they may also be accessed from the host system. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

The second form provides support what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026]. In this form of resource partitioning multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as Network Instances and are supported by the network-instance module defined in [NI-MODEL]. Configuration and operation of each network-instance is always via the network device and the network-instance module.

This document was motivated by, and derived from, [I-D.openconfig-netmod-model-structure]. The requirements from that document have been combined with the requirements from "Consistent Modeling of Operational State Data in YANG",

[I-D.openconfig-netmod-opstate], into "NETMOD Operational State Requirements", [I-D.ietf-netmod-opstate-reqs]. This document is aimed at the requirement related to a common model-structure, currently Requirement 7, and also aims to provide a modeling base for Operational State representation.

The approach taken in this (and the original) document is to organize the models describing various aspects of network infrastructure, focusing on devices, their subsystems, and relevant protocols operating at the link and network layers. The proposal does not consider a common model for higher level network services. We focus on the set of models that are commonly used by network operators, and suggest a corresponding organization.

A significant portion of the text and model contained in this document was taken from the -00 of [I-D.openconfig-netmod-model-structure].

1.1. Status of Work and Open Issues

This version of the document and structure are a product of the Routing Area YANG Architecture design team and is very much a work in progress rather than a final proposal. This version is a major change from the prior version and this change was enabled by the work on the previously mentioned Schema Mount.

Schema Mount enables a dramatic simplification of the presented device model, particularly for "lower-end" devices which are unlikely to support multiple network instances or logical network elements. Should structural-mount/YSDL not be available, the more explicit tree structure presented in earlier versions of this document will need to be utilized.

The top open issues are:

1. This document will need to match the evolution and standardization of [I-D.openconfig-netmod-opstate] or [I-D.ietf-netmod-opstate-reqs] by the Netmod WG.
2. Interpretation of different policy containers requires clarification.
3. It may make sense to use the identityref structuring with hardware and QoS model.
4. Which document(s) define the base System management, network services, and oam protocols modules is TBD. This includes the

possibility of simply using RFC7317 in place of the presented System management module.

5. The model will be updated once the "opstate" requirements are addressed.

2. Module Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g., routers, firewalls and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

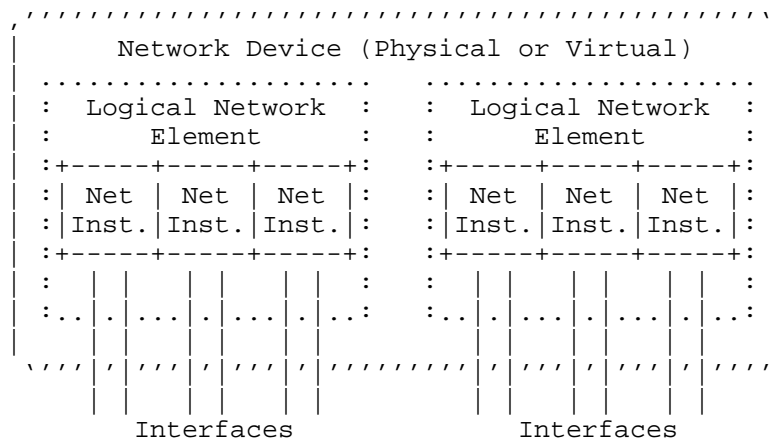


Figure 1: Module Element Relationships

A model for LNEs is described in [LNE-MODEL] and the model for network instances is covered in [NI-MODEL].

The presented notional network device module can itself be thought of as a "meta-model" as it describes the relationships between individual models. We choose to represent it also as a simple YANG module consisting of other models, which are in fact independent top

level individual models. Although it is never expected to be implemented.

The presented modules do not follow the hierarchy of any Particular implementation, and hence is vendor-neutral. Nevertheless, the structure should be familiar to network operators and also readily mapped to vendor implementations.

The overall structure is:

```
module: ietf-network-device
  +--rw modules-state          [I-D.ietf-netconf-yang-library]
  |
  +--rw interfaces             [RFC7223]
  +--rw hardware
  +--rw qos
  |
  +--rw system-management      [RFC7317 or derived]
  +--rw network-services
  +--rw oam-protocols
  |
  +--rw routing                [I-D.ietf-netmod-routing-cfg]
  +--rw mpls
  +--rw ieee-dot1Q
  |
  +--rw acls                   [I-D.ietf-netmod-acl-model]
  +--rw key-chains             [I-D.ietf-rtgwg-yang-key-chain]
  |
  +--rw logical-network-elements [I-D.rtgyangdt-rtgwg-lne-model]
  +--rw network-instances      [I-D.rtgyangdt-rtgwg-ni-model]
```

The network device is composed of top level modules that can be used to configure and operate a network device. (This is a significant difference from earlier versions of this document where there was a strict model hierarchy.) Importantly the network device structure is the same for a physical network device or a logical network device, such as those instantiated via the logical-network-element model. Extra spacing is included to denote different types of modules included.

YANG library [I-D.ietf-netconf-yang-library] is included as it used to identify details of the top level modules supported by the (physical or logical) network device. The ability to identify supported modules is particularly important for LNEs which may have a set of supported modules which differs from the set supported by the host network device.

The interface management model [RFC7223] is included at the top level. The hardware module is a placeholder for a future device-specific configuration and operational state data model. For example, a common structure for the hardware model might include chassis, line cards, and ports, but we leave this unspecified. The quality of service (QoS) section is also a placeholder module for device configuration and operational state data which relates to the treatment of traffic across the device. This document references augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be included as part of the definition of the future hardware and QoS modules.

System management, network services, and oam protocols represent new top level modules that are used to organize data models of similar functions. Additional information on each is provided below.

The routing and MPLS modules provide core support for the configuration and operation of a devices control plane and data plane functions. IEEE dot1Q [IEEE-8021Q] is an example of another module that provides similar functions for VLAN bridging, and other similar modules are also possible. Each of these modules is expected to be LNE and NI unaware, and to be instantiated as needed as part of the LNE and NI configuration and operation supported by the logical-network-element and network-instance modules. (Note that this is a change from [I-D.ietf-netmod-routing-cfg] which is currently defined with VRF/NI semantics.)

The access control list (ACL) and key chain modules are included as examples of other top level modules that may be supported by a network device.

The logical network element and network instance modules enable LNEs and NIs respectively and are defined below.

2.1. Interface Model Components

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [RFC7223].

The logical-network-element and network-instance modules defined in [LNE-MODEL] and [NI-MODEL] augment the existing interface management model in two ways: The first, by the logical-network-element module, adds an identifier which is used on physical interface types to identify an associated LNE. The second, by the network-instance module, adds a name which is used on interface or sub-interface types to identify an associated network instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [RFC7277].

The interface related augmentations are as follows:

```
module: ietf-logical-network-element
augment /if:interfaces/if:interface:
  +--rw bind-lne-name?   string

module: ietf-network-instance
augment /if:interfaces/if:interface:
  +--rw bind-network-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv4:
  +--rw bind-network-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv6:
  +--rw bind-network-instance-name?   string
```

The following is an example of envisioned combined usage. The interfaces container includes a number of commonly used components as examples:

```

+--rw if:interfaces
|   +--rw interface* [name]
|       +--rw name                               string
|       +--rw lne:bind-lne-name?                 string
|       +--rw ethernet
|           +--rw ni:bind-network-instance-name? string
|           +--rw aggregates
|           +--rw rstp
|           +--rw lldp
|           +--rw ptp
|       +--rw vlans
|       +--rw tunnels
|       +--rw ipv4
|           +--rw ni:bind-network-instance-name? string
|           +--rw arp
|           +--rw icmp
|           +--rw vrrp
|           +--rw dhcp-client
|       +--rw ipv6
|           +--rw ni:bind-network-instance-name? string
|           +--rw vrrp
|           +--rw icmpv6
|           +--rw nd
|           +--rw dhcpv6-client

```

The [RFC7223] defined interface model is structured to include all interfaces in a flat list, without regard to logical or virtual instances (e.g., VRFs) supported on the device. The bind-lne-name and bind-network-instance-name leaves provide the association between an interface and its associated LNE and NI (e.g., VRF or VSI).

2.2. System Management

[Editor's note: need to discuss and resolve relationship between this structure and RFC7317 and determine if 7317 is close enough to simply use as is.]

System management is expected to reuse definitions contained in [RFC7317]. It is expected to be instantiated per device and LNE. Its structure is shown below:

```

module: ietf-network-device
+--rw system-management
|   +--rw system-management-global
|   +--rw system-management-protocol* [type]
|       +--rw type      identityref

```


System-management-global is used for configuration information and state that is independent of a particular management protocol. System-management-protocol is a list of management protocol specific elements. The type-specific sub-modules are expected to be defined.

The following is an example of envisioned usage:

```
module: ietf-network-device
  +--rw system-management
    +--rw system-management-global
      |   +--rw statistics-collection
      |   ...
    +--rw system-management-protocol* [type]
      |   +--rw type=syslog
      |   +--rw type=dns
      |   +--rw type=ntp
      |   +--rw type=ssh
      |   +--rw type=tacacs
      |   +--rw type=snmp
      |   +--rw type=netconf
```

2.3. Network Services

A device may provide different network services to other devices, for example a device may act as a DHCP server. The model may be instantiated per device, LNE, and NI. An identityref is used to identify the type of specific service being provided and its associated configuration and state information. The defined structure is as follows:

```
module: ietf-network-device
  +--rw network-services
    |   +--rw network-service* [type]
    |   +--rw type identityref
```

The following is an example of envisioned usage: Examples shown below include a device-based Network Time Protocol (NTP) server, a Domain Name System (DNS) server, and a Dynamic Host Configuration Protocol (DHCP) server:

```
module: ietf-network-device
  +--rw network-services
    +--rw network-service* [type]
      |   +--rw type=ntp-server
      |   +--rw type=dns-server
      |   +--rw type=dhcp-server
```

2.4. OAM Protocols

OAM protocols that may run within the context of a device are grouped within the `oam-protocols` model. The model may be instantiated per device, LNE, and NI. An `identityref` is used to identify the information and state that may relate to a specific OAM protocol. The defined structure is as follows:

```
module: ietf-network-device
  +--rw oam-protocols
    +--rw oam-protocol* [type]
      +--rw type      identityref
```

The following is an example of envisioned usage. Examples shown below include Bi-directional Forwarding Detection (BFD), Ethernet Connectivity Fault Management (CFM), and Two-Way Active Measurement Protocol (TWAMP):

```
module: ietf-network-device
  +--rw oam-protocols
    +--rw oam-protocol* [type]
      +--rw type=bfd
      +--rw type=cfm
      +--rw type=twamp
```

2.5. Routing

Routing protocol and IP forwarding configuration and operation information is modeled via a routing model, such as the one defined in [I-D.ietf-netmod-routing-cfg].

The routing module is expected to include all IETF defined control plane protocols, such as BGP, OSPF, LDP and RSVP-TE. It is also expected to support configuration and operation of or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Finally, policy is expected to be represented within each control plane protocol and RIB.

The anticipated structure is as follows:

```

module: ietf-network-device
  +--rw rt:routing [I-D.ietf-netmod-routing-cfg]
  +--rw control-plane-protocol* [type]
  |   +--rw type identityref
  |   +--rw policy
  +--rw rib* [name]
  |   +--rw name string
  |   +--rw description? string
  |   +--rw policy

```

2.6. MPLS

MPLS data plane related information is grouped together, as with the previously discussed modules, is unaware of VRFs/NIs. The model may be instantiated per device, LNE, and NI. MPLS control plane protocols are expected to be included in Section 2.5. MPLS may reuse and build on [I-D.openconfig-mpls-consolidated-model] or other emerging models and has an anticipated structure as follows:

```

module: ietf-network-device
  +--rw mpls
  |   +--rw global
  |   +--rw lsps* [type]
  |   |   +--rw type identityref

```

Type refers to LSP type, such as static, traffic engineered or routing congruent. The following is an example of such usage:

```

module: ietf-network-device
  +--rw mpls
  |   +--rw global
  |   +--rw lsps* [type]
  |   |   +--rw type=static
  |   |   +--rw type=constrained-paths
  |   |   +--rw type=igp-congruent

```

3. Security Considerations

The network-device model structure described in this document does not define actual configuration and state data, hence it is not directly responsible for security risks.

Each of the component models that provide the corresponding configuration and state data should be considered sensitive from a security standpoint since they generally manipulate aspects of network configurations. Each component model should be carefully evaluated to determine its security risks, along with mitigations to reduce such risks.

LNE portion is TBD

NI portion is TBD

4. IANA Considerations

This YANG model currently uses a temporary ad-hoc namespace. If it is placed or redirected for the standards track, an appropriate namespace URI will be registered in the "IETF XML Registry" [RFC3688]. The YANG structure modules will be registered in the "YANG Module Names" registry [RFC6020].

5. Network Device Model Structure

```
<CODE BEGINS> file "ietf-network-device@2016-05-01.yang"
module ietf-network-device {

    yang-version "1";

    // namespace
    namespace "urn:ietf:params:xml:ns:yang:ietf-network-device";

    prefix "nd";

    // import some basic types

    // meta
    organization "IETF RTG YANG Design Team Collaboration
                  with OpenConfig";

    contact
        "Routing Area YANG Architecture Design Team -
        <rtg-dt-yang-arch@ietf.org>";

    description
        "This module describes a model structure for YANG
        configuration and operational state data models. Its intent is
        to describe how individual device protocol and feature models
        fit together and interact.";

    revision "2016-05-01" {
        description
            "IETF Routing YANG Design Team Meta-Model";
        reference "TBD";
    }

    // extension statements
```

```
// identity statements

identity oam-protocol-type {
    description
        "Base identity for derivation of OAM protocols";
}

identity network-service-type {
    description
        "Base identity for derivation of network services";
}

identity system-management-protocol-type {
    description
        "Base identity for derivation of system management
        protocols";
}

identity oam-service-type {
    description
        "Base identity for derivation of Operations,
        Administration, and Maintenance (OAM) services.";
}

identity control-plane-protocol-type {
    description
        "Base identity for derivation of control-plane protocols";
}

identity mpls-lsp-type {
    description
        "Base identity for derivation of MPLS LSP types";
}

// typedef statements

// grouping statements

grouping ribs {
    description
        "Routing Information Bases (RIBs) supported by a
        network-instance";
    container ribs {
        description
            "RIBs supported by a network-instance";
        list rib {
            key "name";
            min-elements "1";
        }
    }
}
```

```
description
    "Each entry represents a RIB identified by the
    'name' key. All routes in a RIB must belong to the
    same address family.

    For each routing instance, an implementation should
    provide one system-controlled default RIB for each
    supported address family.";
leaf name {
    type string;
    description
        "The name of the RIB.";
}
reference "draft-ietf-netmod-routing-cfg";
leaf description {
    type string;
    description
        "Description of the RIB";
}
// Note that there is no list of interfaces within
container policy {
    description "Policy specific to RIB";
}
}
}

// top level device definition statements
container ietf-yang-library {
    description
        "YANG Module Library as defined in
        draft-ietf-netconf-yang-library";
}

container interfaces {
    description
        "Interface list as defined by RFC7223/RFC7224";
}

container hardware {
    description
        "Hardware / vendor-specific data relevant to the platform.
        This container is an anchor point for platform-specific
        configuration and operational state data. It may be further
        organized into chassis, line cards, ports, etc. It is
        expected that vendor or platform-specific augmentations
        would be used to populate this part of the device model";
}
```

```
container qos {
  description "QoS features, for example policing, shaping, etc.";
}

container system-management {
  description
    "System management for physical or virtual device.";
  container system-management-global {
    description "System management - with reuse of RFC 7317";
  }
  list system-management-protocol {
    key "type";
    leaf type {
      type identityref {
        base system-management-protocol-type;
      }
      mandatory true;
      description
        "Syslog, ssh, TACAC+, SNMP, NETCONF, etc.";
    }
    description "List of system management protocol
      configured for a logical network
      element.";
  }
}

container network-services {
  description
    "Container for list of configured network
    services.";
  list network-service {
    key "type";
    description
      "List of network services configured for a
      network instance.";
    leaf type {
      type identityref {
        base network-service-type;
      }
      mandatory true;
      description
        "The network service type supported within
        a network instance, e.g., NTP server, DNS
        server, DHCP server, etc.";
    }
  }
}
```

```
container oam-protocols {
  description
    "Container for configured OAM protocols.";
  list oam-protocol {
    key "type";
    leaf type {
      type identityref {
        base oam-protocol-type;
      }
      mandatory true;
      description
        "The Operations, Administration, and
        Maintenance (OAM) protocol type, e.g., BFD,
        TWAMP, CFM, etc.";
    }
    description
      "List of configured OAM protocols.";
  }
}

container routing {
  description
    "The YANG Data Model for Routing Management revised to be
    Network Instance / VRF independent. ";
  // Note that there is no routing or network instance
  list control-plane-protocol {
    key "type";
    description
      "List of control plane protocols configured for
      a network instance.";
    leaf type {
      type identityref {
        base control-plane-protocol-type;
      }
      description
        "The control plane protocol type, e.g., BGP,
        OSPF IS-IS, etc";
    }
    container policy {
      description
        "Protocol specific policy,
        reusing [RTG-POLICY]";
    }
  }
  list rib {
    key "name";
    min-elements "1";
    description
```


"Each entry represents a RIB identified by the 'name' key. All routes in a RIB must belong to the same address family.

For each routing instance, an implementation should provide one system-controlled default RIB for each supported address family."

```

leaf name {
  type string;
  description
    "The name of the RIB.";
}
reference "draft-ietf-netmod-routing-cfg";
leaf description {
  type string;
  description
    "Description of the RIB";
}
// Note that there is no list of interfaces within
container policy {
  description "Policy specific to RIB";
}
}
}

container mpls {
  description "MPLS and TE configuration";
  container global {
    description "Global MPLS configuration";
  }
  list lsps {
    key "type";
    description
      "List of LSP types.";
    leaf type {
      type identityref {
        base mpls-lsp-type;
      }
      mandatory true;
      description
        "MPLS and Traffic Engineering protocol LSP types,
        static, LDP/SR (igp-congruent),
        RSVP TE (constrained-paths) , etc.";
    }
  }
}

container ieee-dot1Q {

```

```
    description
      "The YANG Data Model for VLAN bridges as defined by the IEEE";
  }

  container ietf-acl {
    description "Packet Access Control Lists (ACLs) as specified
                in draft-ietf-netmod-acl-model";
  }

  container ietf-key-chain {
    description "Key chains as specified in
                draft-ietf-rtgwg-yang-key-chain";
  }

  container logical-network-element {
    description
      "This module is used to support multiple logical network
       elements on a single physical or virtual system.";
  }

  container network-instance {
    description
      "This module is used to support multiple network instances
       within a single physical or virtual device.  Network
       instances are commonly know as VRFs (virtual routing
       and forwarding) and VSIs (virtual switching instances).";
  }
  // rpc statements

  // notification statements
}
<CODE ENDS>
```

6. References

6.1. Normative References

[LNE-MODEL]

Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic,
"Logical Network Element Model", draft-rtgyangdt-rtgwg-
lne-model-00.txt (work in progress), May 2016.

[NI-MODEL]

Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic,
"Network Instance Model", draft-rtgyangdt-rtgwg-ni-model-
00.txt (work in progress), May 2016.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <<http://www.rfc-editor.org/info/rfc4026>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<http://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

6.2. Informative References

- [I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library-05 (work in progress), April 2016.
- [I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-03 (work in progress), January 2016.
- [I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-20 (work in progress), October 2015.
- [I-D.openconfig-mpls-consolidated-model]
George, J., Fang, L., eric.osborne@level3.com, e., and R. Shakir, "MPLS / TE Model for Service Provider Networks", draft-openconfig-mpls-consolidated-model-02 (work in progress), October 2015.

[I-D.openconfig-netmod-model-structure]

Shaikh, A., Shakir, R., D'Souza, K., and L. Fang,
"Operational Structure and Organization of YANG Models",
draft-openconfig-netmod-model-structure-00 (work in
progress), March 2015.

[I-D.openconfig-netmod-opstate]

Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling
of Operational State Data in YANG", draft-openconfig-
netmod-opstate-01 (work in progress), July 2015.

[IEEE-8021Q]

Holness, M., "IEEE 802.1Q YANG Module Specifications",
IEEE-Draft [http://www.ieee802.org/1/files/public/docs2015/
new-mholness-yang-8021Q-0515-v04.pdf](http://www.ieee802.org/1/files/public/docs2015/new-mholness-yang-8021Q-0515-v04.pdf), May 2015.

Appendix A. Acknowledgments

This document is derived from draft-openconfig-netmod-model-structure-00. We thank the Authors of that document and acknowledge their indirect contribution to this work. The authors include: Anees Shaikh, Rob Shakir, Kevin D'Souza, Luyuan Fang, Qin Wu, Stephane Litkowski and Gang Yan.

This work was discussed in and produced by the Routing Area Yang Architecture design team. Members at the time of writing included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Gang Yan.

The identityref approach was proposed by Mahesh Jethanandani.

The RFC text was produced using Marshall Rose's xml2rfc tool.

Authors' Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org